Microsoft

OFFICIAL MICROSOFT LEARNING PRODUCT

10267A

Lab Instructions and Lab Answer Key: Introduction to Web Development with Microsoft® Visual Studio® 2010

Product Number: 10267A

Part Number: X17-01978

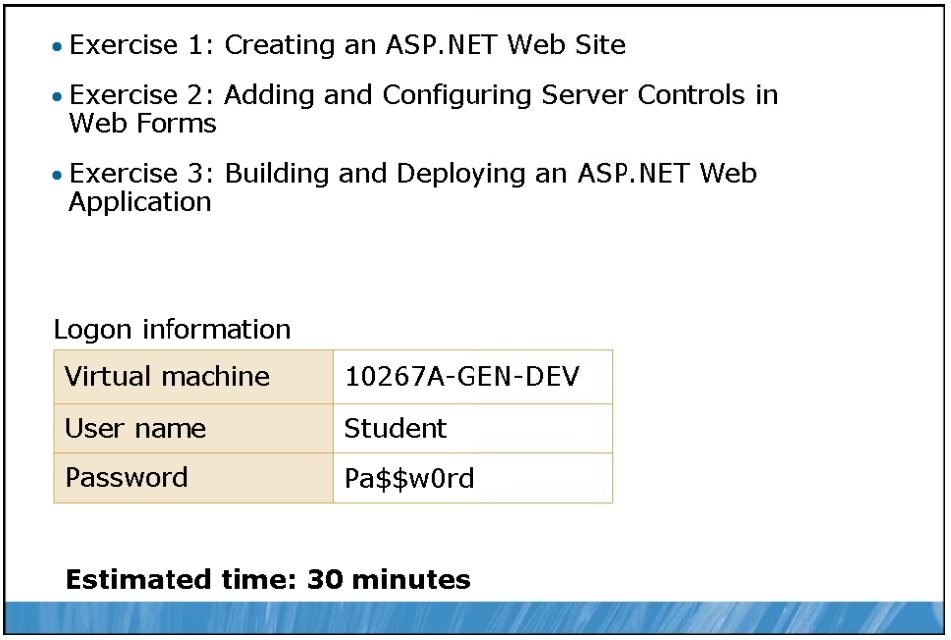
Released: 07/2010

Module 2

Lab Instructions: Creating Web Applications by Using Microsoft® Visual Studio® 2010 and Microsoft .NET–Based Languages (Visual Basic)

Contents:

Lab: Creating Web Applications by Using Microsoft Visual Studio 2010 and Microsoft .NET–Based Languages



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in **Section 1** of the lab page. If you are using Visual C# as your programming language, refer to the steps provided in **Section 2** of the lab page.

Introduction

In this lab, you will create a simple ASP.NET Web site project, then add a server control, to a Web Form, and then configure its properties. In addition, you will build and deploy an ASP.NET Web site.

Objectives

After completing this lab, you will be able to:

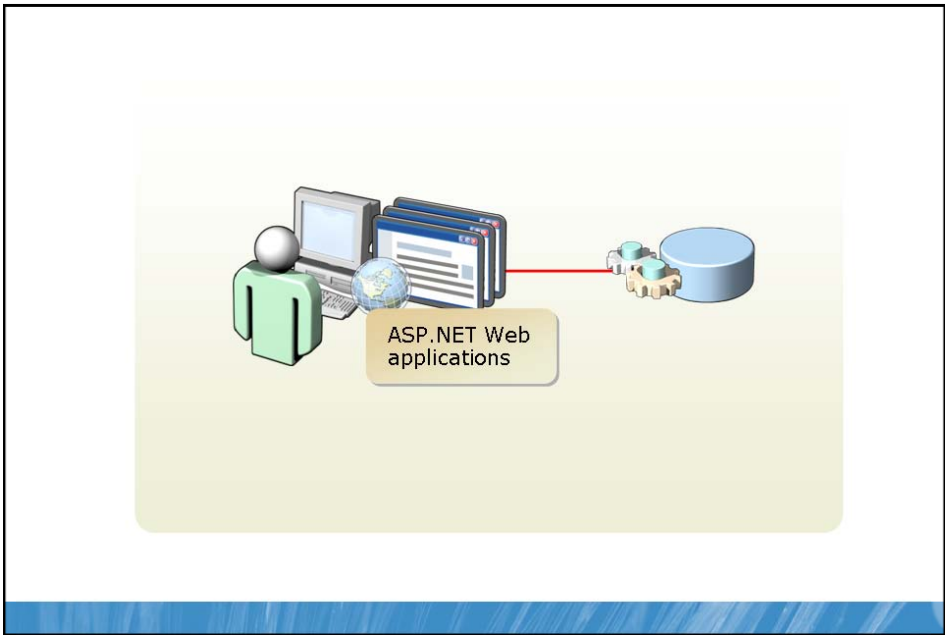
1. Create a simple ASP.NET Web site project.
2. Add a server control to a Web Form and configure its properties.
3. Build and deploy an ASP.NET Web site.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following user name and password:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage its customer information.

Your organization decides to create a Web site for fast and easy interaction with its customers. In addition to external Customers Web site, your organization plans to create a Web site to manage customer data and services in ASP.NET.

You are assigned the task of creating the Web site by using the ASP.NET Web site template, and then deploying the Web site to an IIS virtual directory.

Section 1: Visual Basic

Exercise 1: Creating an ASP.NET Web Site

The main tasks for this exercise are as follows:

1. Create an empty ASP.NET Web site.
2. Use a static port with the ASP.NET Development server.
3. Save the Solution file.
4. Add an existing CSS file to the Web site.

▶ Task 1: Create an empty ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Create the empty **CustomerManagement** Web site with the following settings, by using the **New Web Site** dialog box:
  - Template: **Empty ASP.NET Web Site**
  - Name: **CustomerManagement**
  - Location: **File system**
  - Path: **D:\Labfiles\Starter\M2\VB\CustomerManagement**
  - Language: **Visual Basic**

▶ Task 2: Use a static port with the ASP.NET Development server

- In Solution Explorer, click **D:\Labfiles\Starter\M2\VB \CustomerManagement**.
- In the Properties window, in the **Use dynamic ports** list, click **False**.
- In the Properties window, in the **Port number box**, type **1111**, and then press ENTER.

Note: It may take a few seconds before the **Port number** property is ready for editing after setting the **Use dynamic ports** property.

▶ Task 3: Save the Solution file

- In Solution Explorer, click Solution '**CustomerManagement' (1 project)**, and then save the solution file as **D:\Labfiles\Starter\M2\VB \CustomerManagement.sln**, by using the **Save As** command on the **File** menu.

▶ Task 4: Add an existing CSS file to the Web site

- Create a folder named **Styles**, in the **CustomerManagement** Web site
- Add the D:\Labfiles\Starter\M2\Styles\Site.css file to the Styles folder, by using the **Add Existing Item** dialog box.

Results: After completing this exercise, you will have created a file system–based ASP.NET site, and added styles to the Web site.

Exercise 2: Adding and Configuring Server Controls in Web Forms

The main tasks for this exercise are as follows:

1. Add a default Web Form to the Web site.
2. Close Visual Studio 2010.
3. Add the application title to the default Web Form.
4. Set the control properties of the default Web Form.
5. Apply the predefined style to the Web Form.

▶ Task 1: Add a default Web Form to the Web site

- Add the Default.aspx Web Form to the **CustomerManagement** Web site by using the **Add New Item** dialog box.

Note: The Default Web Form displays in Source view, where you can notice that the elements such as form, div, and body, are empty.

▶ Task 2: Close Visual Studio 2010

- Save modified files.
- Close Visual Studio 2010.

▶ Task 3: Add the application title to the default Web Form

- Open Microsoft Visual Studio 2010 as an administrator by using the **Run as administrator** command on the context menu.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M2\VB folder, by using the **Open Project** dialog box.
- Add a **Literal** control from the Toolbox to the **div** element in the Default.aspx window, and set its **Text** attribute to **Customer Management**, and **ID** property to **AppTitleLiteral**, by using the Properties window.

- Save the changes, and view the default Web Form in a Web browser, by using the **View in Browser** context menu command.

Note: You can now view the text that you have entered in the **Literal** control.

▶ Task 4: Set the control properties of the default Web Form

- Open the default Web Form in Design view.
- Set the **Visible** property of the **Literal** control to **False** by using the Properties window, and then save the changes.
- View the Web Form in the browser, and then view the source rendered to the browser, by using the Source command on the View menu of Internet Explorer.
- Close the open Internet Explorer windows.
- Set the **Visible** property of the **Literal** control to **True**.
- Set the **Styles** property of the **div** element by using the following properties in the **Modify Style** dialog box, which are accessible from the **Style** property in the Property window:
  - Font-family: **Trebuchet MS**
  - Font-size: **22**
  - Color: **Gray**
- Save the changes, and view the default Web Form in a Web browser.

▶ Task 5: Apply the predefined style to the Web Form

- View the changes in the Source view.
- Add a reference to the **Site.css** file in the **Styles** folder from within the head element by dragging the **Styles/Site.css** file to the Default.aspx window, next to the **title** element.
- Set the **Class** property of the **div** element to **appTitle**, and then remove the specific styles applied for the **Styles** property, by using the Properties window.
- Save the changes, and view the default Web Form in a Web browser.

Note: You can now view the changes that you have made to the **Literal** control.

Result: After completing this exercise, you will have designed the initial version of the default Web Form for your Web site.

Exercise 3: Building and Deploying an ASP.NET Web Application

The main task for this exercise is to build and deploy the CustomerManagement Web site.

▶ Task 1: Build and deploy the CustomerManagement Web site

- Open the Copy Web Site tool by selecting the Web site in Solution Explorer, and then using the **Website** menu.
- Connect to local IIS in the Copy Web Site tool.
- Create a new virtual directory below the Default Web Site, by selecting **Default Web Site**, and then clicking the **Create New Virtual Directory** button. Use the following settings:
  - Alias name: **CustomerManagement**
  - Folder: **C:\inetpub\wwwroot\CustomerManagement**
- Select and open the new virtual directory in the **Open Web Site** dialog box.
- Copy the **CustomerManagement** Web site to the new virtual directory on the local IIS, by selecting all files and folders in the left pane of the Copy Web Site tool, and then clicking the **Copy selected files from source to remote web site** button.
- View the deployed Web site in Internet Explorer at the address **http://localhost:1112/CustomerManagement**.

▶ Task 2: Turn off the virtual machine and revert the changes.

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

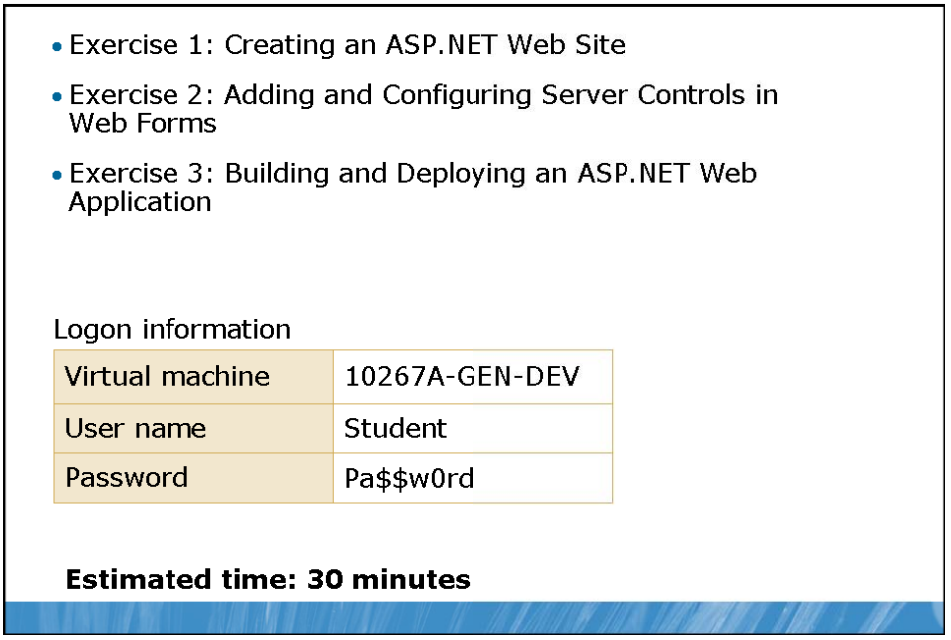
Result: After completing this exercise, you will have built and deployed the **CustomerManagement** Web site to the local IIS.

Module 2

Lab Instructions: Creating Web Applications by Using Microsoft® Visual Studio® 2010 and Microsoft .NET–Based Languages (Visual C#)

Contents:

Lab: Creating Web Applications by Using Microsoft Visual Studio 2010 and Microsoft .NET–Based Languages



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in **Section 1** of the lab page. If you are using Visual C# as your programming language, refer to the steps provided in **Section 2** of the lab page.

Introduction

In this lab, you will create a simple ASP.NET Web site project, then add a server control, to a Web Form, and then configure its properties. In addition, you will build and deploy an ASP.NET Web site.

Objectives

After completing this lab, you will be able to:

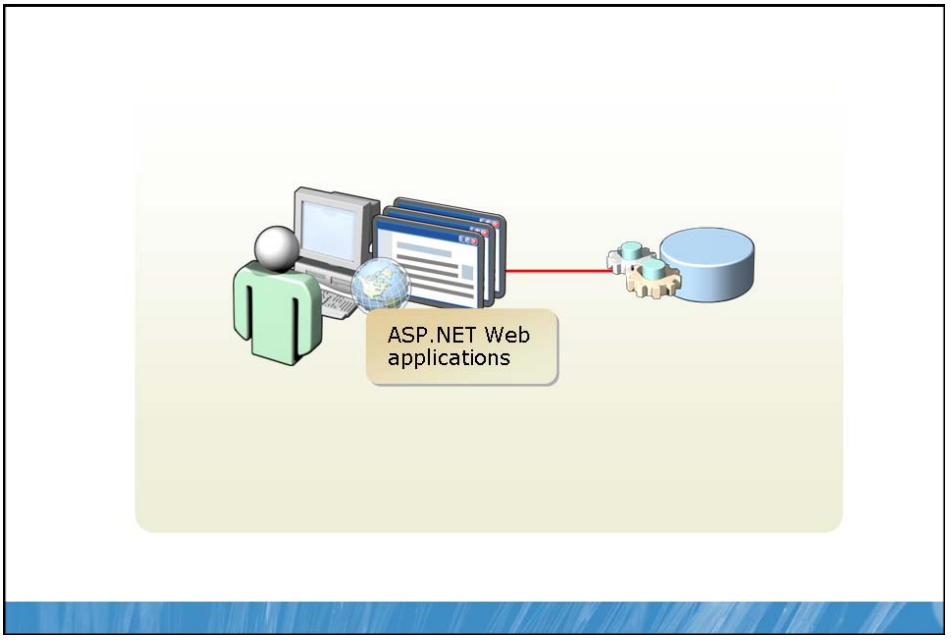
1. Create a simple ASP.NET Web site project.
2. Add a server control to a Web Form and configure its properties.
3. Build and deploy an ASP.NET Web site.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following user name and password:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage its customer information.

Your organization decides to create a Web site for fast and easy interaction with its customers. In addition to external Customers Web site, your organization plans to create a Web site to manage customer data and services in ASP.NET.

You are assigned the task of creating the Web site by using the ASP.NET Web site template, and then deploying the Web site to an IIS virtual directory.

Section 2: Visual C#

Exercise 1: Creating an ASP.NET Web Site

The main tasks for this exercise are as follows:

1. Create an empty ASP.NET Web site.
2. Use a static port with the ASP.NET Development server.
3. Save the Solution file.
4. Add an existing CSS file to the Web site.

▶ Task 1: Create an empty ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Create the empty **CustomerManagement** Web site with the following settings, by using the **New Web Site** dialog box:
  - Template: **Empty ASP.NET Web Site**
  - Name: **CustomerManagement**
  - Location: **File system**
  - Path: **D:\Labfiles\Starter\M2\CS\CustomerManagement**
  - Language: **Visual C#**

▶ Task 2: Use a static port with the ASP.NET Development server

- In Solution Explorer, click **D:\Labfiles\Starter\M2\CS\CustomerManagement**.
- In the Properties window, in the **Use dynamic ports** list, click **False**.
- In the Properties window, in the **Port number** box, type **1110**, and then press ENTER.

Note: It may take a few seconds before the **Port number** property is ready for editing after setting the **Use dynamic ports** property.

▶ Task 3: Save the Solution file

- In Solution Explorer, click Solution '**CustomerManagement' (1 project)**, and then save the solution file as **D:\Labfiles\Starter\M2\CS\CustomerManagement.sln**, by using the **Save As** command on the **File** menu.

▶ Task 4: Add an existing CSS file to the Web site

- Create a folder named, **Styles**, in the **CustomerManagement** Web site. Right-click the Web site in Solution Explorer, and then click **New Folder**.
- Add the **D:\Labfiles\Starter\M2\Styles\Site.css** file to the **Styles** folder, by using the **Add Existing Item** dialog box.

Results: After completing this exercise, you will have created a file system–based ASP.NET site, and added styles to the Web site.

Exercise 2: Adding and Configuring Server Controls in Web Forms

The main tasks for this exercise are as follows:

1. Add a default Web Form to the Web site.
2. Close Visual Studio 2010.
3. Add the application title to the default Web Form.
4. Set the control properties of the default Web Form.
5. Apply the predefined style to the Web Form.

▶ Task 1: Add a default Web Form to the Web site

- Add the Default.aspx Web Form to the **CustomerManagement** Web site by using the **Add New Item** dialog box.

Note: The Default Web Form displays in Source view, where you can see the elements such as form, div, and body, are empty.

▶ Task 2: Close Visual Studio 2010

- Save modified files.
- Close Visual Studio 2010.

▶ Task 3: Add the application title to the default Web Form

- Open Microsoft Visual Studio 2010 as an administrator, by using the **Run as administrator** command on the context menu.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M2\CS folder, by using the **Open Project** dialog box.
- Add a **Literal** control from the Toolbox to the **div** element in the Default.aspx window, and set its **Text** attribute to **Customer Management**, and **ID** property to **AppTitleLiteral**, by using the Properties window.
- Save the changes, and view the default Web Form in a Web browser, by using the **View in Browser** context menu command.

Note: You can now view the text that you have entered in the **Literal** control.

▶ Task 4: Set the control properties of the default Web Form

- Open the default Web Form in Design view.
- Set the **Visible** property of the **Literal** control to **False** by using the Properties window, and then save the changes.
- View the Web Form in the browser, and then view the source rendered to the browser, by using the **Source** command on the **View** menu of Internet Explorer.
- Close the open Internet Explorer windows.
- Set the **Visible** property of the **Literal** control to **True**. Set the **Styles** property of the **div** element by using the following properties in the **Modify Style** dialog box, which are accessible from the **Style** property in the Property window:
  - Font-family: **Trebuchet MS**
  - Font-size: **22**
  - Color: **Gray**
- Save the changes, and view the default Web Form in a Web browser.

▶ Task 5: Apply the predefined style to the Web Form

- View the changes in the Source view.
- Add a reference to the **Site.css** file in the **Styles** folder from within the head element by dragging the **Styles/Site.css** file to the Default.aspx window, next to the **title** element.
- Set the **Class** property of the **div** element to **appTitle**, and then remove the specific styles applied for the **Styles** property, by using the Properties window.
- Save the changes, and view the default Web Form in a Web browser.

Note: You can now view the changes that you have made to the **Literal** control.

Results: After completing this exercise, you will have designed the initial version of the default Web Form for your Web site.

Exercise 3: Building and Deploying an ASP.NET Web Application

The main task for this exercise is to build and deploy the CustomerManagement Web site.

▶ Task 1: Build and deploy the CustomerManagement Web site

- Build the **CustomerManagement** Web site, and then verify that the Web site has no errors.
- Open the Copy Web Site tool, by selecting the Web site in Solution Explorer and then using the **Web site** menu.
- Connect to local IIS in the Copy Web Site tool.
- Create a new virtual directory below the Default Web Site, by selecting **Default Web Site**, and then clicking the **Create New Virtual Directory** button. Use the following settings:
  - Alias name: **CustomerManagement**
  - Folder: **C:\inetpub\wwwroot\CustomerManagement**
- Select and open the new virtual directory in the **Open Web Site** dialog box.
- Copy the **CustomerManagement** Web site to the new virtual directory on the local IIS, by selecting all files and folders in the left pane of the Copy Web Site tool, and then clicking the **Copy selected files from source to remote web site** button.
- View the deployed Web site in Internet Explorer at the address **http://localhost:1112/CustomerManagement**.

▶ Task 2: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

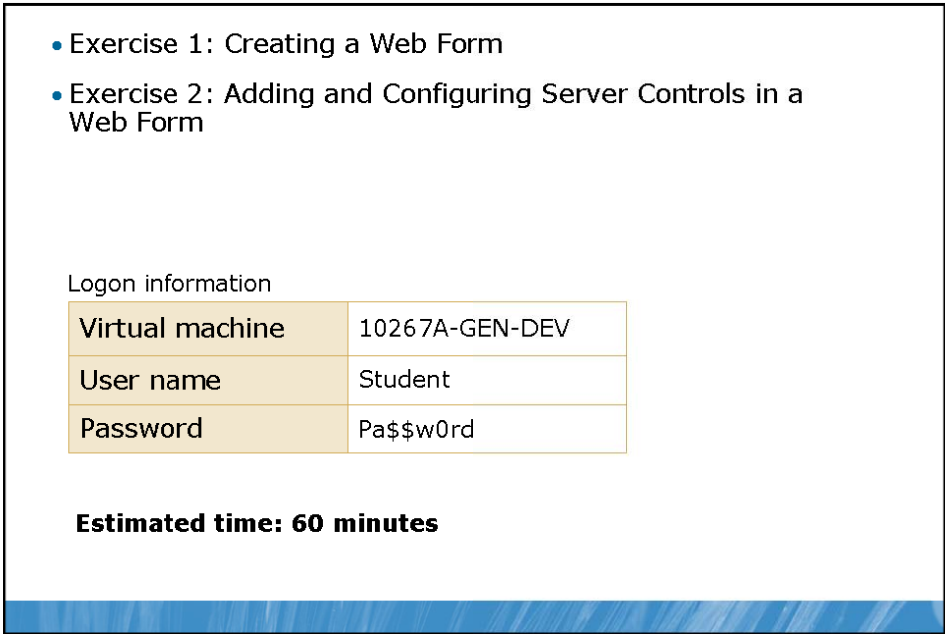
Results: After completing this exercise, you will have built and deployed the **CustomerManagement** Web site to the local IIS.

Module 3

Lab Instructions: Creating a Microsoft® ASP.NET Web Form (Visual Basic)

Contents:

Lab: Creating a Microsoft ASP.NET Web Form



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in **Section 1** of the lab page. If you are using Visual C# as your programming language, refer to the steps provided in **Section 2** of the lab page.

Introduction

In this lab, you will add a Web Form to an ASP.NET Web application, then add server controls to the Web Form, and then configure its properties.

Objectives

After completing this lab, you will be able to:

- Create a new Web Form.
- Apply a style sheet to the Web Form.
- Create a table-style layout in the Web Form.

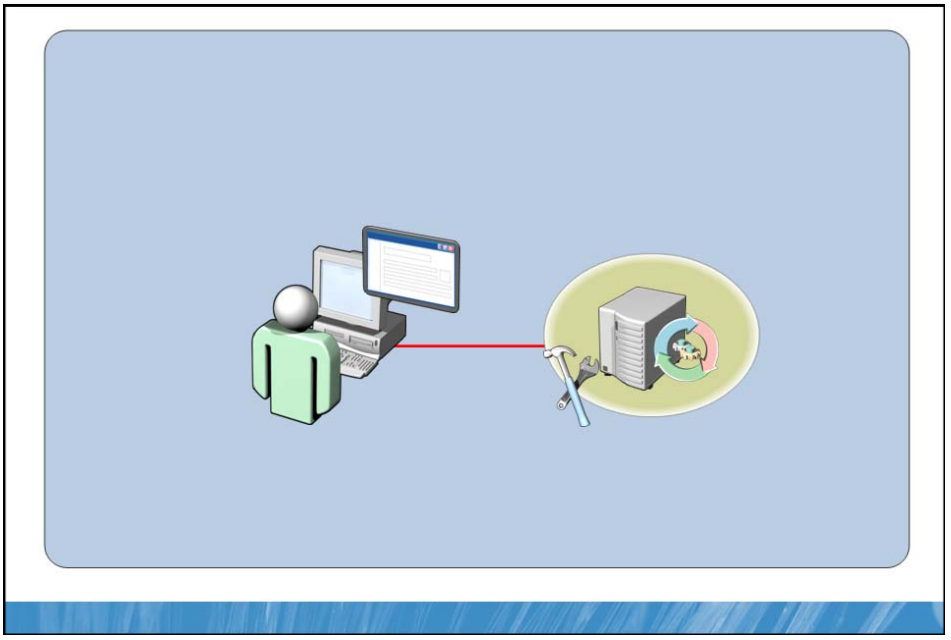
- Add a server control to the Web Form and configure its properties.
- Apply styles to the HTML elements and server controls.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
    - User name: **Student**
    - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage its customer information. Your organization has created a Web site to manage customer data and services in ASP.NET.

Contoso, Ltd wants to create an application to maintain and update its customer information. You need to customize the application to meet the specific requirements of the sales team. Updating the customer information is an ongoing process for the sales team, and the application requires updates based on the feedback from senior developers and other stakeholders. You need to build a UI that meets the defined requirements.

You must create an application and build a user interface that is easy to update and modify, by using ASP.NET server controls.

Section 1: Visual Basic

Exercise 1: Creating a Web Form

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Add a Web Form to the Web site.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M3\VB folder.

▶ Task 2: Add a Web Form to the Web site

- Add the **InsertCustomer.aspx** Web Form to the CustomerManagement Web site, by using the **Add New Item** dialog box.

Results: After completing this exercise, you will have opened the existing **CustomerManagement** Web site, and added the **InsertCustomer** Web Form.

Exercise 2: Adding and Configuring Server Controls in a Web Form

The main tasks for this exercise are as follows:

1. Add an existing style sheet to the Web Form.
2. Create a table-style layout in the Web Form.
3. Add the server controls to the Web Form and configure their basic properties.
4. Set the non-trivial control properties.
5. Apply a predefined style to the Web Form.

Note: In this exercise, you are creating a two-column table using CSS for holding various controls used for displaying and managing Customer data.

▶ Task 1: Add an existing style sheet to the Web Form

- Reference the Styles/Site.css file in the InsertCustomer Web Form, relative to the root folder, by using the Manage Styles window.
- Press CTRL+S to save InsertCustomer.aspx.
- View the styles added in the style sheet by using the Manage Styles window.

▶ Task 2: Create a table-style layout in the Web Form

- In the InsertCustomer.aspx window, place the cursor between the opening and closing **div** tags.
- Add a new **div** element from the Toolbox to the existing **div** element.

```
<div>
    <div>
    </div>
</div>
```

- Add two new **div** elements from the Toolbox to the newly added **div** element.

```
<div>
    <div>
        <div>
        </div>
        <div>
        </div>
    </div>
</div>
```

- Save the changes to the **InsertCustomer** Web Form.
- Use the Apply Styles window to apply the **customerTable** style to the outermost **div** element. Apply **div.customerTable** from the **Contextual Selectors** section of the Apply Styles window.
- Click the opening tag of the **div** element, which is a child element of the **div** element with a **class** attribute value of **customerTable**.
- Create new style in the Apply Styles window, by using the **New Style** button.
- Name the new style **div.customerTableRow** by using the **Selector** box of the **New Style** dialog box.
- Apply the style to the current document selection by using the **Apply new style to document selection** check box.
- Define the new style in the existing **Styles/Site.css** CSS file.
- Set the layout of the new style to **table-row**, by using the display list of the **Layout** category, and then close the **New Style** dialog box.
- Use the Apply Styles window to apply the **customerTableLeftCol** style to the first child **div** element of the **div** element with a **class** attribute value of **customerTableRow**.
- Use the Apply Styles window to apply the **customerTableRightCol** style to the second child **div** element of the **div** element with a **class** attribute value of **customerTableRow**.

- Add 11 more table rows by copying the existing **div** element with a **class** attribute value of **customerTableRow**.

```
<div class="customerTableRow">
    <div class="customerTableLeftCol">
    </div>
    <div class="customerTableRightCol">
    </div>
</div>
```

- Append a new **div** element from the Toolbox to the **div** element with a **class** attribute value of **customerTable**, placing the new **div** element immediately before the closing **div** tag of the **customerTable** div element.

```
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
        </div>
        <div class="customerTableRightCol">
        </div>
    </div>
    ...
    <div>
    </div>
</div>
```

- View the **InsertCustomer** Web Form in Design view.

Note: Notice that two equal-sized columns are added to the **div** element.

- View the **InsertCustomer** Web Form in Source view.
- Use the Apply Styles window to apply the **customerTableFooter** style to the last child **div** element of the **div** element with **class** attribute value of **customerTable**.
- Save the changes to the **InsertCustomer Web Form** and the Site.css file.
- In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save All**.
- View the **InsertCustomer Web Form** in a Web browser.

Note: Although you have styled the Web form by using the CSS file, notice that nothing displays. This is because the div elements are empty.

▶ Task 3: Add the server controls to the Web Form and configure their basic properties

- View the **InsertCustomer** Web Form in Design view.
- Add the **Label** control from the Toolbox to the first cell of the left column in the **div** element, change the **(ID)** property to **CustomerFirstNameLabel**, and then set the **Text** property to **First Name**:.
- Add the **TextBox** control from the Toolbox to the first cell of the right column in the **div** element, and change the **(ID)** property to **CustomerFirstNameTextBox**.
- Add the **Label** control from the Toolbox to the second cell of the left column in the **div** element, change the **(ID)** property to **CustomerLastNameLabel**, and then set the **Text** property to **Last Name:**.
- Add the **TextBox** control from the Toolbox to the second cell of the right column in the **div** element, and change the **(ID)** property to **CustomerLastNameTextBox**.
- Add the **Label** control from the Toolbox to the third cell of the left column in the **div** element, change the **(ID)** property to **CustomerAddressLabel**, and then set the **Text** property to **Address**:.
- Add the **TextBox** control from the Toolbox to the third cell of the right column in the **div** element, and change the **(ID)** property to **CustomerAddressTextBox**.
- Add the **Label** control from the Toolbox to the fourth cell of the left column in the **div** element, change the **(ID)** property to **CustomerZipCodeLabel**, and then set the **Text** property to **Zip Code:**.
- Add the **TextBox** control from the Toolbox to the fourth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerZipCodeTextBox**.
- Add the **Label** control from the Toolbox to the fifth cell of the left column in the **div** element, change the **(ID)** property to **CustomerCityLabel**, and then set the **Text** property to **City:**.
- Add the **TextBox** control from the Toolbox to the fifth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCityTextBox**.

- Add the **Label** control from the Toolbox to the sixth cell of the left column in the **div** element, change the **(ID)** property to **CustomerStateLabel**, and then set the **Text** property to **State:**.
- Add the **TextBox** control from the Toolbox to the sixth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerStateTextBox**.
- Add the **Label** control from the Toolbox to the seventh cell of the left column in the **div** element, change the **(ID)** property to **CustomerCountryLabel**, and then set the **Text** property to **Country:**.
- Add the **DropDownList** control from the Toolbox to the seventh cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCountryDropDownList**.
- Add the **Label** control from the Toolbox to the eighth cell of the left column in the **div** element, change the **(ID)** property to **CustomerPhoneLabel**, and then set the **Text** property to **Phone:**.
- Add the **TextBox** control from the Toolbox to the eighth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerPhoneTextBox**.
- Add the **Label** control from the Toolbox to the ninth cell of the left column in the **div** element, change the **(ID)** property to **CustomerEmailAddressLabel**, and then set the **Text** property to **Email Address:**.
- Add the **TextBox** control from the Toolbox to the ninth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerEmailAddressTextBox**.
- Add the **Label** control from the Toolbox to the tenth cell of the left column in the **div** element, change the **(ID)** property to **CustomerWebAddressLabel**, and then set the **Text** property to **Web Address:**.
- Add the **TextBox** control from the Toolbox to the tenth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerWebAddressTextBox**.
- Add the **Label** control from the Toolbox to the eleventh cell of the left column in the **div** element, change the **(ID)** property to **CustomerCreditLimitLabel**, and then set the **Text** property to **Credit Limit:**.
- Add the **TextBox** control from the Toolbox to the eleventh cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCreditLimitTextBox**.

- Add the **Label** control from the Toolbox to the twelfth cell of the left column in the **div** element, change the **(ID)** property to **CustomerNewsSubscriberLabel**, and then set the **Text** property to **News Subscriber:**.
- Add the **CheckBox** control from the Toolbox to the twelfth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerNewsSubscriberCheckBox**.
- Save the changes, and view the **InsertCustomer** Web Form in a Web browser.
- Add the **Button** control from the Toolbox to the footer of the table in the **div** element, change the **(ID)** property to **CustomerInsertButton**, and then set the **Text** property to **Insert**.
- Add the **Button** control from the Toolbox to the footer of the table in the **div** element, change the **(ID)** property to **CustomerCancelButton**, and then set the **Text** property to Cancel.
- Save the changes, and view the **InsertCustomer** Web Form in a Web browser.

▶ Task 4: Set the non-trivial control properties

- Set the **MaxLength** property of the **CustomerFirstNameTextBox**, **CustomerAddressTextBox**, and **CustomerEmailAddressTextBox** controls to **50**.
- Set the **MaxLength** property of the **CustomerLastNameTextBox**, **CustomerCityTextBox**, **CustomerStateTextBox**, and **CustomerPhoneTextBox** controls to **30**.
- Set the **MaxLength** property of the **CustomerZipCodeTextBox** and **CustomerCreditLimitTextBox** controls to **10**.
- Open the **InsertCustomer** Web Form in the Source view.
- Set the **MaxLength** property of the **CustomerWebAddressTextBox** control to **80**.
- Save the changes and view the **InsertCustomer** Web Form in a Web browser.

▶ Task 5: Apply a predefined style to the Web Form

- Apply the predefined template style to the **body** element of the **InsertCustomer** Web form.
- Save the changes, and view the **InsertCustomer** Web Form in a Web browser.

• Task 6: Turn off the virtual machine and revert the changes

- Turn off the 10267A-GEN-DEV virtual machine.
- Revert the changes made to the 10267A-GEN-DEV virtual machine.

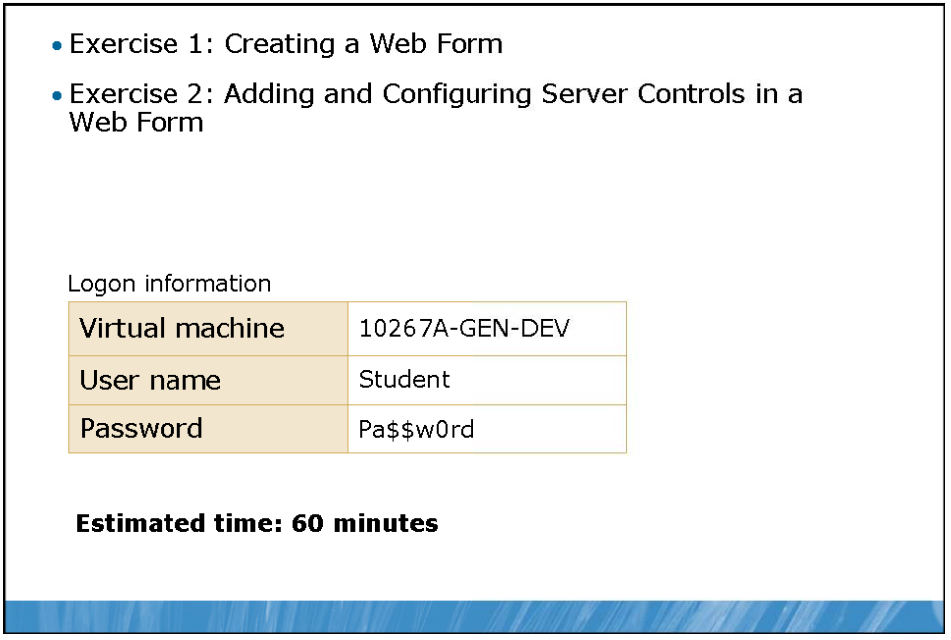
Results: After completing this exercise, you will have designed the initial version of the **InsertCustomer** Web Form for your Web site.

Module 3

Lab Instructions: Creating a Microsoft® ASP.NET Web Form (Visual C#)

Contents:

Lab: Creating a Microsoft ASP.NET Web Form



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in **Section 1** of the lab page. If you are using Visual C# as your programming language, refer to the steps provided in **Section 2** of the lab page.

Introduction

In this lab, you will add a Web Form to an ASP.NET Web application, then add server controls to the Web Form, and then configure its properties.

Objectives

After completing this lab, you will be able to:

- Create a new Web Form.
- Apply a style sheet to the Web Form.
- Create a table-style layout in the Web Form.

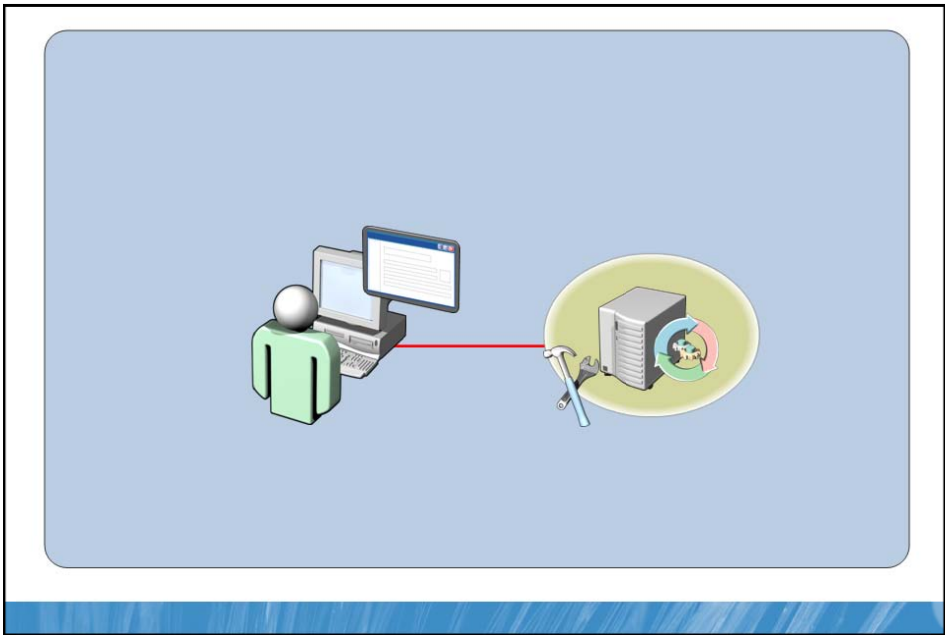
- Add a server control to the Web Form and configure its properties.
- Apply styles to the HTML elements and server controls.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
    - User name: **Student**
    - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage its customer information. Your organization has created a Web site to manage customer data and services in ASP.NET.

Contoso, Ltd wants to create an application to maintain and update its customer information. You need to customize the application to meet the specific requirements of the sales team. Updating the customer information is an ongoing process for the sales team, and the application requires updates based on the feedback from senior developers and other stakeholders. You need to build a UI that meets the defined requirements.

You must create an application and build a user interface that is easy to update and modify, by using ASP.NET server controls.

Section 2: Visual C#

Exercise 1: Creating a Web Form

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Add a Web Form to the Web site.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M3\CS folder.

▶ Task 2: Add a Web Form to the Web site

- Add the **InsertCustomer.aspx** Web Form to the **CustomerManagement** Web site by using the **Add New Item** dialog box.

Results: After completing this exercise, you will have opened the existing **CustomerManagement** Web site, and added the **InsertCustomer** Web Form.

Exercise 2: Adding and Configuring Server Controls in a Web Form

The main tasks for this exercise are as follows:

1. Add an existing style sheet to the Web Form.
2. Create a table-style layout in the Web Form.
3. Add the server controls to the Web Form and configure their basic properties.
4. Set the non-trivial control properties.
5. Apply a predefined style to the Web Form.

Note: In this exercise, you are creating a two-column table using CSS for holding various controls used for displaying and managing Customer data.

▶ Task 1: Add an existing style sheet to the Web Form

- Reference the **Styles/Site.css** file in the InsertCustomer Web Form, relative to the root folder, by using the Manage Styles window.
- Press CTRL+S to save InsertCustomer.aspx.
- View the styles added in the style sheet by using the Manage Styles window.

▶ Task 2: Create a table-style layout in the Web Form

- In the InsertCustomer.aspx window, place the cursor between the opening and closing **div** tags.
- Add a new **div** element from the Toolbox to the existing **div** element.

```
<div>
    <div>
    </div>
</div>
```

- Add two new **div** elements from the Toolbox to the newly added **div** element.

```
<div>
    <div>
        <div>
        </div>
        <div>
        </div>
    </div>
</div>
```

- Save the changes to the **InsertCustomer** Web Form.
- Use the Apply Styles window to apply the **customerTable** style to the outermost **div** element. Apply **div.customerTable** from the **Contextual Selectors** section of the Apply Styles window.
- Click the opening tag of the **div** element, which is a child element of the **div** element with a **class** attribute value of **customerTable**.
- Create new style in the Apply Styles window, by using the **New Style** button.
- Name the new style **div.customerTableRow** by using the **Selector** box of the **New Style** dialog box.
- Apply the style to the current document selection by using the **Apply new style to document selection** check box.
- Define the new style in the existing **Styles/Site.css** CSS file.
- Set the layout of the new style to **table-row**, by using the display list of the **Layout** category, and then close the **New Style** dialog box.
- Use the Apply Styles window to apply the **customerTableLeftCol** style to the first child **div** element of the **div** element with a **class** attribute value of **customerTableRow**.
- Use the Apply Styles window to apply the **customerTableRightCol** style to the second child **div** element of the **div** element with a **class** attribute value of **customerTableRow**.

- Add 11 more table rows by copying the existing **div** element with a **class** attribute value of **customerTableRow**.

```
<div class="customerTableRow">
    <div class="customerTableLeftCol">
    </div>
    <div class="customerTableRightCol">
    </div>
</div>
```

- Append a new **div** element from the Toolbox to the **div** element with a **class** attribute value of **customerTable**, placing the new **div** element immediately before the closing div tag of the **customerTable** div element.

```
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
        </div>
        <div class="customerTableRightCol">
        </div>
    </div>
    ...
    <div>
    </div>
</div>
```

- View the **InsertCustomer** Web Form in Design view.

Note: Notice that two equal-sized columns are added to the **div** element.

- View the **InsertCustomer** Web Form in Source view.
- Use the Apply Styles window to apply the **customerTableFooter** style to the last child **div** element of the **div** element with **class** attribute value of **customerTable**.
- Save the changes to the **InsertCustomer** Web Form and the **Site.css** CSS style sheet file.
- View the **InsertCustomer** Web Form in a Web browser.

Note: Although you have styled the Web form by using the CSS file, notice that nothing displays. This is because the div elements are empty.

▶ Task 3: Add the server controls to the Web Form and configure their basic properties

- View the **InsertCustomer** Web Form in Design view.
- Add the **Label** control from the Toolbox to the first cell of the left column in the **div** element, change the **(ID)** property to **CustomerFirstNameLabel**, and then set the **Text** property to **First Name:**.
- Add the **TextBox** control from the Toolbox to the first cell of the right column in the **div** element, and change the **(ID)** property to **CustomerFirstNameTextBox**.
- Add the **Label** control from the Toolbox to the second cell of the left column in the **div** element, change the **(ID)** property to **CustomerLastNameLabel**, and then set the **Text** property to **Last Name:**.
- Add the **TextBox** control from the Toolbox to the second cell of the right column in the **div** element, and change the **(ID)** property to **CustomerLastNameTextBox**.
- Add the **Label** control from the Toolbox to the third cell of the left column in the **div** element, change the **(ID)** property to **CustomerAddressLabel**, and then set the **Text** property to **Address:**.
- Add the **TextBox** control from the Toolbox to the third cell of the right column in the **div** element, and change the **(ID)** property to **CustomerAddressTextBox**.
- Add the **Label** control from the Toolbox to the fourth cell of the left column in the **div** element, change the **(ID)** property to **CustomerZipCodeLabel**, and then set the **Text** property to **Zip Code:**.
- Add the **TextBox** control from the Toolbox to the fourth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerZipCodeTextBox**.
- Add the **Label** control from the Toolbox to the fifth cell of the left column in the **div** element, change the **(ID)** property to **CustomerCityLabel**, and then set the **Text** property to **City:**.
- Add the **TextBox** control from the Toolbox to the fifth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCityTextBox**.
- Add the **Label** control from the Toolbox to the sixth cell of the left column in the **div** element, change the **(ID)** property to **CustomerStateLabel**, and then set the **Text** property to **State:**.

- Add the **TextBox** control from the Toolbox to the sixth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerStateTextBox**.
- Add the **Label** control from the Toolbox to the seventh cell of the left column in the **div** element, change the **(ID)** property to **CustomerCountryLabel**, and then set the **Text** property to **Country:**.
- Add the **DropDownList** control from the Toolbox to the seventh cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCountryDropDownList**.
- Add the **Label** control from the Toolbox to the eighth cell of the left column in the **div** element, change the **(ID)** property to **CustomerPhoneLabel**, and then set the **Text** property to **Phone:**.
- Add the **TextBox** control from the Toolbox to the eighth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerPhoneTextBox**.
- Add the **Label** control from the Toolbox to the ninth cell of the left column in the **div** element, change the **(ID)** property to **CustomerEmailAddressLabel**, and then set the **Text** property to **Email Address:**.
- Add the **TextBox** control from the Toolbox to the ninth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerEmailAddressTextBox**.
- Add the **Label** control from the Toolbox to the tenth cell of the left column in the **div** element, change the **(ID)** property to **CustomerWebAddressLabel**, and then set the **Text** property to **Web Address:**.
- Add the **TextBox** control from the Toolbox to the tenth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerWebAddressTextBox**.
- Add the **Label** control from the Toolbox to the eleventh cell of the left column in the **div** element, change the **(ID)** property to **CustomerCreditLimitLabel**, and then set the **Text** property to **Credit Limit:**.
- Add the **TextBox** control from the Toolbox to the eleventh cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCreditLimitTextBox**.

- Add the **Label** control from the Toolbox to the twelfth cell of the left column in the **div** element, change the **(ID)** property to **CustomerNewsSubscriberLabel**, and then set the **Text** property to **News Subscriber:**.
- Add the **CheckBox** control from the Toolbox to the twelfth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerNewsSubscriberCheckBox**.
- Save the changes, and view the **InsertCustomer** Web Form in a Web browser.
- Add the **Button** control from the Toolbox to the footer of the table in the **div** element, change the **(ID)** property to **CustomerInsertButton**, and then set the **Text** property to **Insert**.
- Add the **Button** control from the Toolbox to the footer of the table in the **div** element, change the **(ID)** property to **CustomerCancelButton**, and then set the **Text** property to **Cancel**.
- Save the changes, and view the **InsertCustomer** Web Form in a Web browser.

▶ Task 4: Set the non-trivial control properties

- Set the **MaxLength** property of the **CustomerFirstNameTextBox**, **CustomerAddressTextBox**, and **CustomerEmailAddressTextBox** controls to **50**.
- Set the **MaxLength** property of the **CustomerLastNameTextBox**, **CustomerCityTextBox**, **CustomerStateTextBox**, and **CustomerPhoneTextBox** controls to **30**.
- Set the **MaxLength** property of the **CustomerZipCodeTextBox** and **CustomerCreditLimitTextBox** controls to **10**.
- Open the **InsertCustomer** Web Form in the Source view.
- Set the **MaxLength** property of the **CustomerWebAddressTextBox** control to **80**.
- Save the changes and view the **InsertCustomer** Web Form in a Web browser.

▶ Task 5: Apply a predefined style to the Web Form

- Apply the predefined template style to the **body** element of the **InsertCustomer** Web form.
- Save the changes, and view the **InsertCustomer** Web Form in a Web browser.

▶ Task 6: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the 10267A-GEN-DEV virtual machine.

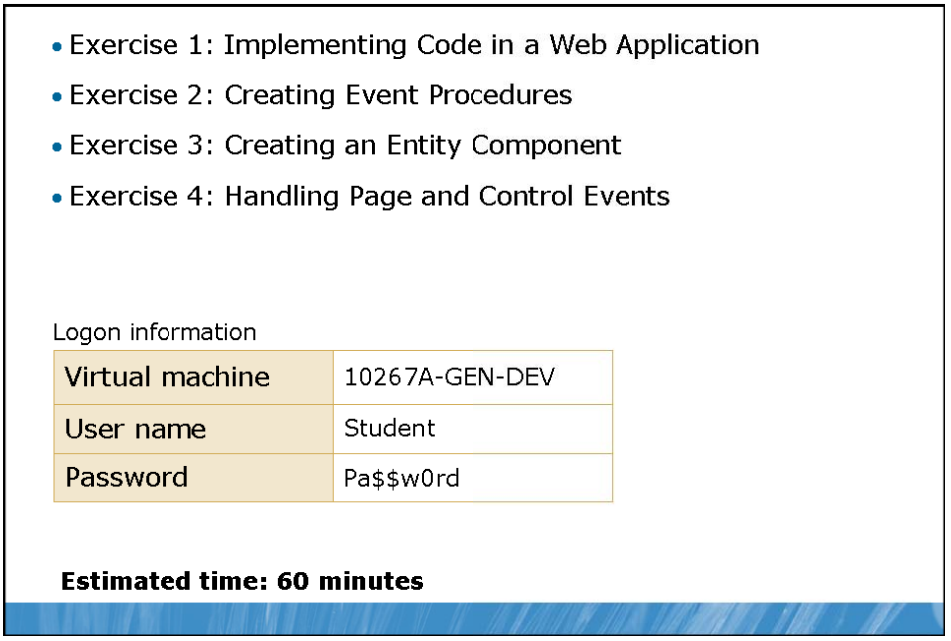
Results: After completing this exercise, you will have designed the initial version of the **InsertCustomer** Web Form for your Web site.

Module 4

Lab Instructions: Adding Functionality to a Microsoft® ASP.NET Web Form (Visual Basic)

Contents:

Lab: Adding Functionality to a Microsoft ASP.NET Web Form



Note: You can perform the tasks in this lab either by using the Visual Basic or the Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in **Section 1** of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in **Section 2** of the lab document.

Introduction

In this lab, you will implement the code and the event procedures on an ASP.NET Web site, and then create a component, which you will reference from the Web application. In addition, you will implement page and server control events on the Web site.

Objectives

After completing this lab, you will be able to:

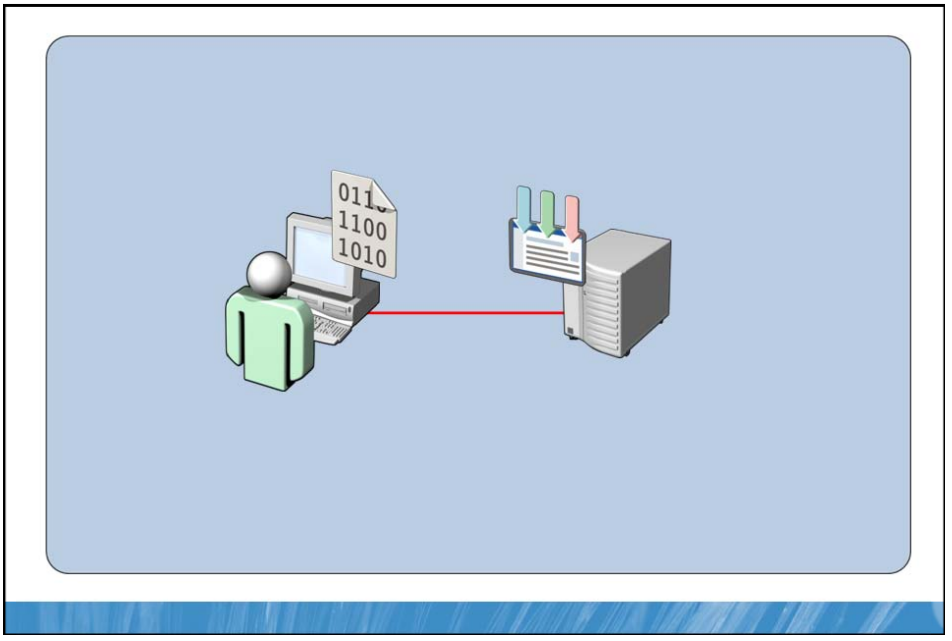
- Implement code in a Web application.
- Create event procedures.
- Create an entity component, and then reference it from a Web application.
- Handle page and server control events.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage its customer information.

Your organization is using a separate Web site for fast and easy interaction with its customers. The organization wants to make its Web site dynamic to enable users to enter and save customer details with minimum turnaround time.

To do this, you need to attach the required code to the UI that enables the application to respond to user actions and other events. In addition, you need to ensure that the application achieves a specified performance level by adding the code in a code-behind class file that is precompiled, thus saving considerable processing.

Section 1: Visual Basic

Exercise 1: Implementing Code in a Web Application

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Open the code-behind file for an existing Web Form.

▶ Task 1: Open an existing Web site

- Log on to **10267A-GEN-DEV** virtual machine as **Student**, with the password **Pa$$w0rd**.
- Open Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M4\VB folder.

▶ Task 2: Open the code-behind file for an existing Web Form

- Open the code-behind file of the **InsertCustomer** Web Form, by using the **View Code** context menu command.

Results: After completing this exercise, you will have opened the existing **CustomerManagement** Web site, and the InsertCustomer Web Form code-behind file.

Exercise 2: Creating Event Procedures

The main tasks for this exercise are as follows:

1. Create an event procedure for the **Click** event of the **Insert** button.
2. Create an event procedure for the **Click** event of the **Cancel** button.
3. Create an event procedure for the **Page_Load** event.
4. Create an event procedure for the **Page_LoadComplete** event.
5. Create an event procedure for the **Page_Unload** event.

▶ Task 1: Create an event procedure for the Click event of the Insert button

- Open the **InsertCustomer** Web Form in the Design view, and create an event procedure for the **Click** event of the **Insert** button, by double-clicking the **Button** control.

Note: Notice that the initial event procedure for the **Click** event of the **CustomerInsertButton** control is added in the code window.

▶ Task 2: Create an event procedure for the Click event of the Cancel button

- Open the **InsertCustomer** Web Form in the Design view, and create an event procedure for the **Click** event of the **Cancel** button, by double-clicking the box next to the **Click** event in the Properties window, with the **Button** control selected in the **Designer**.

Note: Notice that the initial event procedure for the **Click** event of the **CustomerCancelButton** is added to the class in the code window.

▶ Task 3: Create an event procedure for the Page_Load event

- Create an event procedure for the **Page_Load** event.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load

End Sub
```

▶ Task 4: Create an event procedure for the Page_LoadComplete event

- Create an event procedure for the **Page_LoadComplete** event.

```
Protected Sub Page_LoadComplete(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.LoadComplete

End Sub
```

▶ Task 5: Create an event procedure for the Page_Unload event

- Create an event procedure for the **Page_Unload** event.

```
Protected Sub Page_Unload(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Unload

End Sub
```

Results: After completing this exercise, you will have created event procedures for button controls, **Page_Load** event, **Page_LoadComplete** event, and **Page_Unload** event.

Exercise 3: Creating an Entity Component

The main tasks for this exercise are as follows:

1. Create an entity component.
2. Add the class member fields
3. Add the properties.
4. Add the constructors.
5. Reference CustomerManagementEntities project from CustomerManagement project.
6. Add a customer member declaration.

▶ Task 1: Create an entity component

- Create the **CustomerManagementEntities** class library project by using the **Class Library** project item in the **Add New Project** dialog box. Place the new project in the D:\Labfiles\Starter\M4\VB folder.

Note: Notice that the **CustomerManagementEntities** class library project is added to the solution.

- Rename the default class file **Class1.vb**, to **Customer.vb**.

▶ Task 2: Add the class member fields

- In the **Customer** class, add a region named **Class member fields**, just below the class declaration.

```
#Region "Class member fields"
#End Region
```

- In the **Customer** class, within the **Class member fields** region, add a private member field for the first name of the customer named **customerFirstName**, of type **String**, and initialize to **Nothing**.

```
Private customerFirstName As String = Nothing
```

- In the **Customer** class, within the **Class member fields** region, add a private member field for the last name of the customer named **customerLastName**, of type **String**, and initialize to **Nothing**.

```
Private customerLastName As String = Nothing
```

- In the **Customer** class, within the **Class member fields** region, add a private member field for the address of the customer named **customerAddress**, of type **String**, and initialize to **Nothing**.

```
Private customerAddress As String = Nothing
```

- Append the remaining backing fields by using a code snippet named **Customer class backing fields**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the declaration of the **customerAddress** backing field, and insert the snippet by clicking **Insert Snippet**.

```
Private customerZipCode As String = Nothing
Private customerCity As String = Nothing
Private customerState As String = Nothing
Private customerCountryID As Guid? = Nothing
Private customerPhone As String = Nothing
Private customerEmailAddress As String = Nothing
Private customerWebAddress As String = Nothing
Private customerCreditLimit As Integer = 0
Private customerNewsSubscriber As Boolean = False
Private customerCreatedDate As DateTime? = Nothing
Private customerCreatedBy As String = Nothing
Private customerModifiedDate As DateTime? = Nothing
Private customerModifiedBy As String = Nothing
```

▶ Task 3: Add the properties

- In the **Customer** class, add a region named **Properties**, below the **Class member fields** region.

```
#Region "Properties"
#End Region
```

- In the **Customer** class, within the **Properties** region, append an auto-implemented public property named **ID**, of nullable type **Guid**.

```
''' <summary>
''' The unique customer ID
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property ID() As Guid?
```

- In the **Customer** class, within the **Properties** region, add a public property named **FirstName**, of type **String**, that sets and gets the private member backing field named **customerFirstName**, and ensure that the length is no longer than 50 characters by using the following lines of code.

```
''' <summary>
''' The customer first name
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property FirstName As String
    Get
        Return Me.customerFirstName
    End Get

    Set(ByVal value As String)
        ' Null value?
        If Value Is Nothing Then
            Me.customerFirstName = String.Empty
        Else
            ' Only get the first 50 characters
            If (Value.Length > 50) Then
                Me.customerFirstName = value.Substring(0, 50)
            Else
                Me.customerFirstName = value
            End If
        End If
    End Set
End Property
```

- In the **Customer** class, within the **Properties** region, add a public property named **LastName**, of type **String**, that sets and gets the private member backing field named **customerLastName**, and ensure that the length is no longer than 30 characters by using the following lines of code.

```
''' <summary>
''' The customer last name
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property LastName As String
    Get
        Return Me.customerLastName
    End Get

    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerLastName = String.Empty
        Else
            ' Only get the first 30 characters
            If (value.Length > 30) Then
                Me.customerLastName = value.Substring(0, 30)
            Else
                Me.customerLastName = value
            End If
        End If
    End Set
End Property
```

- In the **Customer** class, within the **Properties** region, add a public property named **Address**, of type **String**, that sets and gets the private member backing field named **customerAddress**, and ensure that the length is no longer than 50 characters by using the following lines of code.

```
''' <summary>
''' The customer address, including street name, house number and
floor
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property Address As String
    Get
        Return Me.customerAddress
    End Get

    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerAddress = String.Empty
        Else
            ' Only get the first 50 characters
            If (value.Length > 50) Then
                Me.customerAddress = value.Substring(0, 50)
            Else
                Me.customerAddress = value
            End If
        End If
    End Set
End Property
```

- Append the remaining properties within the **Properties** region, by using a code snippet named **Customer class properties**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the declaration of the **Address** property, and insert the snippet by clicking **Insert Snippet**.

```
''' <summary>
''' The customer zip code or postal code
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property ZipCode() As String
    Get
        Return Me.customerZipCode
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerZipCode = ""
        Else
            ' Only get the first 10 characters
            If (value.Length > 10) Then
                Me.customerZipCode = value.Substring(0, 10)
            Else
                Me.customerZipCode = value
            End If
        End If
     End Set
End Property

''' <summary>
''' The name of the city in which the customer lives
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
```

(*Code continued on the following page.*)

```
Public Property City() As String
    Get
        Return Me.customerCity
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerCity = ""
        Else
            ' Only get the first 30 characters
            If (value.Length > 30) Then
                Me.customerCity = value.Substring(0, 30)
            Else
                Me.customerCity = value
            End If
        End If
    End Set
End Property

''' <summary>
''' The name of the state or region in which the customer lives
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property State() As String
    Get
        Return Me.customerState
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerState = ""
        Else
            ' Only get the first 30 characters
            If (value.Length > 30) Then
                Me.customerState = value.Substring(0, 30)
            Else
                Me.customerState = value
            End If
        End If
    End Set
End Property
```

(*Code continued on the following page.*)

```
''' <summary>
''' The ID of the country in which the customer lives
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property CountryID() As Guid?
    Get
        Return Me.customerCountryID
    End Get
    Set(ByVal value As Guid?)
        Me.customerCountryID = value
    End Set
End Property

''' <summary>
''' The customer phone number
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property Phone() As String
    Get
        Return Me.customerPhone
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerPhone = ""
        Else
            ' Only get the first 30 characters
            If (value.Length > 30) Then
                Me.customerPhone = value.Substring(0, 30)
            Else
                Me.customerPhone = value
            End If
        End If
    End Set
End Property

''' <summary>
''' The customer e-mail address
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
```

(*Code continued on the following page.*)

```
Public Property EmailAddress() As String
    Get
        Return Me.customerEmailAddress
    End Get
    Set(ByVal value As String)
        If (value Is Nothing) Then
            Me.customerEmailAddress = ""
        Else
            ' Only get the first 50 characters
            If (value.Length > 50) Then
                Me.customerEmailAddress = value.Substring(0, 50)
            Else
                Me.customerEmailAddress = value
            End If
        End If
    End Set
End Property

''' <summary>
''' The customer web address
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property WebAddress() As String
    Get
        Return Me.customerWebAddress
    End Get
    Set(ByVal value As String)
        If (value Is Nothing) Then
            Me.customerWebAddress = ""
        Else
           ' Only get the first 80 characters
           If (value.Length > 80) Then
               Me.customerWebAddress = value.Substring(0, 80)
           Else
               Me.customerWebAddress = value
            End If
        End If
    End Set
End Property
```

(*Code continued on the following page.*)

```
''' <summary>
''' The current credit limit of the customer
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property CreditLimit() As Integer
    Get
        Return Me.customerCreditLimit
    End Get
    Set(ByVal value As Integer)
        ' Negative value?
        If value < 0 Then
            Me.customerCreditLimit = 0
        Else
            Me.customerCreditLimit = value
        End If
    End Set
End Property

''' <summary>
''' Does the customer subscriber to news?
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property NewsSubsriber() As Boolean
    Get
        Return Me.customerNewsSubscriber
    End Get
    Set(ByVal value As Boolean)
        Me.customerNewsSubscriber = value
    End Set
End Property

''' <summary>
''' The date the customer was created in the system
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
```

(*Code continued on the following page.*)

```
Public Property CreatedDate() As DateTime?
    Get
        Return Me.customerCreatedDate
    End Get
    Private Set(ByVal value As DateTime?)
        ' Date in the past?
        If (value < DateTime.Now) Then
            Me.customerCreatedDate = DateTime.Now
        Else
            Me.customerCreatedDate = value
        End If
    End Set
End Property

''' <summary>
''' The name of the user creating the customer
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property CreatedBy() As String
    Get
        Return Me.customerCreatedBy
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerCreatedBy = ""
        Else
            ' Only get the first 15 characters
            If (value.Length > 15) Then
                Me.customerCreatedBy = value.Substring(0, 15)
            Else
                Me.customerCreatedBy = value
            End If
        End If
    End Set
End Property

''' <summary>
''' The date the customer was last modified in the system
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
```

(*Code continued on the following page.*)

```
Public Property ModifiedDate() As DateTime?
    Get
        Return Me.customerModifiedDate
    End Get
    Set(ByVal value As DateTime?)
        ' Date in the past?
        If value < DateTime.Now Then
            Me.customerModifiedDate = DateTime.Now
        Else
            Me.customerModifiedDate = value
        End If
    End Set
End Property

''' <summary>
''' The name of the user who last modified the customer
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property ModifiedBy() As String
    Get
        Return Me.customerModifiedBy
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerModifiedBy = ""
        Else
            ' Only get the first 15 characters
            If (value.Length > 15) Then
                Me.customerModifiedBy = value.Substring(0, 15)
            Else
                Me.customerModifiedBy = value
            End If
        End If
    End Set
End Property
```

▶ Task 4: Add the constructors

- In the **Customer** class, add a region named **Constructors** below the **Properties** region.

```
#Region "Constructors"
#End Region
```

- In the **Customer** class, within the **Constructors** region, add the default public parameterless constructor that initializes the **customerID** and **customerCreatedDate** member fields, by using the public properties.

```
''' <summary>
''' Default parameterless constructor
''' </summary>
''' <remarks></remarks>
Public Sub New()
    ' Initialize backing fields with default values
    Me.ID = Guid.NewGuid()
    Me.CreatedDate = DateTime.Now
End Sub
```

- Append the remaining properties within the **Constructors** region by using a code snippet named **Customer class constructors**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the default parameterless constructor, and insert the snippet by clicking **Insert Snippet**.

```
''' <summary>
''' Initializes backing fields with passed and default values
''' </summary>
''' <param name="id"></param>
''' <remarks></remarks>
Public Sub New(ByVal id As Guid?)
    ' Initialize backing fields with passed and default values
    Me.ID = id
    Me.CreatedDate = DateTime.Now
End Sub

''' <summary>
''' Initializes with a value for all backing fields
''' </summary>
''' <param name="id"></param>
''' <param name="firstName"></param>
''' <param name="lastName"></param>
''' <param name="address"></param>
''' <param name="zipCode"></param>
''' <param name="city"></param>
''' <param name="state"></param>
''' <param name="countryID"></param>
''' <param name="phone"></param>
''' <param name="emailAddress"></param>
```

(*Code continued on the following page.*)

```
''' <param name="webAddress"></param>
''' <param name="creditLimit"></param>
''' <param name="newsSubscriber"></param>
''' <param name="createdDate"></param>
''' <param name="createdBy"></param>
''' <param name="modifiedDate"></param>
''' <param name="modifiedBy"></param>
''' <remarks></remarks>
Public Sub New(ByVal id As Guid?, ByVal firstName As String, ByVal
lastName As String,
    ByVal address As String, ByVal zipCode As String, ByVal city
As String, ByVal state As String,
    ByVal countryID As Guid?, ByVal phone As String, ByVal
emailAddress As String,
    ByVal webAddress As String, ByVal creditLimit As Integer,
ByVal newsSubscriber As Boolean,
    ByVal createdDate As DateTime?, ByVal createdBy As String,
ByVal modifiedDate As DateTime?,
    ByVal modifiedBy As String)

    ' Initialize member backing fields with passed values
    Me.ID = id
    Me.FirstName = firstName
    Me.LastName = lastName
    Me.Address = address
    Me.ZipCode = zipCode
    Me.City = city
    Me.State = state
    Me.CountryID = countryID
    Me.Phone = phone
    Me.EmailAddress = emailAddress
    Me.WebAddress = webAddress
    Me.CreditLimit = creditLimit
    Me.NewsSubscriber = newsSubscriber

    If Not createdDate Is Nothing Then
        Me.CreatedDate = createdDate
    Else
        Me.CreatedDate = DateTime.Now
    End If

    Me.CreatedBy = createdBy
    Me.ModifiedDate = modifiedDate
    Me.ModifiedBy = modifiedBy
End Sub
```

- Save the changes to **Customer.vb**.
- Build the component, and fix any errors.

▶ Task 5: Reference CustomerManagementEntities project from CustomerManagement project

- Add a reference to the **CustomerManagement** project by using the **Add Reference** dialog box. Reference the **CustomerManagementEntities** project from the **CustomerManagement** project.

▶ Task 6: Add a customer member declaration

- Open the **InsertCustomer.aspx.vb** file.
- In the InsertCustomer.aspx.vb code window, add a private class member declaration of the **Customer** class in the **CustomerManagementEntities** namespace, named **currentCustomer**, and initialize to **Nothing**.

```
Private currentCustomer As CustomerManagementEntities.Customer =
Nothing
```

Results: After completing this exercise, you will have created a new component by using the Class Library project, added a reference to the component project from the Web site, and added a member variable of type **Customer**.

Exercise 4: Handling Page and Control Events

The main tasks for this exercise are as follows:

1. Instantiate the Customer object.
2. Populate the UI controls.
3. Destroy the objects.
4. Handle the user cancellation.
5. Save the customer information.

▶ Task 1: Instantiate the Customer object

- Instantiate the Customer object by using the following code.

```
''' <summary>
''' Instantiates Customer object
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Instantiate Customer
    instantiateCustomerObject()
End Sub
```

- Appending the following code to the **InsertCustomer** class.

```
''' <summary>
''' Instantiates and populates the Customer member object
''' </summary>
''' <remarks></remarks>
Private Sub instantiateCustomerObject()
    ' First time loading page?
    If Not Me.IsPostBack Then
        ' Instantiate new Customer object
        currentCustomer = New
CustomerManagementEntities.Customer()
    Else
        ' Instantiate new Customer object with user input
        currentCustomer = New CustomerManagementEntities.Customer(
            Nothing, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
            CustomerAddressTextBox.Text,
CustomerZipCodeTextBox.Text, CustomerCityTextBox.Text,
            CustomerStateTextBox.Text, Nothing,
CustomerPhoneTextBox.Text,
            CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text, -1,
            CustomerNewsSubscriberCheckBox.Checked, DateTime.Now,
_
            "", Nothing, "")
    End If
End Sub
```

▶ Task 2: Populate the UI controls

- In the **InsertCustomer** class, populate the server controls in the UI by using the values from the private **Customer** object **currentCustomer**.

```
''' <summary>
''' Populates UI controls
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_LoadComplete(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.LoadComplete
    ' Populate the UI controls
    populateUI()
End Sub

''' <summary>
''' Populates the UI controls with the values in the
''' current Customer object
''' </summary>
''' <remarks></remarks>
Private Sub populateUI()
    CustomerFirstNameTextBox.Text = currentCustomer.FirstName
    CustomerLastNameTextBox.Text = currentCustomer.LastName
    CustomerAddressTextBox.Text = currentCustomer.Address
    CustomerZipCodeTextBox.Text = currentCustomer.ZipCode
    CustomerCityTextBox.Text = currentCustomer.City
    CustomerStateTextBox.Text = currentCustomer.State

    If currentCustomer.CountryID.HasValue Then
        CustomerCountryDropDownList.SelectedValue =
currentCustomer.CountryID.Value.ToString()
    Else
        CustomerCountryDropDownList.SelectedIndex = -1
    End If

    CustomerPhoneTextBox.Text = currentCustomer.Phone
    CustomerEmailAddressTextBox.Text =
currentCustomer.EmailAddress
    CustomerWebAddressTextBox.Text = currentCustomer.WebAddress
    CustomerCreditLimitTextBox.Text =
currentCustomer.CreditLimit.ToString()
    CustomerNewsSubscriberCheckBox.Checked =
currentCustomer.NewsSubscriber
End Sub
```

▶ Task 3: Destroy the objects

- In the **InsertCustomer** class, destroy the objects used in the class that are not automatically handled by the garbage collector.

```
''' <summary>
''' Destroys objects
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_Unload(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Unload
    ' Destroy Customer object
    currentCustomer = Nothing
End Sub
```

▶ Task 4: Handle the user cancellation

- In the **InsertCustomer** class, handle user cancellation by redirecting to the home page.

```
''' <summary>
''' Redirects to home page
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerCancelButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CustomerCancelButton.Click
    ' Redirect to home page
    Response.Redirect("~/Default.aspx")
End Sub
```

▶ Task 5: Save the customer information

- In the **InsertCustomer** class, prepare for saving the customer input information to persistent storage when the user clicks the **Insert** button.

```
''' <summary>
''' Saves the current customer information and adds default values
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub customerInsertButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles customerInsertButton.Click
    ' Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name
    ' Add the user credit limit
    currentCustomer.CreditLimit = 50000
End Sub
```

- Save the changes to **InsertCustomer.aspx.vb**.
- Build the solution and fix any errors.
- Close Visual Studio 2010.

Note: Notice that the validation successfully completes.

Note: The initial code for saving the customer information is created here. However, the final code for saving to the database will be created in Module 8.

▶ Task 6: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the 10267A-GEN-DEV virtual machine.

Results: After completing this exercise, you will have added code to handle the **Page.Load**, **Page.LoadComplete**, and **Page.Unload** events for the InsertCustomer Web Form, and added code to handle the **Click** event for the **Insert** and **Cancel** button controls.

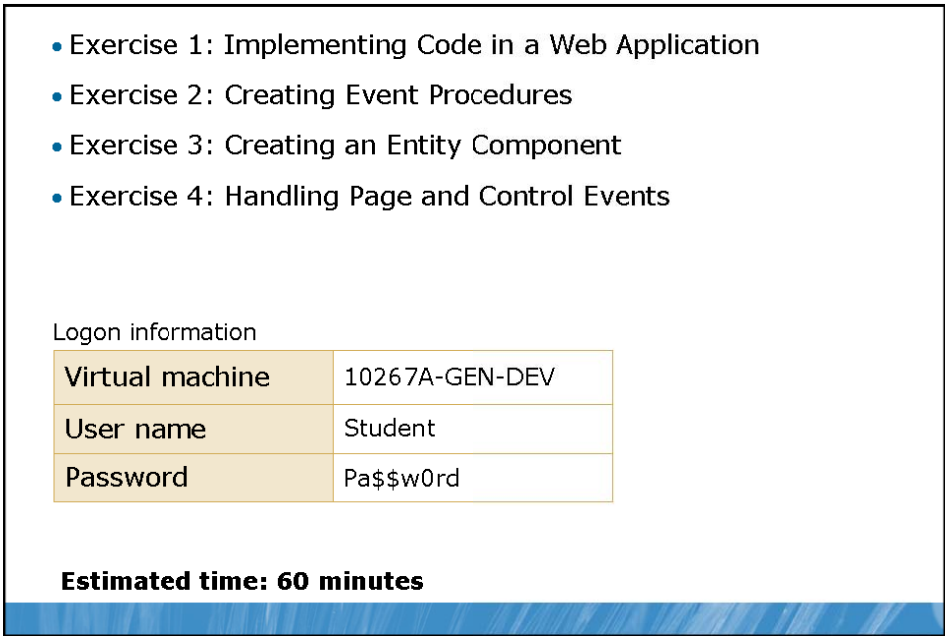
Note: The answers to the exercises are on the Course Companion CD.

Module 4

Lab Instructions: Adding Functionality to a Microsoft® ASP.NET Web Form (Visual C#)

Contents:

Lab: Adding Functionality to a Microsoft ASP.NET Web Form



Note: You can perform the tasks in this lab either by using the Visual Basic or the Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in **Section 1** of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in **Section 2** of the lab document.

Introduction

In this lab, you will implement the code and the event procedures on an ASP.NET Web site, and then create a component, which you will reference from the Web application. In addition, you will implement page and server control events on the Web site.

Objectives

After completing this lab, you will be able to:

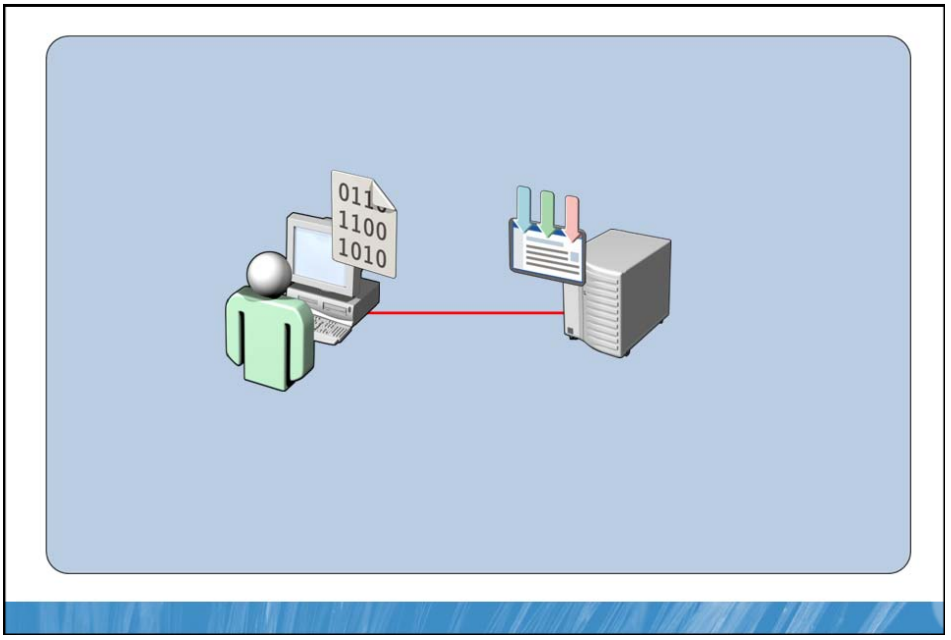
- Implement code in a Web application.
- Create event procedures.
- Create an entity component, and then reference it from a Web application.
- Handle page and server control events.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage its customer information.

Your organization is using a separate Web site for fast and easy interaction with its customers. The organization wants to make its Web site dynamic to enable users to enter and save customer details with minimum turnaround time.

To do this, you need to attach the required code to the UI that enables the application to respond to user actions and other events. In addition, you need to ensure that the application achieves a specified performance level by adding the code in a code-behind class file that is precompiled, thus saving considerable processing.

Section 2: Visual C#

Exercise 1: Implementing Code in a Web Application

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Open the code-behind file for an existing Web Form.

▶ Task 1: Open an existing Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M4\CS folder.

▶ Task 2: Open the code-behind file for an existing Web Form

- Open the code-behind file of the **InsertCustomer** Web Form, by using the **View Code** context menu command.

Results: After completing this exercise, you will have opened the existing **CustomerManagement** Web site and the InsertCustomer Web Form code-behind file.

Exercise 2: Creating Event Procedures

The main tasks for this exercise are as follows:

1. Create an event procedure for the **Click** event of the **Insert** button.
2. Create an event procedure for the **Click** event of the **Cancel** button.
3. Create an event procedure for the **Page_LoadComplete** event.
4. Create an event procedure for the **Page_Unload** event.

▶ Task 1: Create an event procedure for the Click event of the Insert button

- Open the **InsertCustomer** Web Form in the Design view, and create an event procedure for the **Click** event of the **Insert** button, by double-clicking the **Button** control.

Note: Notice that the initial event procedure for the **Click** event of the **CustomerInsertButton** control is added in the code window.

▶ Task 2: Create an event procedure for the Click event of the Cancel button

- Open the **InsertCustomer** Web Form in the Design view, and create an event procedure for the **Click** event of the **Cancel** button, by double-clicking the box next to the **Click** event in the Properties window, with the **Button** control selected in the **Designer**.

Note: Notice that the initial event procedure for the **Click** event of the **CustomerCancelButton** is added to the class in the code window.

▶ Task 3: Create an event procedure for the **Page_LoadComplete** event

```
protected void Page_LoadComplete(object sender, EventArgs e)
{

}
```

▶ Task 4: Create an event procedure for the **Page_Unload** event

```
protected void Page_Unload(object sender, EventArgs e)
{

}
```

Results: After completing this exercise, you will have created event procedures for button controls **Page_LoadComplete** event, and **Page_Unload** event.

Exercise 3: Creating an Entity Component

The main tasks for this exercise are as follows:

1. Create an entity component.
2. Add the class member fields.
3. Add the properties.
4. Add the constructors.
5. Reference CustomerManagementEntities project from CustomerManagement project.
6. Add a customer member declaration.

▶ Task 1: Create an entity component

- Create the **CustomerManagementEntities** class library project by using the **Class Library project** item in the **Add New Project** dialog box. Place the new project in the D:\Labfiles\Starter\M4\CS folder.

Note: Notice that the **CustomerManagementEntities** class library project is added to the solution.

- Rename the default class, **Class1.cs**, file as **Customer.cs**.

▶ Task 2: Add the class member fields

- In the **Customer** class, add a region named **Class member fields**, just below the class declaration.

```
#region Class member fields
#endregion
```

- In the **Customer** class, within the **Class member fields** region, add a private member field for the first name of the customer named **customerFirstName**, of type **string**, and initialize to **null**.

```
private string customerFirstName = null;
```

- In the **Customer** class, within the **Class member fields** region, add a private member field for the last name of the customer named **customerLastName**, of type **string**, and initialize to **null**.

```
private string customerLastName = null;
```

- In the **Customer** class, within the **Class member fields** region, add a private member field for the address of the customer named **customerAddress**, of type **string**, and initialize to **null**.

```
private string customerAddress = null;
```

- Append the remaining backing fields by using a code snippet named **Customer class backing fields**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the declaration of the **customerAddress** backing field, and insert the snippet by clicking **Insert Snippet**.

```
private string customerZipCode = null;
private string customerCity = null;
private string customerState = null;
private Guid? customerCountryID = null;
private string customerPhone = null;
private string customerEmailAddress = null;
private string customerWebAddress = null;
private int customerCreditLimit = 0;
private bool customerNewsSubscriber = false;
private DateTime? customerCreatedDate = null;
private string customerCreatedBy = null;
private DateTime? customerModifiedDate = null;
private string customerModifiedBy = null;
```

▶ Task 3: Add the properties

- In the **Customer** class, add a region named **Properties**, below the **Class member fields** region.

```
#region Properties
#endregion
```

- In the **Customer** class, within the **Properties** region, append an auto-implemented public property named **ID**, of nullable type **Guid**.

```
/// <summary>
/// The unique customer ID
/// </summary>
public Guid? ID { get; set; }
```

- In the **Customer** class, within the **Properties** region, add a public property named **FirstName**, of type **string**, that sets and gets the private member backing field named **customerFirstName**. Ensure that the length is no longer than 50 characters, by using the following lines of code.

```
/// <summary>
/// The customer first name
/// </summary>
public string FirstName
{
    get
    {
        return this.customerFirstName;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerFirstName = "";
        else
            // Only get the first 50 characters
            if (value.Length > 50)
                this.customerFirstName = value.Substring(0, 50);
            else
                this.customerFirstName = value;
    }
}
```

- In the **Customer** class, within the **Properties** region, add a public property named **LastName**, of type **string**, that sets and gets the private member backing field named **customerLastName**. Ensure that the length is no longer than 30 characters, by using the following lines of code.

```
/// <summary>
/// The customer last name
/// </summary>
public string LastName
{
    get
    {
        return this.customerLastName;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerLastName = "";
        else
            // Only get the first 30 characters
            if (value.Length > 30)
                this.customerLastName = value.Substring(0, 30);
            else
                this.customerLastName = value;
    }
}
```

- In the **Customer** class, within the **Properties** region, add a public property named **Address**, of type **string**, that sets and gets the private member backing field named **customerAddress**. Ensure that the length is no longer than 50 characters, by using the following lines of code.

```
/// <summary>
/// The customer address, including street name, house number and
floor
/// </summary>
public string Address
{
    get
    {
        return this.customerAddress;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerAddress = "";
        else
            // Only get the first 50 characters
            if (value.Length > 50)
                this.customerAddress = value.Substring(0, 50);
            else
                this.customerAddress = value;
    }
}
```

- Append the remaining properties within the **Properties** region, by using a code snippet named **Customer class properties**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the declaration of the **Address** property, and insert the snippet by clicking **Insert Snippet**.

```
/// <summary>
/// The customer zip code or postal code
/// </summary>
public string ZipCode
{
    get
    {
        return this.customerZipCode;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerZipCode = "";
        else
            // Only get the first 10 characters
            if (value.Length > 10)
                this.customerZipCode = value.Substring(0, 10);
            else
                this.customerZipCode = value;
    }
}

/// <summary>
/// The name of the city in which the customer lives
/// </summary>
public string City
{
    get
    {
        return this.customerCity;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerCity = "";
        else
```

(*Code continued on the following page.*)

```
            // Only get the first 30 characters
            if (value.Length > 30)
                this.customerCity = value.Substring(0, 30);
            else
                this.customerCity = value;
    }
}

/// <summary>
/// The name of the state or region in which the customer lives
/// </summary>
public string State
{
    get
    {
        return this.customerState;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerState = "";
        else
            // Only get the first 30 characters
            if (value.Length > 30)
                this.customerState = value.Substring(0, 30);
            else
                this.customerState = value;
    }
}

/// <summary>
/// The ID of the country in which the customer lives
/// </summary>
public Guid? CountryID
{
    get
    {
        return this.customerCountryID;
    }
    set
    {
        this.customerCountryID = value;
    }
}
```

(*Code continued on the following page.*)

```
/// <summary>
/// The customer phone number
/// </summary>
public string Phone
{
    get
    {
        return this.customerPhone;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerPhone = "";
        else
            // Only get the first 30 characters
            if (value.Length > 30)
                this.customerPhone = value.Substring(0, 30);
            else
                this.customerPhone = value;
    }
}

/// <summary>
/// The customer e-mail address
/// </summary>
public string EmailAddress
{
    get
    {
        return this.customerEmailAddress;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerEmailAddress = "";
        else
            // Only get the first 50 characters
            if (value.Length > 50)
                this.customerEmailAddress = value.Substring(0,
50);
            else
                this.customerEmailAddress = value;
    }
}
```

(*Code continued on the following page.*)

```
/// <summary>
/// The customer Web address
/// </summary>
public string WebAddress
{
    get
    {
        return this.customerWebAddress;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerWebAddress = "";
        else
            // Only get the first 80 characters
            if (value.Length > 80)
                this.customerWebAddress = value.Substring(0, 80);
            else
                this.customerWebAddress = value;
    }
}

/// <summary>
/// The current credit limit of the customer
/// </summary>
public int CreditLimit
{
    get
    {
        return this.customerCreditLimit;
    }
    set
    {
        // Negative value?
        if (value < 0)
            this.customerCreditLimit = 0;
        else
            this.customerCreditLimit = value;
    }
}
```

(*Code continued on the following page.*)

```
/// <summary>
/// Does the customer subscriber to news?
/// </summary>
public bool NewsSubscriber
{
    get
    {
        return this.customerNewsSubscriber;
    }
    set
    {
        this.customerNewsSubscriber = value;
    }
}

/// <summary>
/// The date the customer was created in the system
/// </summary>
public DateTime? CreatedDate
{
    get
    {
        return this.customerCreatedDate;
    }
    private set
    {
        // Date in the past?
        if (value < DateTime.Now)
            this.customerCreatedDate = DateTime.Now;
        else
            this.customerCreatedDate = value;
    }
}

/// <summary>
/// The name of the user creating the customer
/// </summary>
public string CreatedBy
{
    get
    {
        return this.customerCreatedBy;
    }
```

(*Code continued on the following page.*)

```
    set
    {
        // Null value?
        if (value == null)
            this.customerCreatedBy = "";
        else
            // Only get the first 15 characters
            if (value.Length > 15)
                this.customerCreatedBy = value.Substring(0, 15);
            else
                this.customerCreatedBy = value;
    }
}

/// <summary>
/// The date the customer was last modified in the system
/// </summary>
public DateTime? ModifiedDate
{
    get
    {
        return this.customerModifiedDate;
    }
    set
    {
        // Date in the past?
        if (value < DateTime.Now)
            this.customerModifiedDate = DateTime.Now;
        else
            this.customerModifiedDate = value;
    }
}

/// <summary>
/// The name of the user who last modified the customer
/// </summary>
public string ModifiedBy
{
    get
    {
        return this.customerModifiedBy;
    }
```

(*Code continued on the following page.*)

```
    set
    {
        // Null value?
        if (value == null)
            this.customerModifiedBy = "";
        else
            // Only get the first 15 characters
            if (value.Length > 15)
                this.customerModifiedBy = value.Substring(0, 15);
            else
                this.customerModifiedBy = value;
    }
}
```

▶ Task 4: Add the constructors

- In the **Customer** class, add a region named **Constructors**, below the **Properties** region.

```
#region Constructors
#endregion
```

- In the **Customer** class, within the **Constructors** region, add the default public parameterless constructor that initializes the **customerID** and **customerCreatedDate** member fields by using the public properties.

```
/// <summary>
/// Default parameterless constructor
/// </summary>
public Customer()
{
    // Initialize backing fields with default values
    this.ID = Guid.NewGuid();
    this.CreatedDate = DateTime.Now;
}
```

- Append the remaining properties within the **Constructors** region by using a code snippet named **Customer class constructors**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the default parameterless constructor, and insert the snippet by clicking **Insert Snippet**.

```
/// <summary>
/// Initializes backing fields with passed and default values
/// </summary>
/// <param name="id"></param>
public Customer(Guid? id)
{
    // Initialize backing fields with passed and default values
    this.ID = id;
    this.CreatedDate = DateTime.Now;
}

/// <summary>
/// Initializes with a value for all backing fields
/// </summary>
/// <param name="id"></param>
/// <param name="firstName"></param>
/// <param name="lastName"></param>
/// <param name="address"></param>
/// <param name="zipCode"></param>
/// <param name="city"></param>
/// <param name="state"></param>
/// <param name="countryID"></param>
/// <param name="phone"></param>
/// <param name="emailAddress"></param>
/// <param name="webAddress"></param>
/// <param name="creditLimit"></param>
/// <param name="newsSubscriber"></param>
/// <param name="createdDate"></param>
/// <param name="createdBy"></param>
/// <param name="modifiedDate"></param>
/// <param name="modifiedBy"></param>
public Customer(Guid? id, string firstName, string lastName,
string address,
    string zipCode, string city, string state, Guid? countryID,
string phone,
    string emailAddress, string webAddress, int creditLimit, bool
newsSubscriber,
    DateTime? createdDate, string createdBy, DateTime?
modifiedDate, string modifiedBy)
```

(*Code continued on the following page.*)

```
{
    // Initialize member backing fields with passed values
    this.ID = id;
    this.FirstName = firstName;
    this.LastName = lastName;
    this.Address = address;
    this.ZipCode = zipCode;
    this.City = city;
    this.State = state;
    this.CountryID = countryID;
    this.Phone = phone;
    this.EmailAddress = emailAddress;
    this.WebAddress = webAddress;
    this.CreditLimit = creditLimit;
    this.NewsSubscriber = newsSubscriber;

    if (createdDate != null)
        this.CreatedDate = createdDate;
    else
        this.CreatedDate = DateTime.Now;

    this.CreatedBy = createdBy;
    this.ModifiedDate = modifiedDate;
    this.ModifiedBy = modifiedBy;
}
```

- Save the changes to **Customer.cs**.
- Build the component, and fix any errors.

▶ Task 5: Reference CustomerManagementEntities project from CustomerManagement project

- Add a reference to the **CustomerManagement** project by using the **Add Reference** dialog box. Reference the **CustomerManagementEntities** project from the **CustomerManagement** project.

▶ Task 6: Add a customer member declaration

- Open the **InsertCustomer.aspx.cs** file.
- In the InsertCustomer.aspx.cs code window, add a private class member declaration of the **Customer** class in the **CustomerManagementEntities** namespace named **currentCustomer**, and initialize to **null**.

```
private CustomerManagementEntities.Customer currentCustomer =
null;
```

Results: After completing this exercise, you will have created a new component by using the Class Library project, added a reference to the component project from the Web site, and added a member variable of type Customer.

Exercise 4: Handling Page and Control Events

The main tasks for this exercise are as follows:

1. Instantiate the Customer object.
2. Populate the UI controls.
3. Destroy the objects.
4. Handle the user cancellation.
5. Save the customer information.

▶ Task 1: Instantiate the Customer object

- Instantiate the **Customer** object by using the following code.

```
/// <summary>
/// Instantiates Customer object
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_Load(object sender, EventArgs e)
{
    // Instantiate Customer
    instantiateCustomerObject();
}
```

- Append the following code to the **InsertCustomer** class.

```
/// <summary>
/// Instantiates and populates the Customer member object
/// </summary>
private void instantiateCustomerObject()
{
    // First time loading page?
    if (!this.IsPostBack)
        // Instantiate new Customer object
        currentCustomer = new CustomerManagementEntities.Customer(
            null, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
            CustomerAddressTextBox.Text,
CustomerZipCodeTextBox.Text,
            CustomerCityTextBox.Text, CustomerStateTextBox.Text,
            null, CustomerPhoneTextBox.Text,
CustomerEmailAddressTextBox.Text,
            CustomerWebAddressTextBox.Text, -1,
CustomerNewsSubscriberCheckBox.Checked,
            DateTime.Now, "", null, "");
}
```

▶ Task 2: Populate the UI controls

- In the **InsertCustomer** class, populate the server controls in the UI by using the values from the private **Customer** object, **currentCustomer**.

```
/// <summary>
/// Populates UI controls
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_LoadComplete(object sender, EventArgs e)
{
    // Populate the UI controls
    populateUI();
}
/// <summary>
/// Populates the UI controls with the values in the
/// current Customer object
/// </summary>
private void populateUI()
{
    CustomerFirstNameTextBox.Text = currentCustomer.FirstName;
    CustomerLastNameTextBox.Text = currentCustomer.LastName;
    CustomerAddressTextBox.Text = currentCustomer.Address;
    CustomerZipCodeTextBox.Text = currentCustomer.ZipCode;
    CustomerCityTextBox.Text = currentCustomer.City;
    CustomerStateTextBox.Text = currentCustomer.State;

    if (currentCustomer.CountryID.HasValue)
        CustomerCountryDropDownList.SelectedValue =
currentCustomer.CountryID.Value.ToString();
    else
        CustomerCountryDropDownList.SelectedIndex = -1;

    CustomerPhoneTextBox.Text = currentCustomer.Phone;
    CustomerEmailAddressTextBox.Text =
currentCustomer.EmailAddress;
    CustomerWebAddressTextBox.Text = currentCustomer.WebAddress;
    CustomerCreditLimitTextBox.Text =
currentCustomer.CreditLimit.ToString();
    CustomerNewsSubscriberCheckBox.Checked =
currentCustomer.NewsSubscriber;
}
```

▶ Task 3: Destroy the objects

- In the **InsertCustomer** class, destroy the objects used in the class that are not automatically handled by the garbage collector.

```
/// <summary>
/// Destroys objects
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_Unload(object sender, EventArgs e)
{
    // Destroy Customer object
    currentCustomer = null;
}
```

▶ Task 4: Handle the user cancellation

- In the **InsertCustomer** class, handle user cancellation by redirecting to the home page.

```
/// <summary>
/// Redirects to home page
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerCancelButton_Click(object sender, EventArgs
e)
{
    // Redirect to home page
    Response.Redirect("~/Default.aspx");
}
```

▶ Task 5: Save the customer information

- In the **InsertCustomer** class, prepare to save the customer input information to persistent storage when the user clicks the **Insert** button.

```
/// <summary>
/// Saves the current customer information and adds default values
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerInsertButton_Click(object sender, EventArgs
e)
{
    // Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name;
    // Add the user credit limit
    currentCustomer.CreditLimit = 50000;
}
```

- Save the changes to **InsertCustomer.aspx.cs**.
- Build the solution, and fix any errors.
- Close Visual Studio 2010.

Note: Notice that the validation successfully completed.

Note: The initial code for saving the customer information is created here. However, the final code for saving to the database is created in Module 8.

▶ Task 6: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the 10267A-GEN-DEV virtual machine.

Results: After completing this exercise, you will have added code to handle the **Page.Load**, **Page.LoadComplete**, and **Page.Unload** events for the InsertCustomer Web Form, and added code to handle the **Click** event for the **Insert** and **Cancel** button controls.

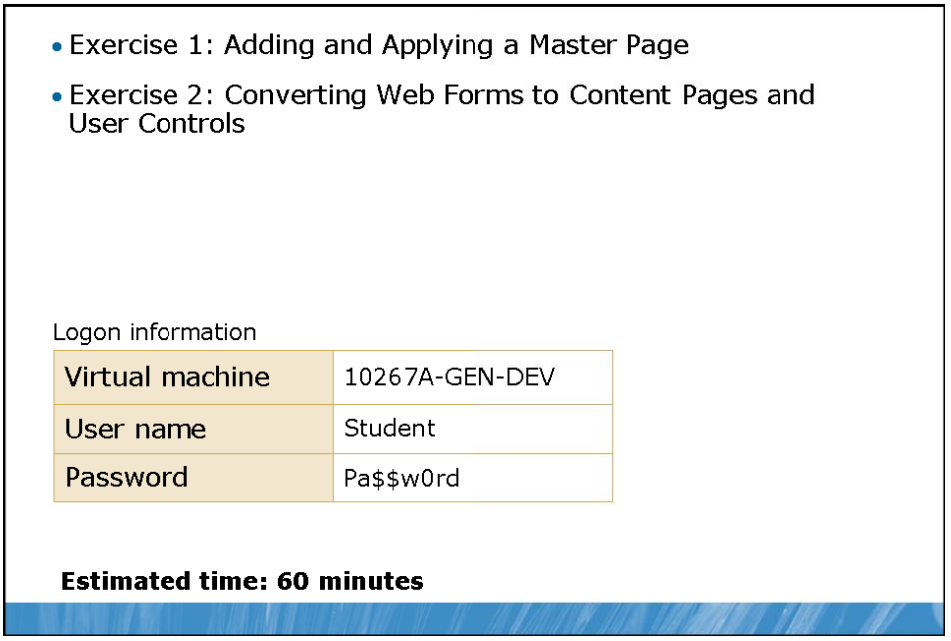
Note: The answers to the exercises are on the Course Companion CD.

Module 5

Lab Instructions: Implementing Master Pages and User Controls (Visual Basic)

Contents:

Lab: Implementing Master Pages and User Controls



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in **Section 1** of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in **Section 2** of the lab document.

Introduction

In this lab, you will implement master pages and user controls in an ASP.NET Web project. In addition, you will add navigation to the master page and convert a Web Form to a user control.

Objectives

After completing this lab, you will be able to:

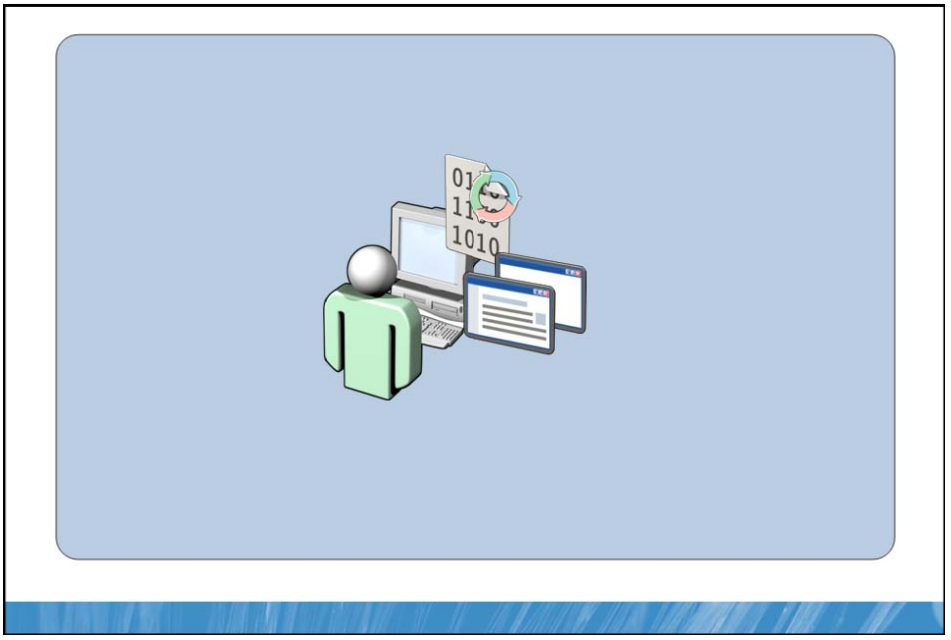
- Add and apply a master page in a Web application.
- Convert a Web Form to a content page.
- Add navigation to the master page.
- Convert a Web Form into a user control.
- Create a content page.
- Insert a user control on the content page.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. The organization has decided to modify their Web site to provide consistent user experience to all customers.

To do this, you need to add a master page and convert an existing Web Form into a user control for reuse. In addition, you will need to add navigation to the master page. A senior developer has already created some of the code to add navigation to the Web site, and the developer has created a sitemap XML document. You can use the code and the document for adding navigation to the master page. In addition, you need to convert a Web Form into a content page, move the content from the content page to the master page, and create a new content page with a user control.

Section 1: Visual Basic

Exercise 1: Adding and Applying a Master Page

The main tasks for this exercise are as follows:

1. Add a master page to an existing Web site.
2. Initialize the style controls and elements on the master page.
3. Define a **ContentPlaceHolder** control on the master page.

▶ Task 1: Add a master page to an existing Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the **D:\Labfiles\Starter\M5\VB** folder.
- Add a new master page named **Site.master**, to the **CustomerManagement** Web site.

▶ Task 2: Initialize the style controls and elements on the master page

- In the Site.master window, add an **id** property to the **head** element by using the following code.

```
<head runat="server" id="MainHead">
```

- In the Site.master window, change the **id** property of the **form** element to **MainForm**.

```
<form id="MainForm" runat="server">
```

- Reference the **Site.css** file in the **Site.master** Web Form, relative to the root folder, by placing the following markup next to the closing tag of the **title** element.

```
<link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />
```

- In the Site.master window, add a **Class** property to the **body** element by using the following markup.

```
<body class="template">
```

- In the Site.master window, add a **Class** property to the **div** element by using the following markup.

```
<div class="content">
```

- In the Site.master window, set the value of the **title** element to **Contoso Customer Management** by using the following code.

```
<title>Contoso Customer Management</title>
```

▶ Task 3: Define a ContentPlaceHolder control on the master page

- Remove the **ContentPlaceHolder** from the **head** element.

```
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
```

- Change the **id** property of the **ContentPlaceHolder** control within the **div** element to **MainContentPlaceHolder**.

```
<div class="content">
<asp:ContentPlaceHolder id="MainContentPlaceHolder"
runat="server">
</asp:ContentPlaceHolder>

</div>
```

- Save the master page.

Results: After completing this exercise, you will have created a master page named **Site.master**, and defined a **ContentPlaceHolder** control on the master page.

Exercise 2: Converting Web Forms to Content Pages and User Controls

The main tasks for this exercise are as follows:

1. Convert the default Web Form into a content page.
2. Add navigation to the master page.
3. Convert the Web Form into a user control.
4. Create a content page and insert a user control.

▶ Task 1: Convert the default Web Form into a content page

- Open the **Default.aspx** Web Form.
- In the Default.aspx window, in the **Page** directive, add a **MasterPageFile** property, with a value of **~/Site.master**, by using the following code.

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="Default.aspx.vb"
Inherits="_Default" MasterPageFile="~/Site.master"%>
```

- Remove the top-level HTML elements from the **Default** Web Form.

Note: You should take care not to delete the **div** element and its content within the **form** element.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="Styles/Site.css" rel="stylesheet" type="text/css"
/>
</head>
<body>
<form id="form1" runat"server">
</form>
</body>
</html>
```

- In the Default.aspx window, add a server-side **Content** control.

```
<asp:Content ID="MainContent"
ContentPlaceHolderID="MainContentPlaceHolder" runat="server">
</asp:Content>
```

- In the Default.aspx window, move the following code and it after the opening tag of the **form** element on the **Site.master** master page.

```
<div class="appTitle">
    <asp:Literal ID="AppTitleLiteral" runat="server"
Text="Customer Management"></asp:Literal>
</div>
```

- Format the Site.master master page, by pressing CTRL+K, and then pressing CTRL+D.
- Save the changes to the **Site.master** master page.
- Format the Default.aspx Web Form, by pressing CTRL+K, and then pressing CTRL+D.
- Save and close the **Default.aspx** Web Form.

▶ Task 2: Add navigation to the master page

- Add a breadcrumb to the master page by adding a **SiteMapPath** control named **MainSiteMapPath**, wrapped in a **div** element with a **class** attribute value of **siteMapPath**. Add the new **div** element below the existing **div** element with a **class** attribute value of **appTitle**.

```
<div class="siteMapPath">
    <asp:SiteMapPath ID="MainSiteMapPath" runat="server" />
</div>
```

- Add a menu to the master page by adding a **Menu** control named **MainMenu**, wrapped in a **div** element with a **class** attribute value of **menu**. Add the new **div** element below the existing **div** element with a **class** attribute value of **siteMapPath**.

```
<div class="menu">
    <asp:Menu ID="MainMenu" runat="server">
    </asp:Menu>
</div>
```

- Make the menu layout horizontal by applying the **Orientation** attribute.

```
Orientation="Horizontal"
```

- Ensure that the built-in image that indicates if a static menu item has a child menu is not displayed, by setting the **StaticEnableDefaultPopOutImage** attribute.

```
StaticEnableDefaultPopOutImage="false"
```

- Get the items for the **Menu** control from the **MainSiteMapDataSource** data source control by applying the **DataSourceID** attribute.

```
DataSourceID="MainSiteMapDataSource"
```

- Add the following child elements to the **Menu** control, by placing them between the opening and closing **Menu** tags.

```
<StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px"
/>
<StaticHoverStyle BackColor="White" ForeColor="Black" />
<DynamicHoverStyle BackColor="White" ForeColor="Black" />
<DynamicMenuItemStyle ItemSpacing="2px" HorizontalPadding="5px"
VerticalPadding="2px" />
```

- Add a **SiteMapDataSource** control named **MainSiteMapDataSource** to the master page, after the closing tag of the **div** element, with a **class** attribute value of **menu**. The **SiteMapDataSource** control should not show the starting node.

```
<asp:SiteMapDataSource ID="MainSiteMapDataSource" runat="server"
ShowStartingNode="false" />
```

- Format the **Site.master** master page.
- Add the **D:\Labfiles\Starter\M5\web.sitemap** site map file to the project.
- Modify the **div.menu** style, by using the Manage Styles window. The menu style must have the following definition:
  - Category: **Position**
  - position: **relative**
  - z-index: **1**
  - Top: **62px**
- Add a **siteMapPath** style, by using the Manage Styles window. The siteMapPath style must have the following definition:
  - Selector: **div.siteMapPath**
  - Define in: **Styles/Site.css**
  - Category: **Position**
  - Position: **fixed**
  - Top: **42px**
  - Category: **Box**
  - Padding section: Remove the selection from the **Same for all** check box; in the **bottom** box, type **5px**
- Save all modified files and run the **Site.master** master page.

▶ Task 3: Convert the Web Form into a user control

- Open the **InsertCustomer.aspx** Web Form, and change its **Page** directive to a **Control** directive.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="InsertCustomer.aspx.vb" Inherits="InsertCustomer" %>
```

- Add a **ClassName** property with a value of **Customer** to the **Control** directive.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="InsertCustomer.aspx.vb" Inherits="InsertCustomer"
ClassName="Customer" %>
```

- Change the **Inherits** property value from **InsertCustomer** to **Customer**.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="InsertCustomer.aspx.vb" Inherits="Customer"
ClassName="Customer" %>
```

- Change the **CodeFile** property value from **InsertCustomer.aspx.vb** to **Customer.aspx.vb**.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="Customer.aspx.vb" Inherits="Customer"
ClassName="Customer" %>
```

- Remove all the top-level HTML elements, such as the **DOCTYPE**, **html**, **head**, **body**, **title**, **link**, and **form** elements.
- Format the document.

Note: After removing all top-level HTML elements, you can view the following markup in the InsertCustomer.aspx window.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="Customer.ascx.vb" Inherits="Customer"
    ClassName="Customer" %>
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerFirstNameLabel" runat="server"
Text="First Name:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerFirstNameTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerLastNameLabel" runat="server"
Text="Last Name:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerLastNameTextBox"
runat="server"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerAddressLabel" runat="server"
Text="Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerAddressTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerZipCodeLabel" runat="server"
Text="Zip Code:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerZipCodeTextBox"
runat="server" MaxLength="10"></asp:TextBox>
        </div>
    </div>
```

(*Code continued on the following page.*)

```
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCityLabel" runat="server"
Text="City:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerCityTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerStateLabel" runat="server"
Text="State:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerStateTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCountryLabel" runat="server"
Text="Country:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:DropDownList ID="CustomerCountryDropDownList"
runat="server">
            </asp:DropDownList>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerPhoneLabel" runat="server"
Text="Phone:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerPhoneTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerEmailAddressLabel"
runat="server" Text="Email Address:"></asp:Label>
        </div>
```

(*Code continued on the following page.*)

```
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerEmailAddressTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerWebAddressLabel" runat="server"
Text="Web Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerWebAddressTextBox"
runat="server" MaxLength="80"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCreditLimitLabel"
runat="server" Text="Credit Limit:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerCreditLimitTextBox"
runat="server" MaxLength="10"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerNewsSubscriberLabel"
runat="server" Text="News Subscriber:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:CheckBox ID="CustomerNewsSubscriberCheckBox"
runat="server" />
        </div>
    </div>
    <div class="customerTableFooter">
        <asp:Button ID="CustomerInsertButton" runat="server"
Text="Insert" />
         <asp:Button ID="CustomerCancelButton" runat="server"
Text="Cancel" />
    </div>
</div>
```

- Save the **InsertCustomer.aspx** Web Form.
- Change the Web Form name from **InsertCustomer.aspx** to **Customer.ascx**.

- Open the **Customer.ascx.vb** user control code-behind file, change the class name to **Customer**, and change its base class from **System.Web.UI.Page** to **System.Web.UI.UserControl**.

```
Partial Class InsertCustomer
    Inherits System.Web.UI.UserControl
```

- Move the content from the **Page_LoadComplete** event method, and append it to the **Page_Load** event method.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Instantiate Customer
    instantiateCustomerObject()
    ' Populate the UI controls
    populateUI()
End Sub
```

- Remove the **Page_LoadComplete** event method.

```
''' <summary>
''' Populates UI controls
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_LoadComplete(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.LoadComplete
End Sub
```

- Save the modified files, and close the **Customer.ascx.vb** user control.

▶ Task 4: Create a content page and insert a user control

- Add a new content page named **InsertCustomer.aspx**, with a code-behind file based on the **Site.master** master page.
- Open the **InsertCustomer.aspx** content page in the Design view, and drag the **Customer.ascx** user control to the **MainContentPlaceHolder** control.
- Run the **CustomerManagement** Web application.
- Verify the Contoso Customer Management Web site, by clicking **New** on the **Customers** menu.

Note: Notice that the new user control displays on the InsertCustomer Web Form.

▶ Task 5: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

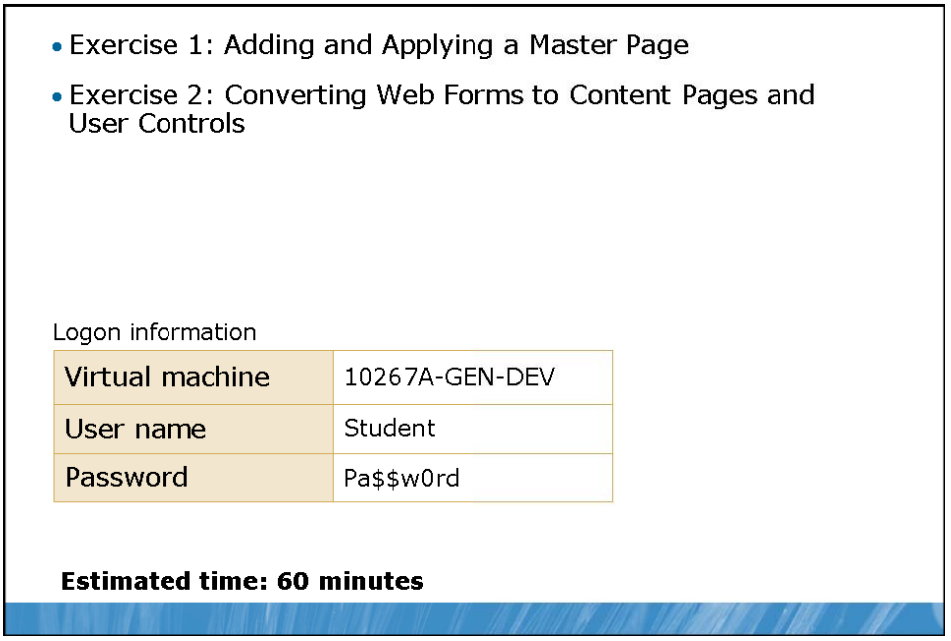
Results: After completing this exercise, you will have converted the default Web Form into a content page, added navigation to the master page, and then converted the Web Form into a user control. In addition, you should have created a content page and inserted a user control.

Module 5

Lab Instructions: Implementing Master Pages and User Controls (Visual C#)

Contents:

Lab: Implementing Master Pages and User Controls



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in **Section 1** of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in **Section 2** of the lab document.

Introduction

In this lab, you will implement master pages and user controls in an ASP.NET Web project. In addition, you will add navigation to the master page and convert a Web Form to a user control.

Objectives

After completing this lab, you will be able to:

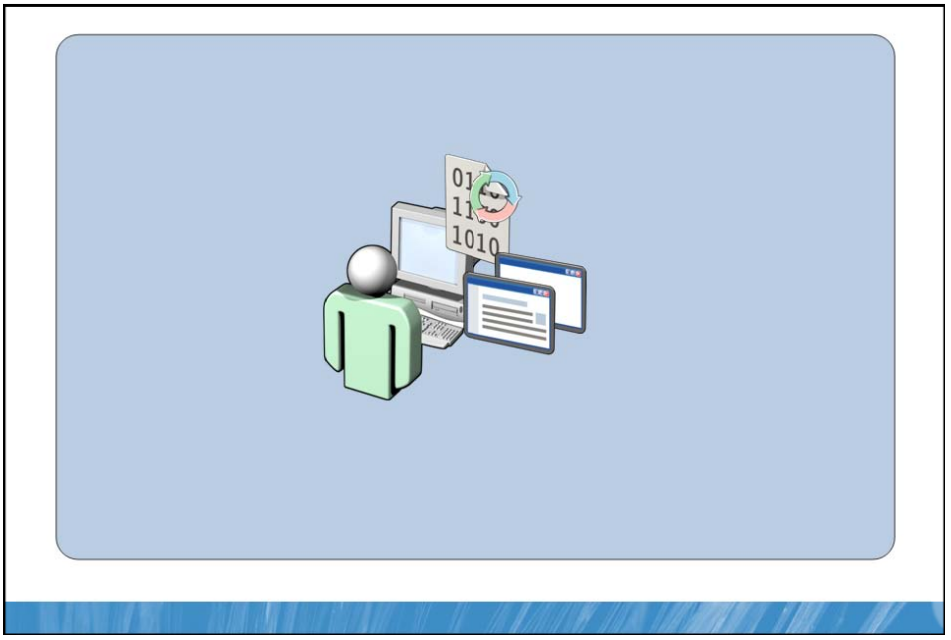
- Add and apply a master page in a Web application.
- Convert a Web Form to a content page.
- Add navigation to the master page.
- Convert a Web Form into a user control.
- Create a content page.
- Insert a user control on the content page.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. The organization has decided to modify their Web site to provide consistent user experience to all customers.

To do this, you need to add a master page and convert an existing Web Form into a user control for reuse. In addition, you will need to add navigation to the master page. A senior developer has already created some of the code to add navigation to the Web site, and the developer has created a sitemap XML document. You can use the code and the document for adding navigation to the master page. In addition, you need to convert a Web Form into a content page, move the content from the content page to the master page, and create a new content page with a user control.

Section 2: Visual C#

Exercise 1: Adding and Applying a Master Page

The main tasks for this exercise are as follows:

1. Add a master page to an existing Web site.
2. Initialize the style controls and elements on the master page.
3. Define a **ContentPlaceHolder** control on the master page.

▶ Task 1: Add a master page to an existing Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M5\CS folder.
- Add a new master page named **Site.master**, to the **CustomerManagement** Web site.

▶ Task 2: Initialize the style controls and elements on the master page

- In the Site.master window, add an **id** property to the **head** element by using the following code.

```
<head runat="server" id="MainHead">
```

- In the Site.master window, change the **id** property of the **form** element to **MainForm**.

```
<form id="MainForm" runat="server">
```

- Reference the **Site.css** file in the **Site.master** Web Form, relative to the root folder, by placing the following markup next to the closing tag of the **title** element.

```
<link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />
```

- In the Site.master window, add a **Class** property to the **body** element by using the following markup.

```
<body class="template">
```

- In the Site.master window, add a **Class** property to the **div** element by using the following markup.

```
<div class="content">
```

- In the Site.master window, set the value of the **title** element to **Contoso Customer Management** by using the following code.

```
<title>Contoso Customer Management</title>
```

▶ Task 3: Define a ContentPlaceHolder control on the master page

- Remove the **ContentPlaceHolder** from the **head** element.

```
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
```

- Change the **id** property of the **ContentPlaceHolder** control within the **div** element to **MainContentPlaceHolder**.

```
<div class="content">
<asp:ContentPlaceHolder id="MainContentPlaceHolder"
runat="server">
</asp:ContentPlaceHolder>

</div>
```

- Save the master page.

Results: After completing this exercise, you will have created a master page named **Site.master**, and defined a **ContentPlaceHolder** control on the master page.

Exercise 2: Converting Web Forms to Content Pages and User Controls

The main tasks for this exercise are as follows:

1. Convert the default Web Form into a content page.
2. Add navigation to the master page.
3. Convert the Web Form into a user control
4. Create a content page and insert a user control.

▶ Task 1: Convert the default Web Form into a content page

- Open the **Default.aspx** Web Form.
- In the Default.aspx window, in the **Page** directive, add a **MasterPageFile** property with a value of **~/Site.master**, by using the following code.

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs"
Inherits="_Default" MasterPageFile="~/Site.master"%>
```

- Remove the top-level HTML elements from the **Default** Web Form.

Note: You should take care not to delete the **div** element and the content within the **form** element.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="Styles/Site.css" rel="stylesheet" type="text/css"
/>
</head>
<body>
<form id="form1" runat"server">
</form>
</body>
</html>
```

- In the Default.aspx window, add a server-side **Content** control.

```
<asp:Content ID="MainContent"
ContentPlaceHolderID="MainContentPlaceHolder" runat="server">
</asp:Content>
```

- In the Default.aspx window, move the following code and place the code after the opening tag of the **form** element on the **Site.master** master page.

```
<div class="appTitle">
    <asp:Literal ID="AppTitleLiteral" runat="server"
Text="Customer Management"></asp:Literal>
</div>
```

- Format the Site.master master page, by pressing CTRL+K, and then pressing CTRL+D.
- Save the changes to the **Site.master** master page.
- Format the Default.aspx Web Form, by pressing CTRL+K, and then pressing CTRL+D.
- Save and close the Default.aspx Web Form.

▶ Task 2: Add navigation to the master page

- Add a breadcrumb to the master page by adding a **SiteMapPath** control named **MainSiteMapPath**, wrapped in a **div** element with a **class** attribute value of **siteMapPath**. Add the new **div** element below the existing **div** element with a **class** attribute value of **appTitle**.

```
<div class="siteMapPath">
    <asp:SiteMapPath ID="MainSiteMapPath" runat="server" />
</div>
```

- Add a menu to the master page by adding a **Menu** control named **MainMenu**, wrapped in a **div** element with a **class** attribute value of **menu**. Add the new **div** element below the existing **div** element with a **class** attribute value of **siteMapPath**.

```
<div class="menu">
    <asp:Menu ID="MainMenu" runat="server">
    </asp:Menu>
</div>
```

- Make the menu layout horizontal, by applying the **Orientation** attribute.

```
Orientation="Horizontal"
```

- Ensure that the built-in image that indicates if a static menu item has a child menu is not displayed, by setting the **StaticEnableDefaultPopOutImage** attribute.

```
StaticEnableDefaultPopOutImage="false"
```

- Get the items for the **Menu** control from the **MainSiteMapDataSource** data source control by applying the **DataSourceID** attribute.

```
DataSourceID="MainSiteMapDataSource"
```

- Add the following child elements to the **Menu** control, by placing them between the opening and closing **Menu** tags.

```
<StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px"
/>
<StaticHoverStyle BackColor="White" ForeColor="Black" />
<DynamicHoverStyle BackColor="White" ForeColor="Black" />
<DynamicMenuItemStyle ItemSpacing="2px" HorizontalPadding="5px"
VerticalPadding="2px" />
```

- Add a **SiteMapDataSource** control named **MainSiteMapDataSource** to the master page, after the closing tag of the **div** element, with a **class** attribute value of **menu**. The **SiteMapDataSource** control should not show the starting node.

```
<asp:SiteMapDataSource ID="MainSiteMapDataSource" runat="server"
ShowStartingNode="false" />
```

- Format the Site.master master page.
- Add the **D:\Labfiles\Starter\M5\web.sitemap** site map file to the project.
- Modify the **div.**menu style, by using the Manage Styles window. The menu style must have the following definition:
  - Selector: **div.menu**
  - Define in: **Styles/Site.css**
  - Category: **Position**

    - Position: **relative**
    - Z-order: **1**
    - Top: **62px**
- Add a **siteMapPath** style, by using the **Manage Styles** window. The siteMapPath style must have the following definition:
    - Selector: **div.siteMapPath**
    - Define in: **Styles/Site.css**
    - Category: **Position**
    - Position: **fixed**
    - Top: **42px**
    - Category: **Box**
    - Padding section: Remove the selection from the **Same for all** check box; in the **bottom** box, type **5px**
- Save all modified files, and run the **Site.master** master page.

▶ Task 3: Convert the Web Form into a user control

- Open the **InsertCustomer.aspx** Web Form, and change its **Page** directive to a **Control** directive.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="InsertCustomer" %>
```

- Add a **ClassName** property with a value of **Customer** to the **Control** directive.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="InsertCustomer"
ClassName="Customer" %>
```

- Change the **Inherits** property value from **InsertCustomer** to **Customer**.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="Customer"
ClassName="Customer" %>
```

- Change the **CodeFile** property value from **InsertCustomer.aspx.cs** to **Customer.aspx.cs**.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Customer.aspx.cs" Inherits="Customer"
ClassName="Customer" %>
```

- Remove all the top-level HTML elements, such as the **DOCTYPE**, **html**, **head**, **body**, **title**, **link**, and **form** elements.
- Format the document.

Note: After removing all top-level HTML elements, you can view the following markup in the InsertCustomer.aspx window.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Customer.ascx.cs" Inherits="Customer"
    ClassName="Customer" %>
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerFirstNameLabel" runat="server"
Text="First Name:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerFirstNameTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerLastNameLabel" runat="server"
Text="Last Name:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerLastNameTextBox"
runat="server"></asp:TextBox>
        </div>
    </div>
```

(*Code continued on the following page.*)

```
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerAddressLabel" runat="server"
Text="Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerAddressTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerZipCodeLabel" runat="server"
Text="Zip Code:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerZipCodeTextBox"
runat="server" MaxLength="10"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCityLabel" runat="server"
Text="City:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerCityTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerStateLabel" runat="server"
Text="State:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerStateTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCountryLabel" runat="server"
Text="Country:"></asp:Label>
        </div>
```

(*Code continued on the following page.*)

```
        <div class="customerTableRightCol">
            <asp:DropDownList ID="CustomerCountryDropDownList"
runat="server">
            </asp:DropDownList>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerPhoneLabel" runat="server"
Text="Phone:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerPhoneTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerEmailAddressLabel"
runat="server" Text="Email Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerEmailAddressTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerWebAddressLabel" runat="server"
Text="Web Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerWebAddressTextBox"
runat="server" MaxLength="80"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCreditLimitLabel"
runat="server" Text="Credit Limit:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerCreditLimitTextBox"
runat="server" MaxLength="10"></asp:TextBox>
        </div>
    </div>
```

(*Code continued on the following page.*)

```
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerNewsSubscriberLabel"
runat="server" Text="News Subscriber:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:CheckBox ID="CustomerNewsSubscriberCheckBox"
runat="server" />
        </div>
    </div>
    <div class="customerTableFooter">
        <asp:Button ID="CustomerInsertButton" runat="server"
Text="Insert" OnClick="CustomerInsertButton_Click" />
         <asp:Button ID="CustomerCancelButton" runat="server"
Text="Cancel" OnClick="CustomerCancelButton_Click" />
    </div>
</div>
```

- Save the **InsertCustomer.aspx** Web Form.
- Change the Web Form name from **InsertCustomer.aspx** to **Customer.ascx**.
- Open the **Customer.ascx.cs** user control code-behind file, change the class name to Customer, and change its base class from **System.Web.UI.Page** to **System.Web.UI.UserControl**.

```
public partial class Customer : System.Web.UI.UserControl
```

- Move the content from the **Page_LoadComplete** event method, and append it to the **Page_Load** event method.

```
protected void Page_Load(object sender, EventArgs e)
{
    // Instantiate Customer
    instantiateCustomerObject();
    // Populate the UI controls
    populateUI();
}
```

- Remove the **Page_LoadComplete** event method.

```
/// <summary>
/// Populates UI controls
/// </summary>
/// <param name="sender">>/param>
/// <param name="e"></param>
protected void Page__LoadComplete(object sender, EventArgs e)
{
}
```

- Save the modified files and close the **Customer.ascx.cs** user control.

▶ Task 4: Create a content page and insert a user control

- Add a new content page named **InsertCustomer.aspx**, with a code-behind file based on the **Site.master** master page.
- Open the **InsertCustomer.aspx** content page in the Design view, and drag the **Customer.ascx** user control to the **MainContentPlaceHolder** control.
- Run the **CustomerManagement** Web application.
- Verify the Contoso Customer Management Web site by clicking **New** on the **Customers** menu.

Note: Notice that the new user control displays on the InsertCustomer Web Form.

▶ Task 5: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

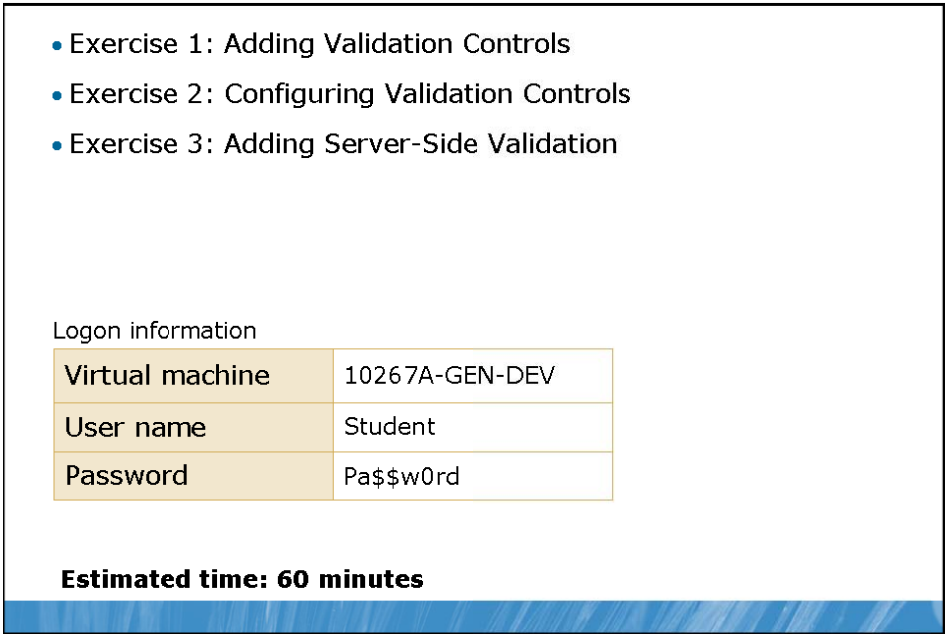
Results: After completing this exercise, you will have converted the default Web Form to a content page, added navigation to the master page, and then converted the Web Form into a user control. In addition, you will have created a content page and inserted a user control.

Module 6

Lab Instructions: Validating User Input (Visual Basic)

Contents:

Lab: Validating User Input



Note: You can perform tasks in this lab either by using either the Microsoft Visual Basic® or Microsoft Visual C#® programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Lab Introduction

In this lab, you will add and configure validation controls in a user control to help you ensure valid user inputs in a Web project. You will use client-side scripting, which will help you avoid frequent cross-checking of user inputs with the server. In addition, you will perform server-side validation to protect the Web project against spoofing and malicious code.

Lab Objectives

After completing this lab, you will be able to:

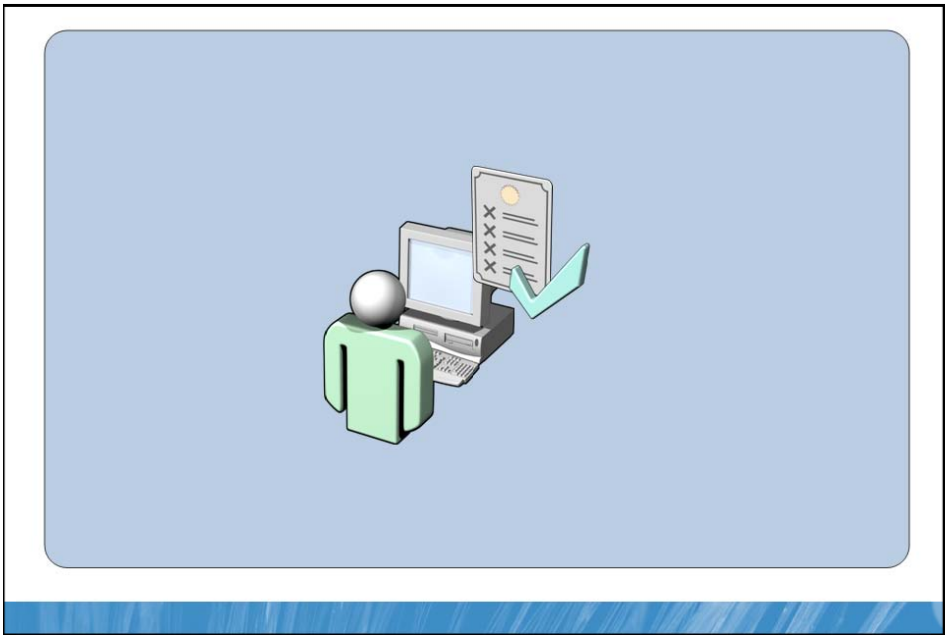
- Add validation controls.
- Configure validation controls.
- Add server-side validation.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
    - User name: **Student**
    - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. For effective communication, the organization should maintain updated customer information in their database. To meet this requirement, you need to add and configure validation controls to the Customer Management project without causing administrative overheads or performance issues from frequent cross-checking with the server.

Section 1: Visual Basic

Exercise 1: Adding Validation Controls

The main tasks for this exercise are as follows:

1. Open an existing Web site.
2. Add validation controls to the user control.

▶ Task 1: Open an existing Web site

- Log on to **10267A-GEN-DEV** virtual machine as **Student** with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M6\VB folder.

▶ Task 2: Add validation controls to the user control

- View the markup of the **Customer** user control.
- Add a **RequiredFieldValidator** control named **CustomerFirstNameRequiredFieldValidator**, for the **CustomerFirstNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerFirstNameRequiredFieldValidator"
ControlToValidate="CustomerFirstNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerLastNameRequiredFieldValidator**, for the **CustomerLastNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerLastNameRequiredFieldValidator"
ControlToValidate="CustomerLastNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerAddressRequiredFieldValidator**, for the **CustomerAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerAddressRequiredFieldValidator"
ControlToValidate="CustomerAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerZipCodeRequiredFieldValidator**, for the **CustomerZipCodeTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerZipCodeRequiredFieldValidator"
ControlToValidate="CustomerZipCodeTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerCityRequiredFieldValidator**, for the **CustomerCityTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCityRequiredFieldValidator"
ControlToValidate="CustomerCityTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerCountryRequiredFieldValidator**, for the **CustomerCountryDropDownList** control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerWebAddressRequiredFieldValidator**, for the **CustomerWebAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerCreditLimitRequiredFieldValidator**, for the **CustomerCreditLimitTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RegularExpressionValidator** control named **CustomerEmailAddressRegularExpressionValidator**, for the **CustomerEmailAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpressionV
alidator>
```

- Add a **RegularExpressionValidator** control named **CustomerWebAddressRegularExpressionValidator**, for the **CustomerWebAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpressionV
alidator>
```

- Add a **RangeValidator** control named **CustomerCreditLimitRangeValidator**, for the **CustomerCreditLimitTextBox** control.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000"
ErrorMessage="RangeValidator"></asp:RangeValidator>
```

- Add a **ValidationSummary** control named **CustomerValidationSummary**, for the **CustomerInsertButton** control.

```
<asp:ValidationSummary ID="CustomerValidationSummary"
runat="server"></asp:ValidationSummary>
```

- Format the Customer.ascx user control.
- Save the **Customer** user control, and view the changes in the browser.
- Test the functionality of the **Customer** user control.
- Close Windows® Internet Explorer®.

Result: After completing this exercise, you will have added validation controls to a user control.

Exercise 2: Configuring Validation Controls

The main tasks for this exercise are as follows:

1. Remove validation from the **Cancel** button.
2. Add error indicators and error messages to the validation controls.
3. Set the e-mail address and credit limit validation controls.

▶ Task 1: Remove validation from the Cancel button

- View the default **Response.Redirect** method of the **Cancel** button.

```
''' <summary>
''' Redirects to home page
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerCancelButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CustomerCancelButton.Click
    ' Redirect to home page
    Response.Redirect("~/Default.aspx")
End Sub
```

- Disable the validation caused by the **CustomerCancelButton** control by setting the **CausesValidation** property in the user control to **false**.

```
<asp:Button ID="CustomerCancelButton" runat="server" Text="Cancel"
CausesValidation="false" />
```

- Save the **Customer** user control, and view the changes in the browser.

Note: Notice that the Web browser is redirected to the default Web Form, instead of displaying the error messages.

▶ Task 2: Add error indicators and error messages to the validation controls

- Add a **Text** property with the value of * to the **CustomerFirstNameRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerFirstNameRequiredFieldValidator** control to **The Customer First Name must be filled in**.
- Add a **Text** property with the value of * to the **CustomerLastNameRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerLastNameRequiredFieldValidator** control to **The Customer Last Name must be filled in**.
- Add a **Text** property with the value of * to the **CustomerAddressRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerAddressRequiredFieldValidator** control to **The Address must be filled in**.
- Add a **Text** property with the value of * to the **CustomerZipCodeRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerZipCodeRequiredFieldValidator** control to **The Zip Code must be filled in**.
- Add a **Text** property with the value of * to the **CustomerCityRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerCityRequiredFieldValidator** control to **The City must be filled in**.
- Add a **Text** property with the value of * to the **CustomerCountryRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerCountryRequiredFieldValidator** control to **A country must be selected**.
- Add a **Text** property with the value of * to the **CustomerEmailAddressRegularExpressionValidator** control.

- Change the **ErrorMessage** property in the **CustomerEmailAddressRegularExpressionValidator** control to **The Email Address must be valid**.
- Add a **Text** property with the value of * to the **CustomerWebAddressRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerWebAddressRequiredFieldValidator** control to **The Web Address must be filled in**.
- Add a **Text** property with the value of * to the **CustomerWebAddressRegularExpressionValidator** control.
- Change the **ErrorMessage** property in the **CustomerWebAddressRegularExpressionValidator** control to **The Web Address must be valid**.
- Add a **Text** property with the value of * to the **CustomerCreditLimitRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerCreditLimitRequiredFieldValidator** control to **The Credit Limit must be filled in**.
- Add a **Text** property with the value of * to the **CustomerCreditLimitRangeValidator** control.
- Change the **ErrorMessage** property in the **CustomerCreditLimitRangeValidator** control to **The Credit Limit must be within the valid range**.
- Save the **Customer** user control, and view the changes in the browser.
- In the Contoso Customer Management – Windows Internet Explorer window, in the **Email Address** box, type **contoso.com**, press the TAB key, in the **Web Address** box, type **www.contoso**, and then press the TAB key again.

Note: If an AutoComplete message box displays, click **No**.

Note: Notice the error indicator that displays next to the **Email Address** and **Web Address** boxes, because of the invalid e-mail and web addresses.

- In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

Note: Notice the error indicator and error messages next to the **First Name**, **Last Name**, **Address**, **Zip Code**, **City**, **Country**, and **Credit Limit** controls.

- Close Windows® Internet Explorer®.

▶ Task 3: Set the e-mail address and credit limit validation controls

- View the **InsertCustomer.aspx** Web Form in a browser.
- In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

Note: Notice that the value for **Credit Limit** box is set to **0** and the error indicator text for the **Web Address** box is not aligned with the other error indictors. Also notice that the error message for the **Credit Limit** box is **The Credit Limit must be within the valid range**.

- Close Windows® Internet Explorer®.
- Add a **Display** property with the value of **Dynamic** to the **CustomerWebAddressRequiredFieldValidator** control to change the display of the error indicator text.
- Add a **Display** property with the value of **Dynamic** to the **CustomerCreditLimitRequiredFieldValidator** control to change the display of the error indicator text.
- Save the **Customer** user control, and view the changes in the browser.
- Click the **Insert** button.

Note: Notice that the location of the error indicator for the **Credit Limit** box has changed. However, the error message for the **Credit Limit** box is still **The Credit Limit must be within the valid range**.

- Close Windows® Internet Explorer®.

- Add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*** to the **CustomerEmailAddressRegularExpressionValidator** control.
- Add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*** to the **CustomerWebAddressRegularExpressionValidator** control.
- Set the **Type** property with the value of **Integer** to the **CustomerCreditLimitRangeValidator** control.
- Change the **MinimumValue** property of the **CustomerCreditLimitRangeValidator** control to **0**.
- Save the **Customer** user control, and view the changes in the browser.
- In the Contoso Customer Management – Windows Internet Explorer window, in the **Email Address** box, type **claus@contoso.com**, in the **Web Address** box, type **http://www.contoso.com**, and then click the **Insert** button.

Note: Notice that the error indicator and error messages do not display for the **Email Address**, **Web Address**, and **Credit Limit** boxes.

Results: After completing this exercise, you will have removed validation control from the **Cancel** button, and added error indicators and error messages to the validation controls.

Exercise 3: Adding Server-Side Validation

The main task for this exercise is to validate the Customer user control.

▶ Task 1: Validate the Customer User Control

- Open the Customer.ascx.vb code window.
- Add the code to validate the user control within the **CustomerInsertButton_Click** event handler method.

```
''' <summary>
''' Saves the current customer information and adds default values
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerInsertButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CustomerInsertButton.Click
    ' Did page validation succeed?
    If Not Page.IsValid Then
        Return
    End If

    ' Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name
    ' Add the user credit limit
    currentCustomer.CreditLimit = 50000
End Sub
```

- Add postback validation to the **Page_Load** event handler method.

```
''' <summary>
''' Instantiates Customer object
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    If Page.IsPostBack Then
        ' Validate Page
        Page.Validate()

        ' Did page validation succeed?
        If Not Page.IsValid Then
            Return
        End If
    End If

    ' Instantiate Customer
    instantiateCustomerObject()
    ' Populate the UI controls
    populateUI()
End Sub
```

- Disable the client-side validation for the **CustomerCountryDropDownList** control by setting the **EnableClientScript** property of the **CustomerCountryRequiredFieldValidator** control to **false**.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected." Text="*"
EnableClientScript="false"></asp:RequiredFieldValidator>
```

- Save the **Customer** user control, and view the changes in the browser.
- In the Contoso Customer Management – Windows Internet Explorer, type the following settings, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**

- Zip Code: **98052**
- City: **Buffalo**
- State: **NY**
- Web Address: **http://www.cohowinery.com**

Note: Notice the postback of the Web page with the inputs. Also notice that after the postback, the error indicator for the **Country** list and associated error message display.

- Close Windows® Internet Explorer®.
- Remove the **EnableClientScript** property for the **CustomerCountryRequiredFieldValidator** control to enable the client-side validation for the **CustomerCountryDropDownList** control, and format and save the **Customer** user control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected."
Text="*"></asp:RequiredFieldValidator>
```

- In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 2: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

Results: After completing this exercise, you will have added server-side validation controls to the **Customer** user control.

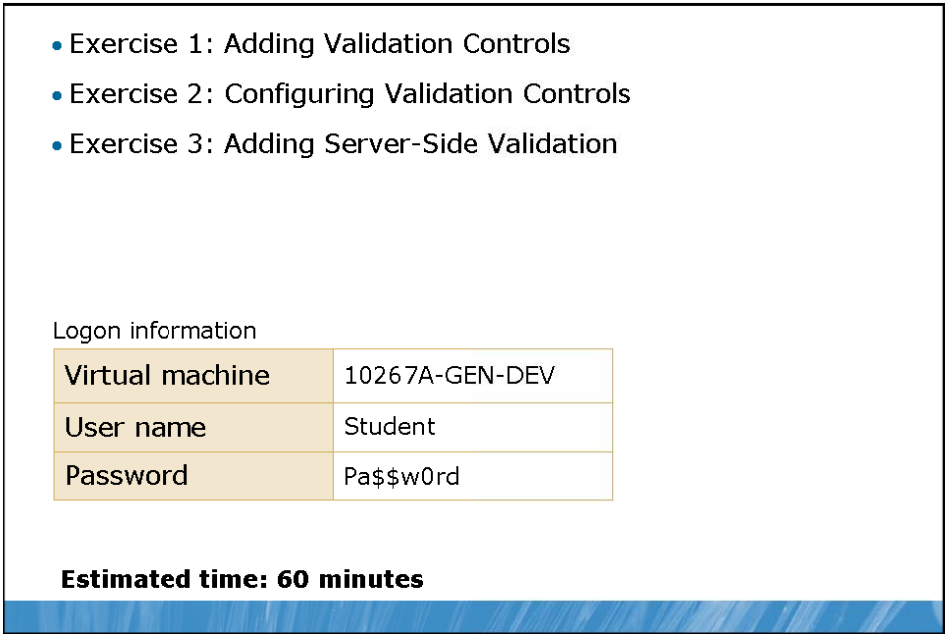
Note: The answers to the exercises are on the Course Companion CD.

Module 6

Lab Instructions: Validating User Input (Visual C#)

Contents:

Lab: Validating User Input



Note: You can perform tasks in this lab either by using either the Microsoft Visual Basic® or Microsoft Visual C#® programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Lab Introduction

In this lab, you will add and configure validation controls in a user control to help you ensure valid user inputs in a Web project. You will use client-side scripting, which will help you avoid frequent cross-checking of user inputs with the server. In addition, you will perform server-side validation to protect the Web project against spoofing and malicious code.

Lab Objectives

After completing this lab, you will be able to:

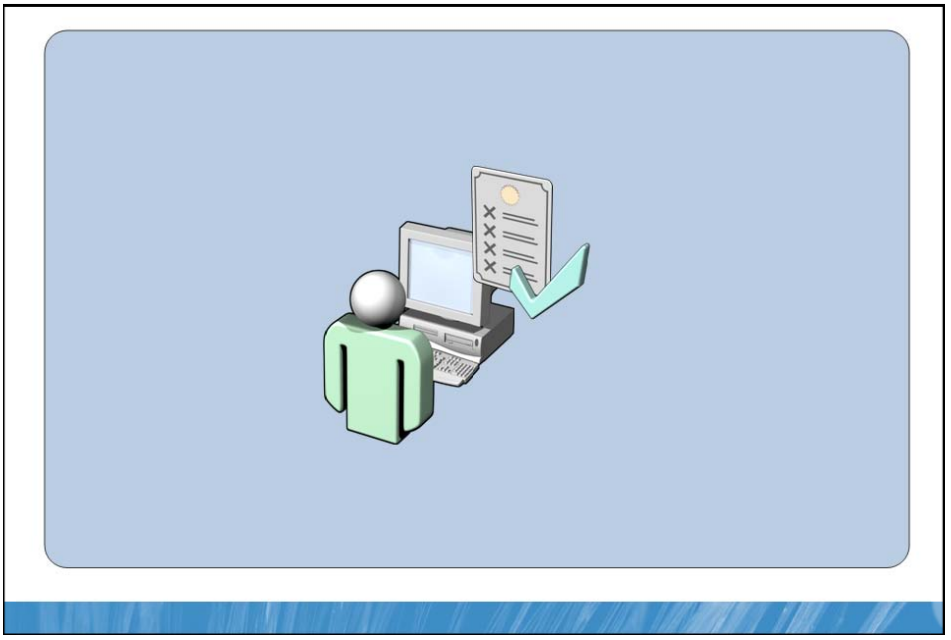
- Add validation controls.
- Configure validation controls.
- Add server-side validation.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
    - User name: **Student**
    - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. For effective communication, the organization should maintain updated customer information in their database. To meet this requirement, you need to add and configure validation controls to the Customer Management project without causing administrative overheads or performance issues from frequent cross-checking with the server.

Section 2: Visual C#

Exercise 1: Adding Validation Controls

The main tasks for this exercise are as follows:

1. Open an existing Web site.
2. Add validation controls to the user control.

▶ Task 1: Open an existing Web site

- Log on to **10267A-GEN-DEV** virtual machine as **Student** with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M6\CS folder.

▶ Task 2: Add validation controls to the user control

- View the markup of the **Customer** user control.
- Add a **RequiredFieldValidator** control named **CustomerFirstNameRequiredFieldValidator**, for the **CustomerFirstNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerFirstNameRequiredFieldValidator"
ControlToValidate="CustomerFirstNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerLastNameRequiredFieldValidator**, for the **CustomerLastNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerLastNameRequiredFieldValidator"
ControlToValidate="CustomerLastNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerAddressRequiredFieldValidator**, for the **CustomerAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerAddressRequiredFieldValidator"
ControlToValidate="CustomerAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerZipCodeRequiredFieldValidator**, for the **CustomerZipCodeTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerZipCodeRequiredFieldValidator"
ControlToValidate="CustomerZipCodeTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerCityRequiredFieldValidator**, for the **CustomerCityTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCityRequiredFieldValidator"
ControlToValidate="CustomerCityTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerCountryRequiredFieldValidator**, for the **CustomerCountryDropDownList** control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerWebAddressRequiredFieldValidator**, for the **CustomerWebAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RequiredFieldValidator** control named **CustomerCreditLimitRequiredFieldValidator**, for the **CustomerCreditLimitTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

- Add a **RegularExpressionValidator** control named **CustomerEmailAddressRegularExpressionValidator**, for the **CustomerEmailAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpressionV
alidator>
```

- Add a **RegularExpressionValidator** control named **CustomerWebAddressRegularExpressionValidator**, for the **CustomerWebAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpressionV
alidator>
```

- Add a **RangeValidator** control named **CustomerCreditLimitRangeValidator**, for the **CustomerCreditLimitTextBox** control. The minimum value is **500** and the maximum value is **50000**.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000"
ErrorMessage="RangeValidator"></asp:RangeValidator>
```

- Add a **ValidationSummary** control named **CustomerValidationSummary**, for the **CustomerInsertButton** control.

```
<asp:ValidationSummary ID="CustomerValidationSummary"
runat="server"></asp:ValidationSummary>
```

- Format the **Customer.ascx** user control.
- Save the **Customer** user control, and view the changes in the browser.
- Test the functionality of the **Customer** user control.
- Close Windows® Internet Explorer®.

Result: After completing this exercise, you will have added validation controls to a user control.

Exercise 2: Configuring Validation Controls

The main tasks for this exercise are as follows:

1. Remove validation from the Cancel button.
2. Add error indicators and error messages to the validation controls.
3. Set the e-mail address and credit limit validation controls.

▶ Task 1: Remove validation from the Cancel button

- View the default **Response.Redirect** method of the **Cancel** button.

```
/// <summary>
/// Redirects to home page
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerCancelButton_Click(object sender, EventArgs
e)
{
    // Redirect to home page
    Response.Redirect("~/Default.aspx");
}
```

- Disable the validation caused by the **CustomerCancelButton** control by setting the **CausesValidation** property in the user control to **false**.

```
<asp:Button ID="CustomerCancelButton" runat="server" Text="Cancel"
OnClick="CustomerCancelButton_Click" CausesValidation="false" />
```

- Save the **Customer** user control and view the changes in the browser.

Note: Notice that the Web browser is redirected to the default Web Form, instead of displaying the error messages.

▶ Task 2: Add error indicators and error messages to the validation controls

- Add a **Text** property with the value of * to the **CustomerFirstNameRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerFirstNameRequiredFieldValidator** control to **The Customer First Name must be filled in**.
- Add a **Text** property with the value of * to the **CustomerLastNameRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerLastNameRequiredFieldValidator** control to **The Customer Last Name must be filled in**.
- Add a **Text** property with the value of * to the **CustomerAddressRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerAddressRequiredFieldValidator** control to **The Address must be filled in**.
- Add a **Text** property with the value of * to the **CustomerZipCodeRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerZipCodeRequiredFieldValidator** control to **The Zip Code must be filled in**.
- Add a **Text** property with the value of * to the **CustomerCityRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerCityRequiredFieldValidator** control to **The City must be filled in**.
- Add a **Text** property with the value of * to the **CustomerCountryRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerCountryRequiredFieldValidator** control to **A country must be selected**.
- Add a **Text** property with the value of * to the **CustomerEmailAddressRegularExpressionValidator** control.

- Change the **ErrorMessage** property in the **CustomerEmailAddressRegularExpressionValidator** control to **The Email Address must be valid**.
- Add a **Text** property with the value of * to the **CustomerWebAddressRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerWebAddressRequiredFieldValidator** control to **The Web Address must be filled in**.
- Add a **Text** property with the value of * to the **CustomerWebAddressRegularExpressionValidator** control.
- Change the **ErrorMessage** property in the **CustomerWebAddressRegularExpressionValidator** control to **The Web Address must be valid**.
- Add a **Text** property with the value of * to the **CustomerCreditLimitRequiredFieldValidator** control.
- Change the **ErrorMessage** property in the **CustomerCreditLimitRequiredFieldValidator** control to **The Credit Limit must be filled in**.
- Add a **Text** property with the value of * to the **CustomerCreditLimitRangeValidator** control.
- Change the **ErrorMessage** property in the **CustomerCreditLimitRangeValidator** control to **The Credit Limit must be within the valid range**.
- Save the **Customer** user control and view the changes in the browser.
- In the Contoso Customer Management – Windows Internet Explorer window, in the **Email Address** box, type **contoso.com**, press the TAB key, in the **Web Address** box, type **www.contoso**, and then press the TAB key.

Note: If you see the AutoComplete message box display, click the **No** button.

Note: Notice the error indicator that displays next to the **Email Address** and **Web Address** boxes, because of the invalid e-mail and web addresses.

- In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

Note: Notice the error indicator and error messages next to the **First Name**, **Last Name**, **Address**, **Zip Code**, **City**, **Country**, and **Credit Limit** controls.

- Close Windows® Internet Explorer®.

▶ Task 3: Set the e-mail address and credit limit validation controls

- View the InsertCustomer.aspx Web Form in a browser.
- In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

Note: Notice that the value for **Credit Limit** box is set to **0**, and the error indicator text for the **Web Address** box is not aligned with the other error indictors. Also notice that the error message for the **Credit Limit** box is **The Credit Limit must be within the valid range**.

- Close Windows® Internet Explorer®.
- Add a **Display** property with the value of **Dynamic** to the **CustomerWebAddressRequiredFieldValidator** control to change the display of the error indicator text.
- Add a **Display** property with the value of **Dynamic** to the **CustomerCreditLimitRequiredFieldValidator** control to change the display of the error indicator text.
- Save the **Customer** user control, and view the changes in the browser.
- Click the **Insert** button.

Note: Notice that the location of the error indicator for the **Credit Limit** box has changed. However, the error message for the **Credit Limit** box is still **The Credit Limit must be within the valid range**.

- Close Windows® Internet Explorer®.

- Add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*** to the **CustomerEmailAddressRegularExpressionValidator** control.
- Add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*** to the **CustomerWebAddressRegularExpressionValidator** control.
- Set the **Type** property with the value of **Integer** to the **CustomerCreditLimitRangeValidator** control.
- Change the **MinimumValue** property of the **CustomerCreditLimitRangeValidator** control to **0**.
- Save the **Customer** user control and view the changes in the browser.
- In the Contoso Customer Management – Windows Internet Explorer window, in the **Email Address** box, type **claus@contoso.com**, in the **Web Address** box, type **http://www.contoso.com**, and then click the **Insert** button.

Note: Notice that the error indicator and error messages do not display for the **Email Address**, **Web Address**, and **Credit Limit** boxes.

Results: After completing this exercise, you will have removed validation control from the **Cancel** button, and added error indicators and error messages to the validation controls.

Exercise 3: Adding Server-Side Validation

The main task for this exercise is to validate the Customer user control.

▶ Task 1 : Validate the Customer User Control

- Open the Customer.ascx.cs code window.
- Add the code to validate the user control within the **CustomerInsertButton_Click** event handler method.

```
/// <summary>
/// Saves the current customer information and adds default values
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerInsertButton_Click(object sender, EventArgs
e)
{
    // Did page validation succeed?
    if (!Page.IsValid)
        return;

    // Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name;
    // Add the user credit limit
    currentCustomer.CreditLimit = 50000;
}
```

- Add postback validation to the **Page_Load** event handler method.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.IsPostBack)
    {
        // Validate Page
        Page.Validate();

        // Did page validation succeed?
        if (!Page.IsValid)
            return;
    }

    // Instantiate Customer
    instantiateCustomerObject();
    // Populate the UI controls
    populateUI();
}
```

- Disable the client-side validation for the **CustomerCountryDropDownList** control, by setting the **EnableClientScript** property of the **CustomerCountryRequiredFieldValidator** control to **false**.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected." Text="*"
EnableClientScript="false"></asp:RequiredFieldValidator>
```

- Save the **Customer** user control and view the changes in the browser.
- In the Contoso Customer Management – Windows Internet Explorer, type the following settings, and then click the **Insert** button.
    - First Name: **Claus**
    - Last Name: **Hansen**
    - Address: **4567 Main St.**
    - Zip Code: **98052**
    - City: **Buffalo**
    - State: **NY**
    - Web Address: **http://www.cohowinery.com**

Note: Notice the postback of the Web page with the inputs. Also notice that after the postback, the error indicator for the **Country** list and its associated error message displays.

- Close Windows® Internet Explorer®.
- Remove the **EnableClientScript** property for the **CustomerCountryRequiredFieldValidator** control to enable the client-side validation for the **CustomerCountryDropDownList** control, and format and save the **Customer** user control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected."
Text="*"></asp:RequiredFieldValidator>
```

- In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 2: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

Results: After completing this exercise, you will have added server-side validation controls to the **Customer** user control.

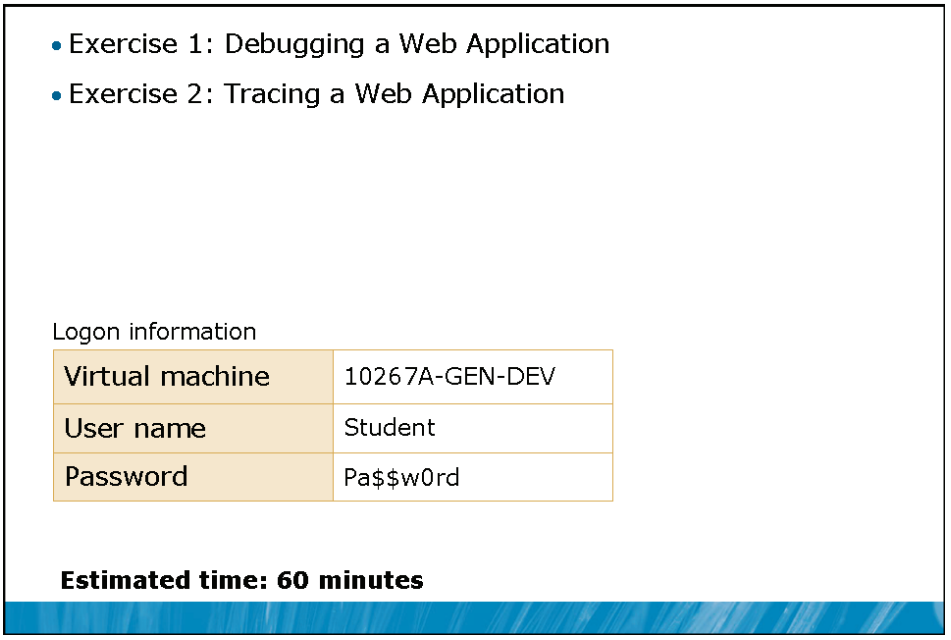
Note: The answers to the exercises are on the Course Companion CD.

Module 7

Lab Instructions: Troubleshooting Microsoft® ASP.NET Web Applications (Visual Basic)

Contents:

Lab: Troubleshooting Microsoft ASP.NET Web Applications



Note: You can perform tasks in this lab by using either the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab page. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab page.

Introduction

In this lab, you will debug an ASP.NET Web application for runtime errors by adding breakpoints and watches. In addition, you will implement tracing in the Web application to view information such as Session state, Application state, and server variables.

Objectives

After completing this lab, you will be able to:

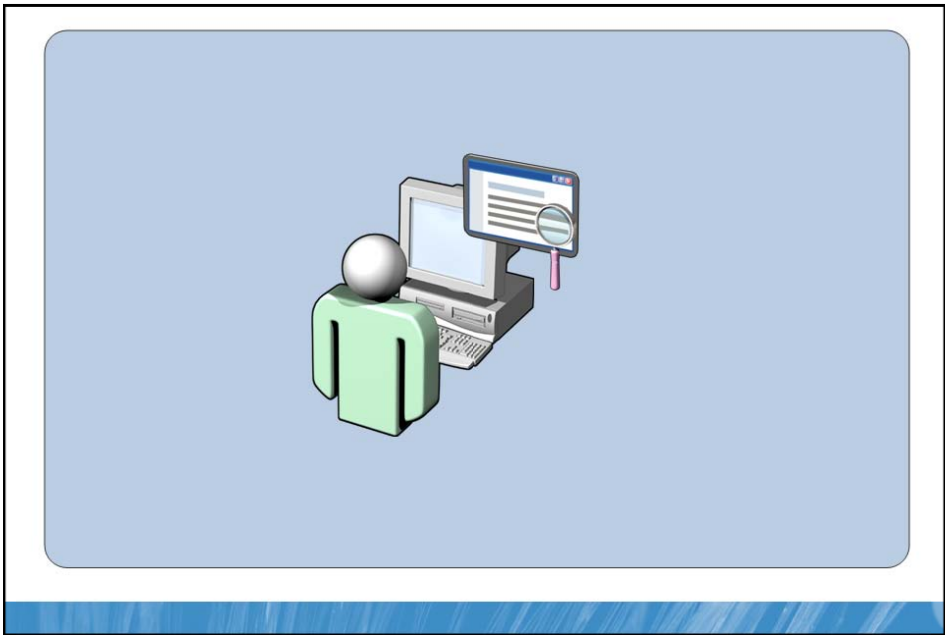
- Debug a Web application to view runtime information.
- Enable and implement tracing in a Web application.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. To maintain correct user information, you need to debug the customer management Web application for logical issues in its functionality. You need to debug the application by adding appropriate output statements, breakpoints, and watches. You also need to enable tracing by using the web.config file to identify possible errors in the code after the application's deployment. The version of the Web application that was finished in the previous module has been deployed to a staging server where debugging is not possible. However, there seems to be a performance issue when loading or rendering the InsertCustomer.aspx Web Form. You are not allowed to append trace information to each page, so you must therefore enable application-level tracing, and then test that you can view the timings for the InsertCustomer.aspx Web Form.

Section 1: Visual Basic

Exercise 1: Debugging a Web Application

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Enable debugging of the CustomerManagement Web project.
3. Add debug output statements to the user control.
4. Find and fix a bug.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M7\VB folder.

▶ Task 2: Enable debugging of the CustomerManagement Web project

- Open the **web.config** file of the **CustomerManagement** Web project.
- Set the **debug** attribute of the **compilation** element to **true**.

```
<compilation debug="true" strict="false" explicit="true"
targetFramework="4.0" />
```

- Save and close the **web.config** file.

▶ Task 3: Add debug output statements to the user control

- Open the **Customer** user control.
- Import the **System.Diagnostics** namespace.

```
Imports System.Diagnostics
```

- In the **Page_Load** event handler, send the message, "Page Postback detected in Page_Load" to the trace listeners, when the page is loaded in response to a postback.

```
Debug.WriteLine("Page Postback detected in Page_Load")
```

- In the **Page_Load** event handler, send the message, "No Page Postback detected in Page_Load" to the trace listeners, when the page is not loaded in response to a postback.

```
Debug.WriteLineIf(Not Page.IsPostBack, "No Page Postback detected
in Page_Load")
```

- At the end of the **Page_Unload** event handler, send the message, "Page has been unloaded" to the trace listeners.

```
Debug.WriteLine("Page has been unloaded")
```

- At the end of the **Click** event of the **CustomerInsertButton** control, send the message, "Customer has been inserted in CustomerInsertButton_Click" to the trace listeners.

```
Debug.WriteLine("Customer has been inserted in
CustomerInsertButton_Click");
```

- At the end of the private **populateUI** method, send the message, "UI controls have been populated" to the trace listeners.

```
Debug.WriteLine("UI controls have been populated")
```

- At the end of the private **instantiateCustomerObject** method, send the message, "Customer object has been instantiated" to the trace listeners.

```
Debug.WriteLine("Customer object has been instantiated")
```

- Save the user control code file.
- Add a default item to the **Country DropDownList** control.
  - In the Customer.ascx.vb window, right-click and then click **View Designer**.
  - In the Customer.ascx window, click the **CustomerCountryDropDownList** control.

    - Click the **Smart Tag** button, and then click **Edit Items**.
    - In the **ListItem Collection Editor** dialog box, click **Add**.
    - In the **ListItem properties** pane, in the **Text** box, type **USA**, and then click **OK**.
- Save and close the user control file.
- Run the Web application in the debug mode.
- Verify the output for the **Page_Load**, **Page_Unload**, and **instantiateCustomerObject** methods, by creating a new customer and viewing the Output pane of the Debugger variable windows.

Note: In the **Debug** pane of the Output window, notice that the output of the **Debug** statements displays. You may have to scroll up to see the statements.

- Verify the output for the **Click** event handler of the **CustomerInsertButton** control, by creating a new customer, using the following information, and then clicking the **Insert** button:
    - First Name: **Claus**
    - Last Name: **Hansen**
    - Address: **4567 Main St.**
    - Zip Code: **98052**
    - City: **Buffalo**
    - State: **NY**
    - Web Address: **http://www.cohowinery.com**
- Close Windows Internet Explorer®.

Note: In the Debug pane of the Output window, notice that the **Customer has been inserted in CustomerInsertButton_Click** message is displayed.

▶ Task 4: Find and fix a bug

- Run the **CustomerManagement** Web application to test its functionality.
- Create a new customer, by using the following information:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**

Note: Notice that the value for the **Credit Limit** box is set to **0** by default.

- Click the **Insert** button.

Note: Notice that the value for the **Credit Limit** box is set to **50**, which is incorrect.

- Close Windows Internet Explorer®.
- Add a breakpoint in the **Page_Load** event handler method in the line of code that calls the **instantiateCustomerObject** method.

```
instantiateCustomerObject();
```

- Run the Web application in debug mode.
- In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
- Step into the **instantiateCustomerObject** method.
- Step over the first line of code and then check for postback in the **instantiateCustomerObject** method.
- Add a watch to the **Text** property of the **CustomerCreditLimitTextBox** control.
- Continue to debug the Web application.

- Create a new customer, by using the following information, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**

Note: Notice that in the Watch 1 window, the value of the **CustomerCreditLimitTextBox** Text property is set to **0**.

- Step over the call to the **instantiateCustomerObject** method.
- Step over the call to the **populateUI** method.

Note: In the Watch1 window, notice that the value for the **CustomerCreditLimitTextBox** is changed to **50**.

- Stop debugging the Web project.
- Examine the code in the **populateUI** method that assigns a value to the **CustomerCreditLimitTextBox**.**Text** property.

Note: Notice that the value of **50** for the **CreditLimit** property is not assigned here.

- Re-run the Web application in debug mode to examine the **Customer** class.
- Create a new customer, and ignore the first breakpoint, by continuing program execution.

- Create a new customer, by using the following information, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code**: 98052**
  - City: Buffalo
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**
- Step into the **instantiateCustomerObject** method.
- Step over the first line of code, and then the check for postback.
- Step into the method that instantiates the customer object.

Note: Stepping into the instantiation of the customer object action may take some time.

- Step over each single line of code in the Customer.vb code window, until the following line of code in the constructor that initializes the **CreditLimit** property is reached.

```
Me.CreditLimit = creditLimit
```

- Step through the assignment of the passed value to the **CreditLimit** property.
- Locate the step that adds the extra **50** to the private **customerCreditLimit** member **Credit Limit** box.

```
Me.customerCreditLimit = value + 50
```

- View the functions and procedure calls that are currently on the stack, by viewing the Call Stack window.
- Stop debugging the Web project.
- Remove the extra value from the **customerCreditLimit** property.

```
Me.customerCreditLimit = value
```

- Save and close the class file.

- Run the Web application to verify the **Credit Limit** value.
- Create a new customer, by using the following information:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**
- Click the **Insert** button.

Note: Notice that the value for the **Credit Limit** box is still **0**.

- Close Windows Internet Explorer®.

Results: After completing this exercise, you will have enabled debugging for the CustomerManagement Web project, added debug output statements to the user control, and fixed a bug in the CustomerManagement Web application functionality.

Exercise 2: Tracing a Web Application

The main tasks for this exercise are as follows:

1. Enable application-level tracing of the CustomerManagement Web project.
2. Implement application tracing.

▶ Task 1: Enable application-level tracing of the CustomerManagement Web project

- Add a trace element to the **web.config** file as the first element within the **system.web**, and set the value of the **enabled** attribute to **true**.

```
<trace enabled="true" />
```

- Save and close the **web.config** file.

▶ Task 2: Implement application tracing

- Run the Web application, and view the trace details.
- Create a new customer, by using the following information, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**
- View the trace details for the application, by using the **http://localhost:1111/CustomerManagement/trace.axd** URL in the browser.

- View the details for the **InsertCustomer.aspx** Web page.
  - In the http://localhost:1111/CustomerManagement/trace.axd - Windows Internet Explorer window, click **View Details** of **/InsertCustomer.aspx** corresponding to the **Verb**, **GET**.

Note: Scroll down to see all of the information output, including the control tree, session and application state, form and querystring collection, and server variables of the **InsertCustomer.aspx** Web page. Verify that you can see the Trace Information section, which gives you information about how much time is spent when loading, rendering, and unloading the Web Form.

- Close Windows Internet Explorer®.

▶ Task 3: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

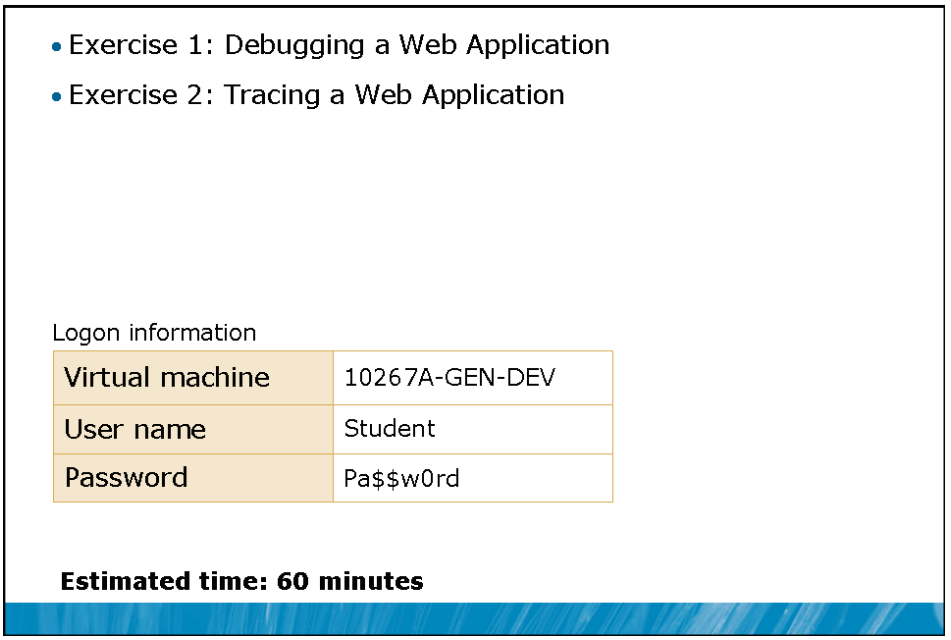
Results: After completing this exercise, you will have enabled and implemented application-level tracing for the CustomerManagement Web application.

Module 7

Lab Instructions: Troubleshooting Microsoft® ASP.NET Web Applications (Visual C#)

Contents:

Lab: Troubleshooting Microsoft ASP.NET Web Applications



Note: You can perform tasks in this lab by using either the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab page. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab page.

Introduction

In this lab, you will debug an ASP.NET Web application for runtime errors by adding breakpoints and watches. In addition, you will implement tracing in the Web application to view information such as Session state, Application state, and server variables.

Objectives

After completing this lab, you will be able to:

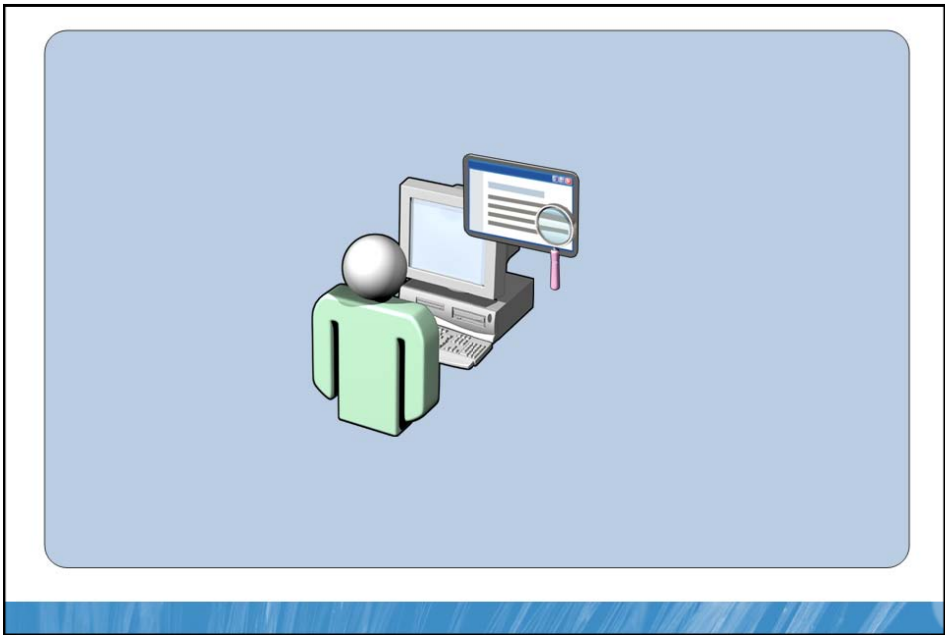
- Debug a Web application to view runtime information.
- Enable and implement tracing in a Web application.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. To maintain correct user information, you need to debug the customer management Web application for logical issues in its functionality. You need to debug the application by adding appropriate output statements, breakpoints, and watches. You also need to enable tracing by using the web.config file to identify possible errors in the code after the application's deployment. The version of the Web application that was finished in the previous module has been deployed to a staging server where debugging is not possible. However, there seems to be a performance issue when loading or rendering the InsertCustomer.aspx Web Form. You are not allowed to append trace information to each page, so you must therefore enable application-level tracing, and then test that you can view the timings for the InsertCustomer.aspx Web Form.

Section 2: Visual C#

Exercise 1: Debugging a Web Application

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Enable debugging of the CustomerManagement Web project.
3. Add debug output statements to the user control.
4. Find and fix the bug.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M7\CS folder.

▶ Task 2: Enable debugging of the CustomerManagement Web project

- Open the **web.config** file of the **CustomerManagement** Web project.
- Set the **debug** attribute of the **compilation** element to **true**.

```
<compilation debug="true" targetFramework="4.0">
```

- Save and close the **web.config** file.

▶ Task 3: Add debug output statements to the user control

- Open the **Customer** user control.
- Import the **System.Diagnostics** namespace to the user control.

```
using System.Diagnostics;
```

- In the **Page_Load** event handler, send the message, "Page Postback detected in Page_Load" to the trace listeners, when the page is loaded in response to a postback.

```
Debug.WriteLine("Page Postback detected in Page_Load");
```

- In the **Page_Load** event handler, send the message, "No Page Postback detected in Page_Load" to the trace listeners, when the page is not loaded in response to a postback.

```
Debug.WriteLineIf(Not Page.IsPostBack, "No Page Postback detected
in Page_Load");
```

- At the end of the **Page_Unload** event handler, send the message, "Page has been unloaded" to the trace listeners.

```
Debug.WriteLine("Page has been unloaded");
```

- At the end of the **Click** event of the **CustomerInsertButton** control, send the message, "Customer has been inserted in CustomerInsertButton_Click" to the trace listeners.

```
Debug.WriteLine("Customer has been inserted in
CustomerInsertButton_Click");
```

- At the end of the private **populateUI** method, send the message, "UI controls have been populated" to the trace listeners.

```
Debug.WriteLine("UI controls have been populated");
```

- At the end of the private **instantiateCustomerObject** method, send the message, "Customer object has been instantiated" to the trace listeners.

```
Debug.WriteLine("Customer object has been instantiated");
```

- Save the user control code file.
- Add a default item to the **Country DropDownList** control.
  - In the Customer.ascx.cs window, right-click and then click **View Designer**.
  - In the Customer.ascx window, click the **CustomerCountryDropDownList** control.
  - Click the **Smart Tag** button, and then click **Edit Items**.

    - In the **ListItem Collection Editor** dialog box, click **Add**.
    - In the **ListItem properties** pane, in the **Text** box, type **USA**, and then click **OK**.
- Save and close the user control file.
- Run the Web application in the debug mode.
- Verify the output for the **Page_Load**, **Page_Unload**, and **instantiateCustomerObject** methods, by creating a new customer and viewing the Output pane of the Debugger variable windows.

Note: In the Debug pane of the Output window, notice that the output of the **Debug** statements display. You may have to scroll up to see the statements.

- Verify the output for the **Click** event handler of the **CustomerInsertButton** control, by creating a new customer using the following information, and then Click the **Insert** button:
    - First Name: **Claus**
    - Last Name: **Hansen**
    - Address: **4567 Main St.**
    - Zip Code: **98052**
    - City: **Buffalo**
    - State: **NY**
    - Web Address: **http://www.cohowinery.com**
- Close Windows Internet Explorer®.

Note: In the Debug pane of the Output window, notice that the **Customer has been inserted in CustomerInsertButton_Click** message is displayed.

▶ Task 4: Find and fix a bug

- Run the **CustomerManagement** Web application to test its functionality.
- Create a new customer, by using the following information:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**

Note: Notice that the value for the **Credit Limit** box is set to **0** by default.

- Click the **Insert** button.

Note: Notice that the value for the **Credit Limit** box is set to **50**, which is incorrect.

- Close Windows Internet Explorer®.
- Add a breakpoint in the **Page_Load** event handler method in the line of code that calls the **instantiateCustomerObject** method.

```
instantiateCustomerObject();
```

- Run the Web application in debug mode.
- Step into the **instantiateCustomerObject** method.
- Step over the first line of code and the check for postback in the **instantiateCustomerObject** method.
- Add a watch to the **Text** property of the **CustomerCreditLimitTextBox** control.
- Continue to debug the Web application.

- Create a new customer, by using the following information, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**

Note: Notice that in the Watch 1 window, the value of the **CustomerCreditLimitTextBox** Text property is set to **0**.

- Step over the call to the **instantiateCustomerObject** method.
- Step over the call to the **populateUI** method.

Note: In the Watch1 window, notice that the value for the **CustomerCreditLimitTextBox** is changed to **50**.

- Stop debugging the Web project.
- Examine the code in the **populateUI** method that assigns a value to the **CustomerCreditLimitTextBox**.**Text** property.

Note: Notice that the value of **50** for the **CreditLimit** property is not assigned here.

- Re-run the Web application in debug mode to examine the **Customer** class.

- Create a new customer, and ignore the first breakpoint, by continuing program execution.Create a new customer, by using the following information, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**
- Step into the **instantiateCustomerObject** method.
- Step over the first line of code and then the check for postback.
- Step into the method that instantiates the customer object.

Note: Stepping into the instantiation of the customer object action may take some time.

- Step over each single line of code in the Customer.cs code window, until the following line of code in the constructor that initializes the **CreditLimit** property is reached.

```
this.CreditLimit = creditLimit;
```

- Step through the assignment of the passed value to the **CreditLimit** property.
- Locate the step that adds the extra **50** to the private **customerCreditLimit** member **Credit Limit** box.

```
this.customerCreditLimit = value + 50;
```

- View the functions and procedure calls that are currently on the stack, by viewing the **Call Stack** window.
- Stop debugging the Web project.
- Remove the extra value from the **customerCreditLimit** property.

```
this.customerCreditLimit = value;
```

- Save and close the class file.
- Run the Web application to verify the **Credit Limit** value.
- Create a new customer, by using the following information, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**

Note: Notice that the value for the **Credit Limit** box is still **0**.

- Close Windows Internet Explorer®.

Results: After completing this exercise, you will have enabled debugging for the CustomerManagement Web project, added debug output statements to the user control, and fixed a bug in the CustomerManagement Web application functionality.

Exercise 2: Tracing a Web Application

The main tasks for this exercise are as follows:

1. Enable application-level tracing of the CustomerManagement Web project.
2. Implement application tracing.

▶ Task 1: Enable application-level tracing of the CustomerManagement Web project

- Add a trace element to the **web.config** file as the first element within the **system.web**, and set the value of the **enabled** attribute to **true**.

```
<trace enabled="true" />
```

- Save and close the **web.config** file.

▶ Task 2: Implement application tracing

- Run the Web application, and view the trace details.
- Create a new customer, by using the following information, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**
- View the trace details for the application, by using the **http://localhost:1110/CustomerManagement/trace.axd** URL in the browser. View the details for the **InsertCustomer.aspx** Web page.
  - In the http://localhost:1110/CustomerManagement/trace.axd - Windows Internet Explorer window, click **View Details** of **/InsertCustomer.aspx** corresponding to the **Verb**, **GET**.

Note: Scroll down to see all of the information output, including the control tree, session and application state, form and querystring collection, and server variables of the **InsertCustomer.aspx** Web page. Verify that you can see the Trace Information section, which gives you information about how much time is spent when loading, rendering, and unloading the Web Form.

- Close Windows Internet Explorer®.

Results: After completing this exercise, you will have enabled and implemented application-level tracing for the CustomerManagement Web application.

▶ Task 3: Turn off the virtual machine and revert the changes

After you complete the lab, you must turn off the 10267A-GEN-DEV virtual machine and revert the changes.

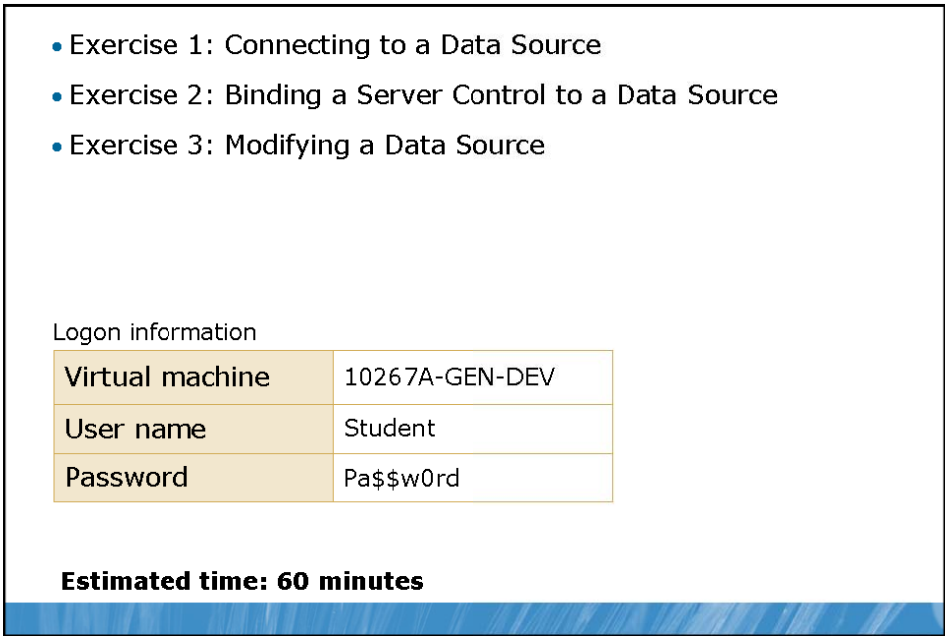
- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

Module 8

Lab Instructions: Managing Data in a Microsoft® ASP.NET 4.0 Web Application (Visual Basic)

Contents:

Lab: Managing Data in an ASP.NET 4.0 Web Application



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will connect the ASP.NET Web application to a SQL Server database by using the server control, and then bind the user control in the application to a data source. In addition, you will modify the source database, and verify the changes that are made to the database.

Objectives

After completing this lab, you will be able to:

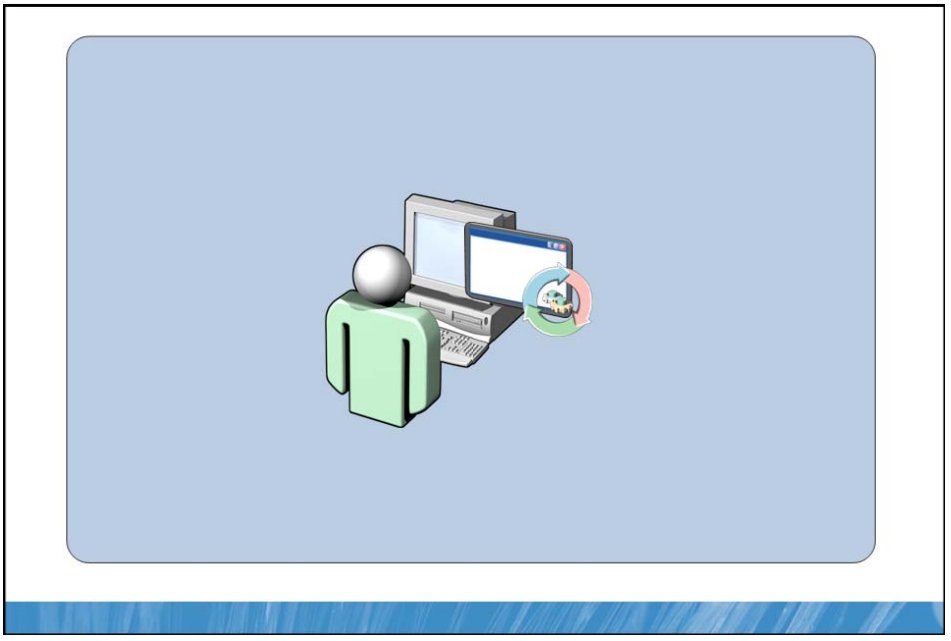
- Connect to a SQL Server database by using the **SqlDataSource** control.
- Bind a user control to the data source.
- Modify the source database, and verify the changes.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **NYC-CL1** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. You are responsible for managing the user information in your organization. To perform this task, you need to add functionality for simple data access tasks to retrieve data in a short time without using a lot of code. To minimize the chances of creating error-prone code when accessing the data in the database, you need to add and configure a data source control to the user control. In addition, you need to add and bind a server control to the user control for displaying specific data from the database.

Section 1: Visual Basic

Exercise 1: Connecting to a Data Source

The main tasks in this exercise are as follows:

1. Open an existing ASP.NET Web project.
2. Add a SQL Server 2008 Express Database.
3. Add a data source control to the user control.
4. Configure a data source control.

▶ Task 1: Open an existing ASP.NET Web project

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M8\VB folder.

▶ Task 2: Add a SQL Server 2008 Express Database

- Add a new database folder named **App_Data**, and an existing database to the database folder, D:\LabFiles\Starter\M8\CustomerManagement.mdf.

▶ Task 3: Add a data control source to the user control

- Open the **Customer** user control in Design view.
- Add a **SqlDataSource** control to the user control to connect to an SQL Server database.
- Rename the **SqlDataSource** control as **CountriesSqlDataSource**.
- Save the **Customer** user control.

▶ Task 4: Configure a data source control

- In Design view, with the **CountriesSqlDataSource** control selected, display the **Smart Tag**.
- Open the Configure Data Source Wizard, connect to the **CustomerManagement.mdf** database, and then create a new connection string named **CustomerManagementConnectionString**.
- Configure the SELECT statement to include the **ID** and **Name** columns from the **Countries** table.
- Test the query, and check whether you get the correct data from the Countries table.

Note: Ensure that the returned rows include the values for the ID and Name columns for various countries.

- Save the **Customer** user control.

Exercise 2: Binding a Server Control to a Data Source

The main tasks in this exercise are as follows:

1. Bind the **DropDownList** control to a data source.
2. Pass the values to the **Customer** object.

▶ Task 1: Bind the DropDownList control to a data source

- Open the **Customer** user control in Source view.
- Locate the markup for the **CustomerCountryDropDownList** control.
- Remove the static **ListItem** element from the **CustomerCountryDropDownList** control.

```
<asp:ListItem>USA</asp:ListItem>
```

- Bind the **CustomerCountryDropDownList** control to the **CountriesSqlDataSource** control by using the **DataSourceID** attribute.

```
DataSourceID="CountriesSqlDataSource"
```

- Set the value field of the **CustomerCountryDropDownList** control to the **ID** column of the database by using the **DataValueField** attribute.

```
DataValueField="ID"
```

- Set the text field of the **CustomerCountryDropDownList** control to the **Name** column of the database by using the **DataTextField** attribute.

```
DataTextField="Name"
```

- Build the user control, and fix any errors.

Note: Notice the Build succeeded message in the **Build** pane of the Output window.

▶ Task 2: Pass the values to the Customer object

- Open the **Customer** user control in the Code view.
- Locate the code for the private **instantiateCustomerObject** method and pass the selected value of the **CustomerCountryDropDownList** control to the **Customer** class constructor by using the **SelectedValue** property wrapped in a **new Guid** object.

```
' Instantiate new Customer object with user input
currentCustomer = New CustomerManagementEntities.Customer(
    Nothing, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
    CustomerAddressTextBox.Text, CustomerZipCodeTextBox.Text,
CustomerCityTextBox.Text,
    CustomerStateTextBox.Text, New
Guid(CustomerCountryDropDownList.SelectedValue),
CustomerPhoneTextBox.Text,
    CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text,
CustomerNewsSubscriberCheckBox.Checked,
    Integer.Parse(CustomerCreditLimitTextBox.Text), DateTime.Now,
"", Nothing, "")
```

- Build the user control, and fix any errors.

Exercise 3: Modifying a Data Source

The main tasks for this exercise are as follows:

1. Create the **Customers** Web Form.
2. Add the **SqlDataSource** control to the Web Form.
3. Configure the **SqlDataSource** control.
4. Add the **ListView** control to the Web Form.
5. Add code to manually save a customer to a data source.
6. Update sitemap to enable view all customers.
7. Create a customer and verify the Data Source.

▶ Task 1: Create the Customers Web Form

- Create the **Customers** Web Form based on the master page, Site.master.
- Open the **Customers** Web Form in the Design view.

▶ Task 2: Add the SqlDataSource control to the Web Form

- Add the **SqlDataSource** control to the **Customers.aspx** Web Form.
- Rename the **SqlDataSource** control as **CustomersSqlDataSource**.
- Save the **Customers** Web Form.

▶ Task 3: Configure the SqlDataSource control

- In Design view, with the **CustomersSqlDataSource** control selected, display the Smart Tag.
- Open the Configure Data Source Wizard.
- Select the **CustomerManagementConnectionString** connection string for the **CustomersSqlDataSource** control.
- Configure the SELECT statement to include all the columns from the **Customers** table, and ensure that it is possible to manipulate the data in the data source, and use optimistic concurrency.
- Save the **Customers** Web Form.

▶ Task 4: Add the ListView control to the Web Form

- Add the **ListView** control to the **Customers** Web Form.
- Rename the **ListView** control as **CustomersListView**.
- Bind the **CustomersListView** control to the **CustomersSqlDataSource** control by using the Smart Tag.
- Enable paging in the **CustomersListView** control.
- Save the **Customers** Web Form.
- Build the user control, and fix any errors.
- View the **Customers** Web Form in the browser.
- Close Windows® Internet Explorer®.

▶ Task 5: Add code to manually save a customer to a data source

- Open the **Customer** user control in the Code view.
- Import the namespace used for accessing a SQL Server 2008 database, **System.Data.SqlClient**.
- Import the namespace used with the disconnected ADO.NET layer, **System.Data**.
- Import the namespace for reading the connection string from the web.config file, **System.Configuration**.
- Locate the **CustomerInsertButton_Click** event handler.

```
Protected Sub CustomerInsertButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CustomerInsertButton.Click
    ' Did page validation succeed?
    If Not Page.IsValid Then
        Exit Sub
    End If

    ' Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name
    ' Add the user credit limit
    currentCustomer.CreditLimit = 50000
End Sub
```

- Remove the assignment of the value **50000** to the **CreditLimit** property.

```
' Add the user credit limit
currentCustomer.CreditLimit = 50000
```

- Append the following code to the **CustomerInsertButton_Click** event handler, by using a code snippet named **ADO.NET Insert Customer**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder.

```
' Create and instantiate connection
Using customerManagementConnection As New SqlConnection()
    ' Initialize connection string from web.config
    customerManagementConnection.ConnectionString =

ConfigurationManager.ConnectionStrings("CustomerManagementConnecti
onString").ConnectionString

    ' Open connection
    customerManagementConnection.Open()
    ' Declare and instantiate data adapter
    Dim customerManagementDataAdapter As New SqlDataAdapter()

    ' Declare and instantiate command objects
    Dim selectCommand As New SqlCommand("SELECT * FROM Customers",
        customerManagementConnection)
    Dim insertCommand As New SqlCommand(
        "INSERT INTO Customers (FirstName, LastName, Address,
ZipCode, City, State, CountryID, Phone, EmailAddress, " &
        "Url, CreditLimit, NewsSubscriber, CreatedDate, CreatedBy)
VALUES(@FirstName, @LastName, @Address, @ZipCode, @City, @State, "
&
        "@CountryID, @Phone, @EmailAddress, @WebAddress,
@CreditLimit, @NewsSubscriber, @CreatedDate, @CreatedBy)",
customerManagementConnection)

    ' Assign command objects
    customerManagementDataAdapter.SelectCommand = selectCommand
    customerManagementDataAdapter.InsertCommand = insertCommand

    ' Declare and instantiate parameter objects
    Dim insertFirstNameParameter As New SqlParameter("@FirstName",
SqlDbType.NVarChar, 50, "FirstName")
    Dim insertLastNameParameter As New SqlParameter("@LastName",
SqlDbType.NVarChar, 30, "LastName")
```

(*Code continued on the following page.*)

```
    Dim insertAddressParameter As New SqlParameter("@Address",
SqlDbType.NVarChar, 50, "Address")
    Dim insertZipCodeParameter As New SqlParameter("@ZipCode",
SqlDbType.NVarChar, 10, "ZipCode")
    Dim insertCityParameter As New SqlParameter("@City",
SqlDbType.NVarChar, 30, "City")
    Dim insertStateParameter As New SqlParameter("@State",
SqlDbType.NVarChar, 30, "State")
    Dim insertCountryIDParameter As New SqlParameter("@CountryID",
SqlDbType.UniqueIdentifier, 0, "CountryID")
    Dim insertPhoneParameter As New SqlParameter("@Phone",
SqlDbType.VarChar, 30, "Phone")
    Dim insertEmailAddressParameter As New
SqlParameter("@EmailAddress", SqlDbType.NVarChar, 50,
"EmailAddress")
    Dim insertWebAddressParameter As New
SqlParameter("@WebAddress", SqlDbType.NVarChar, 80, "Url")
    Dim insertCreditLimitParameter As New
SqlParameter("@CreditLimit", SqlDbType.Int, 0, "CreditLimit")
    Dim insertNewsSubscriberParameter As New
SqlParameter("@NewsSubscriber", SqlDbType.Bit, 0,
"NewsSubscriber")
    Dim insertCreatedDateParameter As New
SqlParameter("@CreatedDate", SqlDbType.SmallDateTime, 0,
"CreatedDate")
    Dim insertCreatedByParameter As New SqlParameter("@CreatedBy",
SqlDbType.VarChar, 15, "CreatedBy")

    ' Assign parameters to command object
    insertCommand.Parameters.Add(insertFirstNameParameter)
    insertCommand.Parameters.Add(insertLastNameParameter)
    insertCommand.Parameters.Add(insertAddressParameter)
    insertCommand.Parameters.Add(insertZipCodeParameter)
    insertCommand.Parameters.Add(insertCityParameter)
    insertCommand.Parameters.Add(insertStateParameter)
    insertCommand.Parameters.Add(insertCountryIDParameter)
    insertCommand.Parameters.Add(insertPhoneParameter)
    insertCommand.Parameters.Add(insertEmailAddressParameter)
    insertCommand.Parameters.Add(insertWebAddressParameter)
    insertCommand.Parameters.Add(insertCreditLimitParameter)
    insertCommand.Parameters.Add(insertNewsSubscriberParameter)
    insertCommand.Parameters.Add(insertCreatedDateParameter)
    insertCommand.Parameters.Add(insertCreatedByParameter)
```

(*Code continued on the following page.*)

```
    ' Declare and instantiate dataset
    Dim customerManagementDataSet As New
DataSet("CustomerManagementDataSet")
    ' Apply the full schema from the data source

customerManagementDataAdapter.FillSchema(customerManagementDataSet
, SchemaType.Source, "Customers")
    customerManagementDataAdapter.MissingSchemaAction =
MissingSchemaAction.AddWithKey
    customerManagementDataAdapter.MissingMappingAction =
MissingMappingAction.Passthrough
    ' Populate Customers DataTable
    customerManagementDataAdapter.Fill(customerManagementDataSet,
"Customers")

    ' Create new row locally
    Dim newCustomerDataRow As DataRow =
customerManagementDataSet.Tables("Customers").NewRow()
    newCustomerDataRow("ID") = Guid.NewGuid()
    newCustomerDataRow("FirstName") = currentCustomer.FirstName
    newCustomerDataRow("LastName") = currentCustomer.LastName
    newCustomerDataRow("Address") = currentCustomer.Address
    newCustomerDataRow("ZipCode") = currentCustomer.ZipCode
    newCustomerDataRow("City") = currentCustomer.City
    newCustomerDataRow("State") = currentCustomer.State
    newCustomerDataRow("CountryID") = currentCustomer.CountryID
    newCustomerDataRow("Phone") = currentCustomer.Phone
    newCustomerDataRow("EmailAddress") =
currentCustomer.EmailAddress
    newCustomerDataRow("Url") = currentCustomer.EmailAddress
    newCustomerDataRow("CreditLimit") =
currentCustomer.CreditLimit
    newCustomerDataRow("NewsSubscriber") =
currentCustomer.NewsSubscriber
    newCustomerDataRow("CreatedDate") =
currentCustomer.CreatedDate
    newCustomerDataRow("CreatedBy") = currentCustomer.CreatedBy

    ' Insert new row locally

customerManagementDataSet.Tables("Customers").Rows.Add(newCustomer
DataRow)
```

(*Code continued on the following page.*)

```
    ' Update data source
    If
customerManagementDataAdapter.Update(customerManagementDataSet,
"Customers") = 1 Then
        ' Instantiate new Customer object
        currentCustomer = New
CustomerManagementEntities.Customer()
        ' Reload page to refresh with "blank" input controls
        Response.Redirect("~/InsertCustomer.aspx")
    End If
End Using
```

- Save the Customer code-behind file.

▶ Task 6: Update sitemap to enable view all customers

- Open the **web.sitemap** file.
- Append the new **siteMapNode** element to the **Customers** siteMapNode.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers.aspx" />
```

- Save and close the **web.sitemap** file.

▶ Task 7: Create a customer and verify the Data Source

- Build the **CustomerManagement** solution.
- Run the **CustomerManagement** Web application.
- Open the **InsertCustomer** Web Form, and click **New** on the **Customers** menu.
- Create a new customer by using the following information, and then click the **Insert** button:
  - First Name: **Kim**
  - Last Name: **Abercrombie**
  - Address: **9876 Maine Road**
  - Zip Code: **M24NG**
  - City: **Manchester**

- Country: **Great Britain**
- Phone: **0161-123 555**
- Email Address: **kim@litwareinc.com**
- Web Address: **http://www.litwareinc.com**
- Credit Limit: **50000**
- News Subscriber: **Yes**

- Check that the new customer has been added in the data source by using the new **Customers** Web Form.
- Close Internet Explorer.

▶ Task 8: Turn off the virtual machine and revert the changes

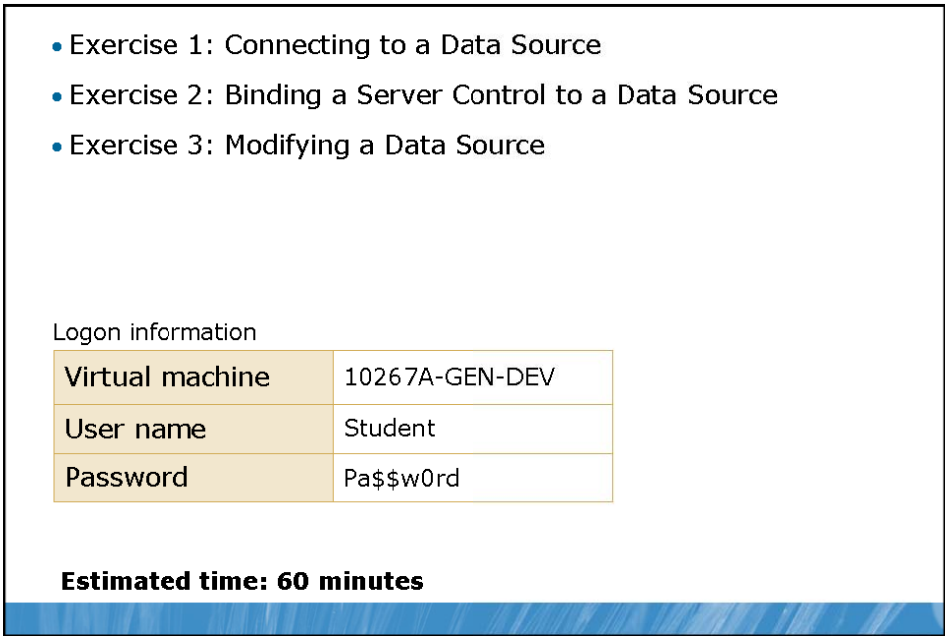
- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

Module 8

Lab Instructions: Managing Data in a Microsoft® ASP.NET 4.0 Web Application (Visual C#)

Contents:

Lab: Managing Data in an ASP.NET 4.0 Web Application



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will connect the ASP.NET Web application to a SQL Server database by using the server control, and then bind the user control in the application to a data source. In addition, you will modify the source database, and verify the changes that are made to the database.

Objectives

After completing this lab, you will be able to:

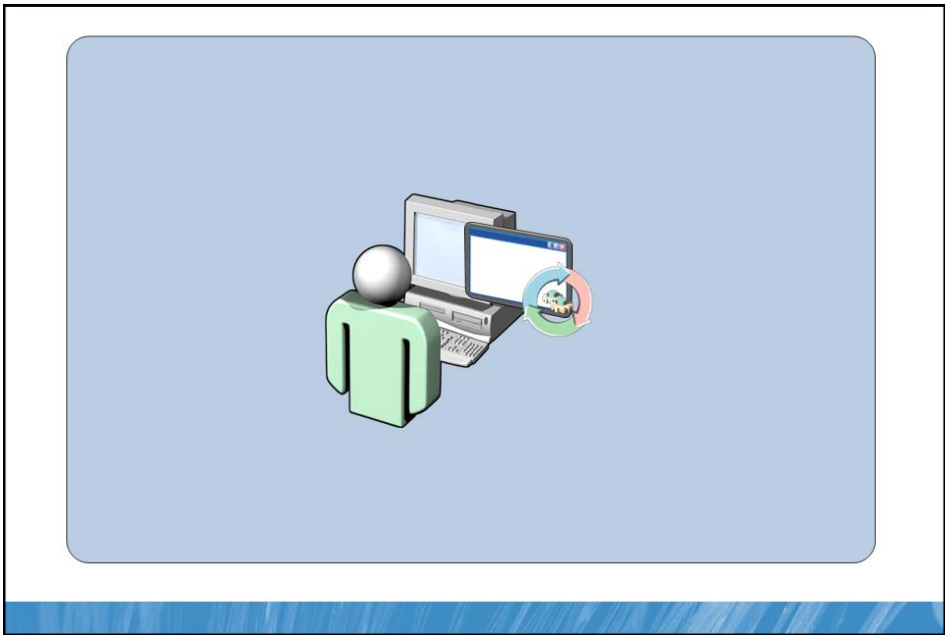
- Connect to a SQL Server database by using the **SqlDataSource** control.
- Bind a user control to the data source.
- Modify the source database, and verify the changes.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **NYC-CL1** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. You are responsible for managing the user information in your organization. To perform this task, you need to add functionality for simple data access tasks to retrieve data in a short time without using a lot of code. To minimize the chances of creating error-prone code when accessing the data in the database, you need to add and configure a data source control to the user control. In addition, you need to add and bind a server control to the user control for displaying specific data from the database.

Section 2: Visual C#

Exercise 1: Connecting to a Data Source

The main tasks in this exercise are as follows:

1. Open an existing ASP.NET Web project.
2. Add a SQL Server 2008 Express Database.
3. Add a data source control to the user control.
4. Configure a data source control.

▶ Task 1: Open an existing ASP.NET Web project

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M8\CS folder.

▶ Task 2: Add a SQL Server 2008 Express Database

- Add a new database folder named **App_Data**, and an existing database to the database folder, D:\LabFiles\Starter\M8\CustomerManagement.mdf.

▶ Task 3: Add a data source control to the user control

- Open the **Customer** user control in Design view.
- Add a **SqlDataSource** control to the user control to connect to a SQL Server database.
- Rename the **SqlDataSource** control as **CountriesSqlDataSource**.
- Save the **Customer** user control.

▶ Task 4: Configure a data source control

- In Design view, with the **CountriesSqlDataSource** control selected, display the Smart Tag.
- Open the Configure Data Source Wizard, connect to the **CustomerManagement.mdf** database, and then create a new connection string named **CustomerManagementConnectionString**.
- Configure the SELECT statement to include the **ID** and **Name** columns from the **Countries** table.
- Test the query, and check whether you get the correct data from the Countries table.

Note: Ensure that the returned rows include the values for the ID and Name columns for various countries.

- Save the **Customer** user control.

Exercise 2: Binding a Server Control to a Data Source

The main tasks in this exercise are as follows:

1. Bind the **DropDownList** control to the data source.
2. Pass the values to the **Customer** object.

▶ Task 1: Bind the DropDownList control to a data source

- Open the **Customer** user control in Source view.
- Locate the markup for the **CustomerCountryDropDownList** control.
- Remove the static **ListItem** element from the **CustomerCountryDropDownList** control.

```
<asp:ListItem>USA</asp:ListItem>
```

- Bind the **CustomerCountryDropDownList** control to the **CountriesSqlDataSource** control by using the **DataSourceID** attribute.

```
DataSourceID="CountriesSqlDataSource"
```

- Set the value field of the **CustomerCountryDropDownList** control to the **ID** column of the database by using the **DataValueField** attribute.

```
DataValueField="ID"
```

- Set the text field of the **CustomerCountryDropDownList** control to the **Name** column of the database by using the **DataTextField** attribute.

```
DataTextField="Name"
```

- Build the user control, and fix any errors.

▶ Task 2: Pass the values to the Customer object

- Open the **Customer** user control in the Code view.
- Locate the code for the private **instantiateCustomerObject** method, and pass the selected value of the **CustomerCountryDropDownList** control to the **Customer** class constructor by using the **SelectedValue** property wrapped in a **new Guid** object.

```
// Instantiate new Customer object with user input
currentCustomer = new CustomerManagementEntities.Customer(
    null, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
    CustomerAddressTextBox.Text, CustomerZipCodeTextBox.Text,
    CustomerCityTextBox.Text, CustomerStateTextBox.Text,
    new Guid(CustomerCountryDropDownList.SelectedValue),
CustomerPhoneTextBox.Text,
    CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text,
    int.Parse(CustomerCreditLimitTextBox.Text),
CustomerNewsSubscriberCheckBox.Checked,
    DateTime.Now, "", null, "");
```

- Build the user control, and fix any errors.

Exercise 3: Modifying a Data Source

The main tasks in this exercise are as follows:

1. Create the **Customers** Web Form.
2. Add the **SqlDataSource** control to the Web Form.
3. Configure the **SqlDataSource** control.
4. Add the **ListView** control to the Web Form.
5. Add code to manually save a customer to a data source.
6. Update sitemap to enable view all customers.
7. Create a customer and verify the Data Source.

▶ Task 1: Create the Customers Web Form

- Create the **Customers** Web Form, based on the Site.master master page.
- Open the **Customers** Web Form in the Design view.

▶ Task 2: Add the SqlDataSource control to the Web Form

- Add the **SqlDataSource** control to the **Customers.aspx** Web Form.
- Rename the **SqlDataSource** control as **CustomersSqlDataSource**.
- Save the **Customers** Web Form.

▶ Task 3: Configure the SqlDataSource control

- In Design view, with the **CustomersSqlDataSource** control selected, display the Smart Tag.
- Open the Configure Data Source Wizard.
- Select the **CustomerManagementConnectionString** connection string for the **CustomersSqlDataSource** control.
- Configure the SELECT statement to include all the columns from the **Customers** table, and ensure that it is possible to manipulate the data in the data source, and use optimistic concurrency.
- Save the **Customers** Web Form.

▶ Task 4: Add the ListView control to the Web Form

- Add the **ListView** control to the **Customers** Web Form.
- Rename the **ListView** control as **CustomersListView**.
- Bind the **CustomersListView** control to the **CustomersSqlDataSource** control by using the Smart Tag.
- Enable paging in the **CustomersListView** control.
- Save the **Customers** Web Form.
- Build the user control, and fix any errors.
- View the **Customers** Web Form in the browser.
- Close Internet Explorer.

▶ Task 5: Add code to manually save a customer to a data source

- Open the **Customer** user control in the Code view.
- Import the namespace used for accessing a SQL Server 2008 database, **System.Data.SqlClient**.
- Import the namespace used with the disconnected ADO.NET layer, **System.Data**.
- Import the namespace for reading the connection string from the web.config file, **System.Configuration**.
- Locate the **CustomerInsertButton_Click** event handler.

```
protected void CustomerInsertButton_Click(object sender, EventArgs
e)
{
    // Did page validation succeed?
    if (!Page.IsValid)
        return;

    // Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name;
    // Add the user credit limit
    currentCustomer.CreditLimit = 50000;
}
```

- Remove the assignment of the value **50000** to the **CreditLimit** property.

```
// Add the user credit limit
currentCustomer.CreditLimit = 50000;
```

- Append the following code to the **CustomerInsertButton_Click** event handler, by using a code snippet named **ADO.NET Insert Customer**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder.

```
// Create and instantiate connection
using (SqlConnection customerManagementConnection = new
SqlConnection())
{
    // Initialize connection string from web.config
    customerManagementConnection.ConnectionString =

ConfigurationManager.ConnectionStrings["CustomerManagementConnecti
onString"].ConnectionString;

    // Open connection
    customerManagementConnection.Open();
    // Declare and instantiate data adapter
    SqlDataAdapter customerManagementDataAdapter = new
SqlDataAdapter();

    // Declare and instantiate command objects
    SqlCommand selectCommand = new SqlCommand("SELECT * FROM
Customers",
        customerManagementConnection);
    SqlCommand insertCommand = new SqlCommand(
        "INSERT INTO Customers (FirstName, LastName, Address,
ZipCode, City, State, CountryID, Phone, EmailAddress, " +
        "Url, CreditLimit, NewsSubscriber, CreatedDate, CreatedBy)
VALUES(@FirstName, @LastName, @Address, @ZipCode, @City, @State, "
+
        "@CountryID, @Phone, @EmailAddress, @WebAddress,
@CreditLimit, @NewsSubscriber, @CreatedDate, @CreatedBy)",
        customerManagementConnection);

    // Assign command objects
    customerManagementDataAdapter.SelectCommand = selectCommand;
    customerManagementDataAdapter.InsertCommand = insertCommand;
```

(*Code continued on the following page.*)

```
    // Declare and instantiate parameter objects
    SqlParameter insertFirstNameParameter = new
SqlParameter("@FirstName", SqlDbType.NVarChar, 50, "FirstName");
    SqlParameter insertLastNameParameter = new
SqlParameter("@LastName", SqlDbType.NVarChar, 30, "LastName");
    SqlParameter insertAddressParameter = new
SqlParameter("@Address", SqlDbType.NVarChar, 50, "Address");
    SqlParameter insertZipCodeParameter = new
SqlParameter("@ZipCode", SqlDbType.NVarChar, 10, "ZipCode");
    SqlParameter insertCityParameter = new SqlParameter("@City",
SqlDbType.NVarChar, 30, "City");
    SqlParameter insertStateParameter = new SqlParameter("@State",
SqlDbType.NVarChar, 30, "State");
    SqlParameter insertCountryIDParameter = new
SqlParameter("@CountryID", SqlDbType.UniqueIdentifier, 0,
"CountryID");
    SqlParameter insertPhoneParameter = new SqlParameter("@Phone",
SqlDbType.VarChar, 30, "Phone");
    SqlParameter insertEmailAddressParameter = new
SqlParameter("@EmailAddress", SqlDbType.NVarChar, 50,
"EmailAddress");
    SqlParameter insertWebAddressParameter = new
SqlParameter("@WebAddress", SqlDbType.NVarChar, 80, "Url");
    SqlParameter insertCreditLimitParameter = new
SqlParameter("@CreditLimit", SqlDbType.Int, 0, "CreditLimit");
    SqlParameter insertNewsSubscriberParameter = new
SqlParameter("@NewsSubscriber", SqlDbType.Bit, 0,
"NewsSubscriber");
    SqlParameter insertCreatedDateParameter = new
SqlParameter("@CreatedDate", SqlDbType.SmallDateTime, 0,
"CreatedDate");
    SqlParameter insertCreatedByParameter = new
SqlParameter("@CreatedBy", SqlDbType.VarChar, 15, "CreatedBy");


    // Assign parameters to command object
    insertCommand.Parameters.Add(insertFirstNameParameter);
    insertCommand.Parameters.Add(insertLastNameParameter);
    insertCommand.Parameters.Add(insertAddressParameter);
    insertCommand.Parameters.Add(insertZipCodeParameter);
    insertCommand.Parameters.Add(insertCityParameter);
    insertCommand.Parameters.Add(insertStateParameter);
    insertCommand.Parameters.Add(insertCountryIDParameter);
    insertCommand.Parameters.Add(insertPhoneParameter);
    insertCommand.Parameters.Add(insertEmailAddressParameter);
```

(*Code continued on the following page.*)

```
    insertCommand.Parameters.Add(insertWebAddressParameter);
    insertCommand.Parameters.Add(insertCreditLimitParameter);
    insertCommand.Parameters.Add(insertNewsSubscriberParameter);
    insertCommand.Parameters.Add(insertCreatedDateParameter);
    insertCommand.Parameters.Add(insertCreatedByParameter);

    // Declare and instantiate dataset
    DataSet customerManagementDataSet = new
DataSet("CustomerManagementDataSet");
    // Apply the full schema from the data source

customerManagementDataAdapter.FillSchema(customerManagementDataSet
, SchemaType.Source, "Customers");
    customerManagementDataAdapter.MissingSchemaAction =
MissingSchemaAction.AddWithKey;
    customerManagementDataAdapter.MissingMappingAction =
MissingMappingAction.Passthrough;
    // Populate Customers DataTable
    customerManagementDataAdapter.Fill(customerManagementDataSet,
"Customers");

    // Create new row locally
    DataRow newCustomerDataRow =
customerManagementDataSet.Tables["Customers"].NewRow();
    newCustomerDataRow["ID"] = Guid.NewGuid();
    newCustomerDataRow["FirstName"] = currentCustomer.FirstName;
    newCustomerDataRow["LastName"] = currentCustomer.LastName;
    newCustomerDataRow["Address"] = currentCustomer.Address;
    newCustomerDataRow["ZipCode"] = currentCustomer.ZipCode;
    newCustomerDataRow["City"] = currentCustomer.City;
    newCustomerDataRow["State"] = currentCustomer.State;
    newCustomerDataRow["CountryID"] = currentCustomer.CountryID;
    newCustomerDataRow["Phone"] = currentCustomer.Phone;
    newCustomerDataRow["EmailAddress"] =
currentCustomer.EmailAddress;
    newCustomerDataRow["Url"] = currentCustomer.EmailAddress;
    newCustomerDataRow["CreditLimit"] =
currentCustomer.CreditLimit;
    newCustomerDataRow["NewsSubscriber"] =
currentCustomer.NewsSubscriber;
    newCustomerDataRow["CreatedDate"] =
currentCustomer.CreatedDate;
    newCustomerDataRow["CreatedBy"] = currentCustomer.CreatedBy;
```

(*Code continued on the following page.*)

```
    // Insert new row locally

customerManagementDataSet.Tables["Customers"].Rows.Add(newCustomer
DataRow);

    // Update data source
    if
(customerManagementDataAdapter.Update(customerManagementDataSet,
"Customers") == 1)
    {
        // Instantiate new Customer object
        currentCustomer = new
CustomerManagementEntities.Customer();
        // Reload page to refresh with "blank" input controls
        Response.Redirect("~/InsertCustomer.aspx");
    }
}
```

- Save the Customer code-behind file.

▶ Task 6: Update sitemap to enable view all customers

- Open the **web.sitemap** file.
- Append the new **siteMapNode** element to the Customers siteMapNode.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers.aspx" />
```

- Save and close the **web.sitemap** file.

▶ Task 7: Create a customer and verify the Data Source

- Build the **CustomerManagement** solution.
- Run the **CustomerManagement** Web application.
- Open the **InsertCustomer** Web Form.
- Create a new customer by using the following information, and then click the **Insert** button:
  - First Name: **Kim**
  - Last Name: **Abercrombie**

- Address: **9876 Maine Road**
- Zip Code: **M24NG**
- City: **Manchester**
- Country: **Great Britain**
- Phone: **0161-123 555**
- Email Address: **kim@litwareinc.com**
- Web Address: **http://www.litwareinc.com**
- Credit Limit: **50000**
- News Subscriber: **Yes**

- Check that the new customer has been added in the data source by using the new **Customers** Web Form.
- Close Internet Explorer.

▶ Task 8: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the 10267A-GEN-DEV virtual machine.

Note: The answers to the exercises are on the Course Companion CD.

Lab Shutdown

After you complete the lab, you must turn off the 10267A-GEN-DEV virtual machine and revert the changes.

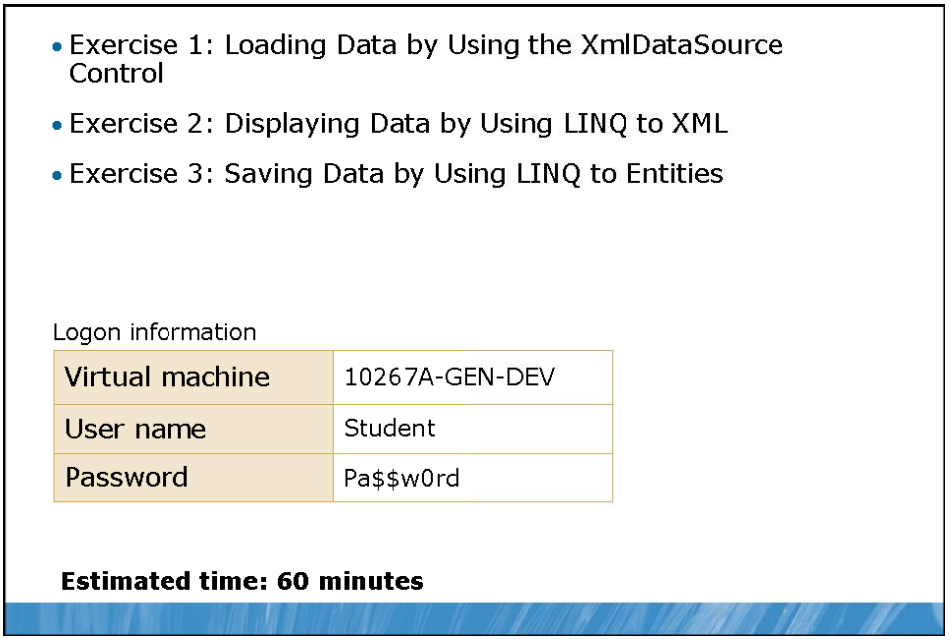
1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 9

Lab Instructions: Managing Data Access Tasks by Using LINQ (Visual Basic)

Contents:

Lab: Managing Data Access Tasks by Using LINQ



Note: You can perform tasks in this lab by using either the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will load data from an XML file, and save the data to a database by using LINQ to Entities. In addition, you will filter and display the data by using LINQ to XML.

Objectives

After completing this lab, you will be able to:

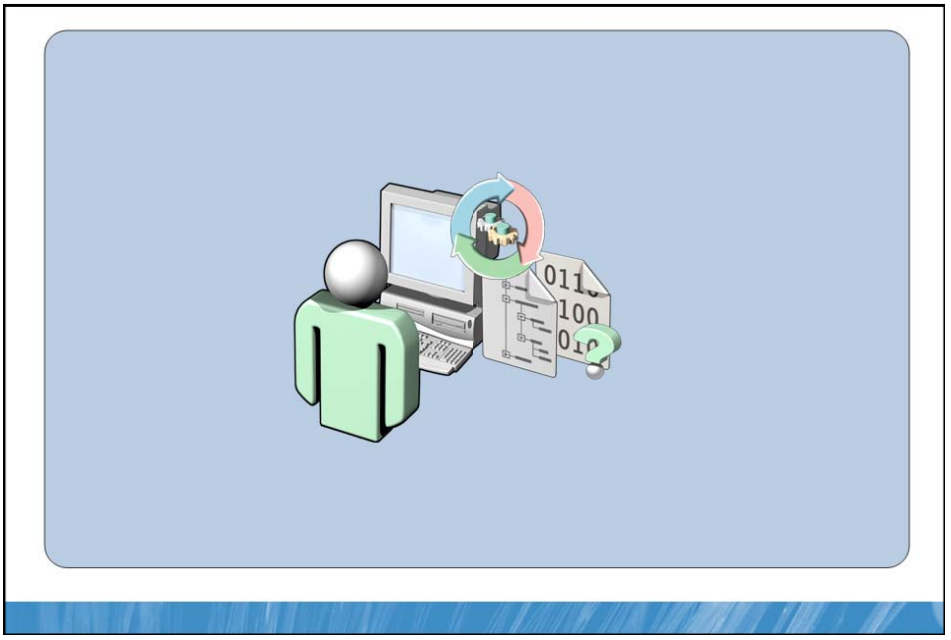
- Load data from an XML file.
- Display data by using LINQ to XML.
- Save data to a database by using LINQ to Entities.

Lab Setup

Before you begin the lab, you must:

- Start the 10267A-GEN-DEV virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization has a Web site to manage its customer information. You are responsible for managing the customer database. Updates to the country data in the database are being delivered in XML files on a monthly basis, and you need to create a Web Form for importing the XML file into the CustomerManagement SQL Server database. You also need to filter the XML data for the countries with a specified phone number from the database. Use LINQ to XML for importing and filtering the XML data, and LINQ to Entities for inserting the new data into the database.

Section 1: Visual Basic

Exercise 1: Loading Data by Using the XmlDataSource Control

The main tasks in this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Create the ImportCountries Web Form.
3. Add an **XmlDataSource** control to the Web Form.
4. Configure the **XmlDataSource** control.
5. Add a **GridView** control to the Web Form.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M9\VB folder.

▶ Task 2: Create the ImportCountries Web Form

- Add a new Web Form named **ImportCountries**, that is based on the **Site.master** master page.
- Open the **ImportCountries** Web Form in the Design view.

▶ Task 3: Add an XmlDataSource control to the Web Form

- Add an existing XML file, **D:\Labfiles\Starter\M9\Countries.xml**, to the Web site.
- Add an **XmlDataSource** control to the **Content** control of the **ImportCountries** Web Form.
- Rename the **XmlDataSource** control you just added to **CountriesXmlDataSource**.
- Save the **ImportCountries** Web Form.

▶ Task 4: Configure the XmlDataSource control

- Open the **Configure Data Source** control by using the **Show Smart Tag** option.
- Use the **Countries.xml** data file in the **XmlDataSource** control.
- Save the **ImportCountries** Web Form.

▶ Task 5: Add a GridView control to the Web Form

- Add a **GridView** control to the **Content** control of the **ImportCountries** Web Form.
- Rename the **GridView** control as **CountriesGridView**.
- Set the width of the **GridView** control to **100%**.
- Bind the **CountriesGridView** control to the **CountriesXmlDataSource** control by using the Smart Tag.
- Save the **ImportCountries** Web Form.
- Build the Web Form and fix any errors.
- View the **ImportCountries** Web Form in the browser.
- Close the Windows® Internet Explorer® browser.

Results: After completing this exercise, you will have created a Web Form, added a **GridView** and an **XmlDataSource** control to the Web Form, configured the **XmlDataSource** control for managing data, and bound the **GridView** control to the **XmlDataSource** control.

Exercise 2: Displaying Data by Using LINQ to XML

The main tasks in this exercise are as follows:

1. Add a **Button** control to the Web Form.
2. Update sitemap to enable country import.
3. Load the XML data file manually.
4. Show the filtered countries by using the **GridView** control.

▶ Task 1: Add a Button control to the Web Form

- Open the **ImportCountries** Web Form in the Source view.
- Add a **div** element with a class attribute value of **importCountriesHeader** to the **Content** control, above the **GridView** control.
- Add a **Button** control to the **div** element.
- Change the value for the **ID** attribute from **Button1** to **FilterButton**.
- Change the **Text** property of the **Button** control from **Button** to **Filter Countries**.

```
Text="Filter Countries"
```

- Open the **ImportCountries** Web Form in Design view, and select the **FilterButton** control.
- Add the default code for the **Click** event of the **FilterButton** control.
- Save the **ImportCountries** code file.
- View the **ImportCountries.aspx** in Source view.
- Add a **div** element with a class attribute value of **importResult** to the **Content** control, between the **GridView** control and the **div** element with a class attribute value of **importCountriesHeader**.

```
<div class="importResult">

</div>
```

- Add a **Label** control to the **div** element.

- Change the value for the **ID** attribute from **Label1** to **ImportResultLabel**.

```
ID="ImportResultLabel"
```

- Remove the **Text** attribute and value.

```
Text="Label"
```

- Disable view state for the **Label** control.

```
EnableViewState="false"
```

- Format and save the **ImportCountries** Web Form.

▶ Task 2: Update sitemap to enable country import

- Open the **web.sitemap** file.
- Append new **siteMapNode** element to the **Countries siteMapNode**.

```
<siteMapNode title="Import" description="Import countries"
url="~/ImportCountries.aspx" />
```

- Save and close the **web.sitemap** file.

▶ Task 3: Load the XML data file manually

- Create a private method named **loadCountries**, to load the XML data to the **ImportCountries** class.

Note: The **loadCountries** method returns an **XElement**, and takes a string parameter named **fileName**.

```
Private Function loadCountries(ByVal fileName As String) As
XElement
End Function
```

- Import the **System.Xml.Linq** namespace.

```
Imports System.Xml.Linq
```

- Add the code to the **loadCountries** method to load the content of the file by using the **XElement.Load** method, and then return to the caller of the method.

```
return XElement.Load(Server.MapPath(fileName))
```

- In the **Click** event handler for the **FilterButton** control, call the **loadCountries** method, passing Countries.xml for the filename, and saving the returned value in an **XElement** variable named **countries**.

```
Dim countries As XElement = loadCountries("Countries.xml")
```

- Create a shared private member variable of type **IEnumerable(XElement)** named **countriesWithPhNoFormat**, for holding the filtered countries. Initialize to **Nothing**, in the **ImportCountries** class.

```
Private Shared countriesWithPhNoFormat As IEnumerable(Of XElement)
= Nothing
```

- Create a new private method named **filterCountries**, for filtering the XML data in the **ImportCountries** class.

Note: The **filterCountries** method returns an **IEnumerable(XElement)**, and takes an **XElement** parameter named **countries**.

```
Private Function filterCountries(ByVal countries As XElement) As
IEnumerable(Of XElement)
End Function
```

- Add a LINQ query to the **filterCountries** method that searches the child elements of the root node in the passed countries variable, and returns the result to the caller of the method.

```
Return _
    From c In countries.Elements()
    Select c
```

- Add filtering to the LINQ query, selecting the **Country** elements for which a value for the **PhoneNoFormat** attribute has been specified.

```
Where Not String.IsNullOrEmpty(c.Attribute("PhoneNoFormat"))
```

- Call the **filterCountries** method, pass the local countries variable, and then save the returned value in the private member variable named **countriesWithPhNoFormat**, in the **Click** event handler of the **FilterButton** control.

```
countriesWithPhNoFormat = filterCountries(countries)
```

- Save the **ImportCountries** code file.

▶ Task 4: Show the filtered countries by using the GridView control

- Create a private method named **buildXmlString** for building an XML string from the filtered XML data in the **ImportCountries** class.

Note: The **buildXmlString** method returns a string and takes an **IEnumerable(XElement)** parameter named **countriesWithPhNoFormat**.

```
Private Function buildXmlString(ByVal countriesWithPhNoFormat As
IEnumerable(Of XElement)) As String
End Function
```

- Return the value of the passed **countriesWithPhNoFormat**, wrapped in a root node named **Countries**, to the **buildXmlString** method.

```
Return "<Countries>" & countriesWithPhNoFormat & "</Countries>"
```

- Call the **IEnumerable(XElement).Select** method on the **countriesWithPhNoFormat** variable to convert each country into a string.

```
countriesWithPhNoFormat.Select()
```

- Convert all the values in the **countriesWithPhNoFormat** variable to strings by using the **ToString** method in an anonymous Lambda function, placed in the call to the **Select** method.

```
countriesWithPhNoFormat.Select(Function(x) x.ToString())
```

- Convert the current output from the **Select** method to an array by using the **ToArray** method.

```
countriesWithPhNoFormat.Select(Function(x) x.ToString()).ToArray()
```

- Join the current output from the **ToArray** method by using the **String.Join** method, specifying an empty string separator.

```
String.Join("", countriesWithPhNoFormat.Select(Function(x)
x.ToString()).ToArray())
```

- Assign the return value of the **buildXmlString** method to the **Data** property of the **CountriesXmlDataSource** control, passing the **countriesWithPhNoFormat** private member variable to the **buildXmlString** method.

```
CountriesXmlDataSource.Data =
buildXmlString(countriesWithPhNoFormat)
```

- Reset the **DataFile** property of the **CountriesXmlDataSource** control, ensuring that the **Data** property is used when rendering.

```
CountriesXmlDataSource.DataFile = ""
```

- Build the **ImportCountries** Web Form, and fix any errors.
- View the ImportCountries Web Form in the browser.
- Filter the countries from the ImportCountries Web Form.
- Close the Internet Explorer browser.

Results: After completing this exercise, you will have loaded the content of the XML file manually, added a **Button** control to the Web Form, added code to the **Click** event for filtering the loaded countries, and displayed the filtered XML data in a **GridView** control.

Exercise 3: Saving Data by Using LINQ to Entities

The main tasks for this exercise are as follows:

1. Add a **Button** control to the Web Form.
2. Add an ADO.NET Entity Data Model project item to the Web site project.
3. Create an **ObjectContext** object.
4. Insert the filtered XML data in the local ObjectContext data.
5. Submit the local ObjectContext data to the database.

▶ Task 1: Add a Button control to the Web Form

- Open the **ImportCountries** Web Form in Source view.
- Add a **Button** control next to the **FilterButton** control.
- Change the value for the **ID** attribute from **Button1** to **SaveButton**.

```
ID="SaveButton"
```

- Change the value for the **Text** property from **Button** to **Save Countries**.

```
Text="Save Countries"
```

- Open the **ImportCountries** Web Form in Design view, and double-click the **SaveButton** control.
- Save the **ImportCountries** code file.

▶ Task 2: Add an ADO.NET Entity Data Model project item to the Web site project

- Add an **ADO.NET Entity Data Model** project item named **CustomerManagement.edmx**.
- In the Entity Data Model Wizard, on the **Choose Model Contents** page, click **Generate from database**, and then click **Next**.

- On the **Choose Your Data Connection** page, on the **Which data connection should your application use to connect to the database?** list, click **CustomerManagementConnectionString (Settings)**, in the box below the **Save entity connection settings in Web.config as** check box, type **Entities**, and then click **Next**.
- On the **Choose Your Database Objects** page, on the **Which database objects do you want to include in your model?** list, expand **Tables**, click **Countries (dbo)**, click **Customers (dbo)**, click **Pluralize or singularize generated object names** , and then click **Finish**.
- Save the **ADO.NET Entity Data Model** item.
- Close the **ADO.NET Entity Data Model** item.

▶ Task 3: Create an ObjectContext object

- Create a private method named **saveCountries** in the **ImportCountries** class, for saving the filtered XML data.

Note: The **saveCountries** method does not return a value, but it takes an **IEnumerable(XElement)** parameter named **countriesWithPhNoFormat**.

```
Private Sub saveCountries(ByVal countriesWithPhNoFormat As
IEnumerable(Of XElement))
End Sub
```

- Add code to create and instantiate an **ObjectContext** of type **CustomerManagementModel.Entities**, named **cmObjectContext**, retrieving the **CustomerManagementEntities** connection string from the **web.config** file by using the **System.Web.Configuration.WebConfigurationManager** class.

```
Dim cmObjectContext As New CustomerManagementModel.Entities(
    System.Web.Configuration.WebConfigurationManager.
    ConnectionStrings("Entities").ConnectionString)
```

▶ Task 4: Insert the filtered XML data in the local ObjectContext data

- Add code to loop through the **Country** elements in the passed **countriesWithPhNoFormat** variable, by using a **For Each** construct.

```
' Loop through the filtered countries
For Each country As XElement In countriesWithPhNoFormat
Next
```

- Add code to insert each country in the **Countries** collection of the **cmObjectContext** object by using the **AddObject** method, and by passing the values from the filtered XML.

```
' Loop through the filtered countries
For Each country As XElement In countriesWithPhNoFormat
    ' Add the new country to the Countries collection
    cmObjectContext.Countries.AddObject(New
CustomerManagementModel.Country With
    {
        .ID = New Guid(country.Attribute("ID").Value),
        .Name = country.Attribute("Name").Value,
        .PhoneNoFormat = country.Attribute("PhoneNoFormat").Value,
        .DialingCountryCode =
country.Attribute("DialingCountryCode").Value,
        .InternationalDialingCode =
country.Attribute("InternationalDialingCode").Value,
        .InternetTLD = country.Attribute("InternetTLD").Value
    })
Next
```

▶ Task 5: Submit the local ObjectContext data to the database

- Add a **Try Catch Finally** construct for saving the local modifications to database. The **Catch** code must catch all exceptions.

```
' Save to database
Try
Catch ex As Exception
Finally
End Try
```

- Add a call to submit the local modifications to the database by using the **SaveChanges** method of the **ObjectContext** object in the **Try** part.

```
' Save to database
Try
    cmObjectContext.SaveChanges()
Catch ex As Exception
Finally
End Try
```

- Display successful import result message by setting the **Text** property of the **ImportResultLabel** control to **Rows successfully exported to SQL Server table** in the **Try** part.

```
' Display success message
ImportResultLabel.Text = "Rows successfully exported to SQL Server
table."
```

- Add code to display a custom error message by setting the **Text** property of the **ImportResultLabel** control to **An error occurred exporting to SQL Server.
** in the **Catch** part. Append the value of the **Message** property of the caught **Exception**, and two additional line breaks **

. The text color must be set to red, by setting the **ForeColor property of the **Label** control to the value **System.Drawing.Color.Red**.

```
Catch ex as Exception
    ' Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" & ex.Message & "<br/><br/>"
    ImportResultLabel.ForeColor = System.Drawing.Color.Red
```

- Add a more specific **Catch** statement to the **Try Catch Finally** construct for catching exceptions that are thrown when errors occur during the actual update, by catching exceptions of type **System.Data.UpdateException**. The **Catch** code must be added before the existing catch.

```
' Save to database
Try
    cmObjectContext.SaveChanges()
    ' Display success message
    ImportResultLabel.Text = "Rows successfully exported to SQL
Server table."
Catch ex As System.Data.UpdateException
Catch ex As Exception
    ' Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" & ex.Message & "<br/><br/>"
    ImportResultLabel.ForeColor = System.Drawing.Color.Red
Finally
End Try
```

- Add code to display a custom error message by setting the **Text** property of the **ImportResultLabel** control to **An error occurred exporting to SQL Server.
One or more of the rows already exist in the table.

** in the new **Catch**. The text color must be set to red by setting the **ForeColor** property of the **Label** control to **System.Drawing.Color.Red**.

```
Catch ex As System.Data.UpdateException
    ' Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>One or more of the rows already exist in the
table.<br/><br/>"
    ImportResultLabel.ForeColor = System.Drawing.Color.Red
```

- Add code to dispose the **ObjectContext** object in the **Finally** part.

```
Finally
    cmObjectContext.Dispose()
```

- Call the **saveCountries** method by passing the static **countriesWithPhNoFormat** private member variable from the **Click** event handler of the **SaveButton** control.

```
saveCountries(countriesWithPhNoFormat)
```

- Build the **ImportCountries** Web Form, and fix any errors.

- View the **ImportCountries** Web Form in the Solution Explorer browser.

Note: Notice that the **CountriesGridView** control displays the content of the XML data file.

- Filter the countries from the **ImportCountries** Web Form.
- Save the imported countries in the database.

Note: If you try to save the countries that already exist in the SQL database, you will receive an error message.

▶ Task 6: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

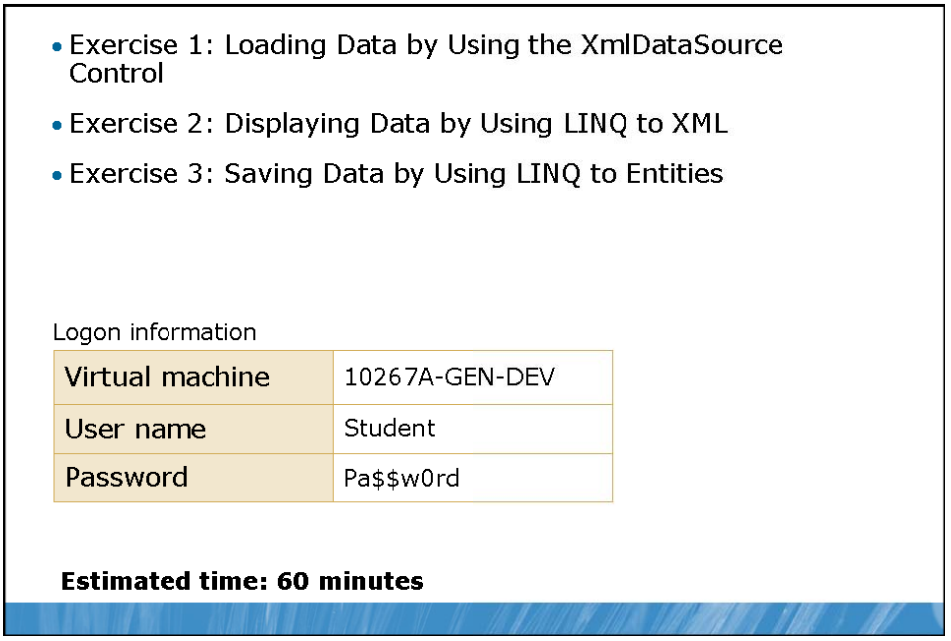
Results: After completing this exercise, you will have added a **Button** control to the Web Form, added an ADO.NET Entity Data Model project item, created an **ObjectContext** object, and saved the imported countries to the database.

Module 9

Lab Instructions: Managing Data Access Tasks by Using LINQ (Visual C#)

Contents:

Lab: Managing Data Access Tasks by Using LINQ



Note: You can perform tasks in this lab by using either the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will load data from an XML file, and save the data to a database by using LINQ to Entities. In addition, you will filter and display the data by using LINQ to XML.

Objectives

After completing this lab, you will be able to:

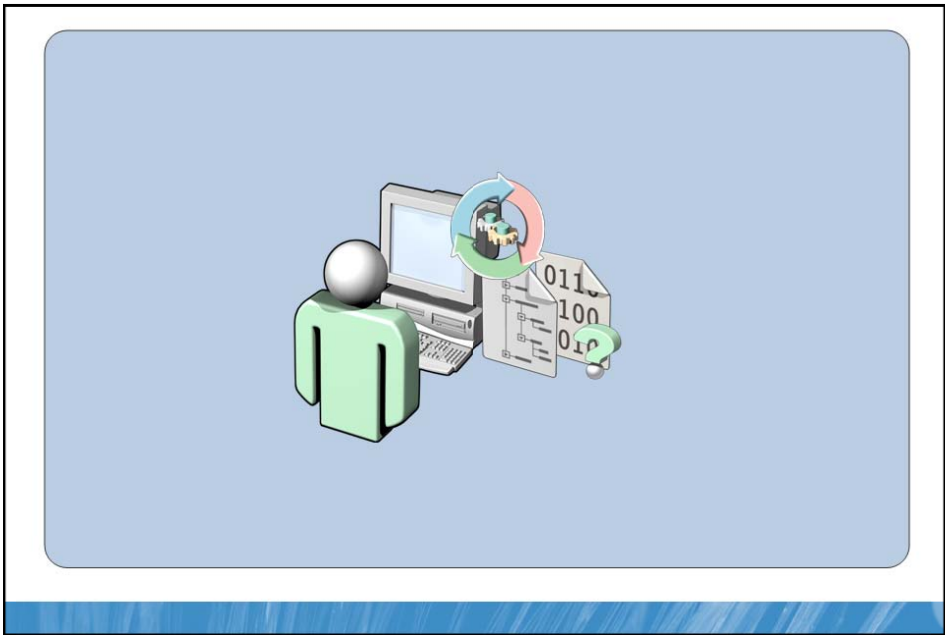
- Load data from an XML file.
- Display data by using LINQ to XML.
- Save data to a database by using LINQ to Entities.

Lab Setup

Before you begin the lab, you must:

- Start the 10267A-GEN-DEV virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization has a Web site to manage its customer information. You are responsible for managing the customer database. Updates to the country data in the database are being delivered in XML files on a monthly basis, and you need to create a Web Form for importing the XML file into the CustomerManagement SQL Server database. You also need to filter the XML data for the countries with a specified phone number from the database. Use LINQ to XML for importing and filtering the XML data, and LINQ to Entities for inserting the new data into the database.

Section 2: Visual C#

Exercise 1: Loading Data by Using the XmlDataSource Control

The main tasks in this exercise are as follows:

1. Open an existing ASP.NET Web site.
1. Create the **ImportCountries** Web Form.
2. Add an **XmlDataSource** control to the Web Form.
3. Configure the **XmlDataSource** control.
4. Add a **GridView** control to the Web Form.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M9\CS folder.

▶ Task 2: Create the ImportCountries Web Form

- Add a new Web Form named **ImportCountries**, that is based on the **Site.master** master page.
- Open the **ImportCountries** Web Form in the Design view.

▶ Task 3: Add an XmlDataSource control to the Web Form

- Add an existing XML file, **D:\Labfiles\Starter\M9\Countries.xml**, to the Web Site.
- Add an **XmlDataSource** control to the **Content** control of the **ImportCountries** Web Form.

- Rename the **XmlDataSource** control that you just added to **CountriesXmlDataSource**.
- Save the **ImportCountries** Web Form.

▶ Task 4: Configure the XmlDataSource control

- Open the **Configure Data Source** control by using the **Show Smart Tag** option.
- Use the **Countries.xml** data file in the **XmlDataSource** control.
- Save the **ImportCountries** Web Form.

▶ Task 5: Add a GridView control to the Web Form

- Add a **GridView** control to the **Content** control of the **ImportCountries** Web Form.
- Rename the **GridView** control as **CountriesGridView**.
- Set the width of the **GridView** control to **100%**.
- Bind the **CountriesGridView** control to the **CountriesXmlDataSource** control by using the Smart Tag.
- Save the **ImportCountries** Web Form.
- Build the Web Form and fix any errors.
- View the ImportCountries Web Form in the browser.
- Close Internet Explorer.

Results: After completing this exercise, you will have created a Web Form, added a **GridView** and an **XmlDataSource** control to the Web Form, configured the **XmlDataSource** control for managing data, and bound the **GridView** control to the **XmlDataSource** control.

Exercise 2: Displaying Data by Using LINQ to XML

The main tasks in this exercise are as follows:

1. Add a **Button** control to the Web Form.
2. Update sitemap to enable country import.
3. Load the XML data file manually.
4. Show the filtered countries by using the **GridView** control.

▶ Task 1: Add a Button control to the Web Form

- Open the **ImportCountries** Web Form in the Source view.
- Add a **div** element with a class attribute value of **importCountriesHeader** to the **Content** control, above the **GridView** control.
- Add a **Button** control to the **div** element.
- Change the value for the **ID** attribute from **Button1** to **FilterButton**.
- Change the **Text** property of the **Button** control from **Button** to **Filter Countries**.

```
Text="Filter Countries"
```

- Open the **ImportCountries** Web Form in Design view, and then select the **FilterButton** control.
- Add the default code for the **Click** event of the **FilterButton** control.
- Save the **ImportCountries** code file.
- View the **ImportCountries.aspx** in Source view.
- Add a **div** element with a class attribute value of **importResult** to the **Content** control, between the **GridView** control and the **div** element with a class attribute value of **importCountriesHeader**.

```
<div class="importResult">

</div>
```

- Add a **Label** control to the **div** element.

- Change the value for the **ID** attribute from **Label1** to **ImportResultLabel**.

```
ID="ImportResultLabel"
```

- Remove the **Text** attribute and value.

```
Text="Label"
```

- Disable view state for the **Label** control.

```
EnableViewState="false"
```

- Format and save the **ImportCountries** Web Form.

▶ Task 2: Update sitemap to enable country import

- Open the **web.sitemap** file.
- Append new **siteMapNode** element to the **Countries siteMapNode**.

```
<siteMapNode title="Import" description="Import countries"
url="~/ImportCountries.aspx" />
```

- Save and close the **web.sitemap** file.

▶ Task 3: Load the XML data file manually

- Create a private method named **loadCountries**, to load the XML data to the **ImportCountries** class.

Note: The **loadCountries** method returns an **XElement**, and takes a string parameter named **fileName**.

```
private XElement loadCountries(string fileName)
{
}
```

- Import the **System.Xml.Linq** namespace.

```
using System.Xml.Linq;
```

- Add the code to the **loadCountries** method to load the content of the file by using the **XElement.Load** method, and then return to the caller of the method.

```
return XElement.Load(Server.MapPath(fileName));
```

- In the **Click** event handler for the **FilterButton** control, call the **loadCountries** method, passing Countries.xml for the filename, and saving the returned value in an XElement variable named **countries**.

```
XElement countries = loadCountries("Countries.xml");
```

- Create a static private member variable of type **IEnumerable(XElement)** named **countriesWithPhNoFormat**, for holding the filtered countries. Initialize to **null**, in the **ImportCountries** class.

```
static private IEnumerable<XElement> countriesWithPhNoFormat =
null;
```

- Create a new private method named **filterCountries**, for filtering the XML data in the **ImportCountries** class.

Note: The **filterCountries** method returns an **IEnumerable(XElement)**, and takes an **XElement** parameter named **countries**.

```
private IEnumerable<XElement> filterCountries(XElement countries)
{
}
```

- Add a LINQ query to the **filterCountries** method that searches the child elements of the root node in the passed countries variable, and returns the result to the caller of the method.

```
return
    from c in countries.Elements()
    select c;
```

- Add filtering to the LINQ query, selecting the **Country** elements for which a value for the **PhoneNoFormat** attribute has been specified.

```
where c.Attribute("PhoneNoFormat") != null &&
    (string) c.Attribute("PhoneNoFormat").Value != ""
```

- Call the **filterCountries** method, pass the local countries variable, and then save the returned value in the private member variable named **countriesWithPhNoFormat**, in the **Click** event handler of the **FilterButton** control.

```
countriesWithPhNoFormat = filterCountries(countries);
```

- Save the **ImportCountries** code file.

▶ Task 4: Show the filtered countries by using the GridView control

- Create a private method named **buildXmlString** for building an XML string from the filtered XML data in the **ImportCountries** class.

Note: The **buildXmlString** method returns a string and takes an **IEnumerable(XElement)** parameter named **countriesWithPhNoFormat**.

```
private string buildXmlString(IEnumerable<XElement>
countriesWithPhNoFormat)
{
}
```

- Return the value of the passed **countriesWithPhNoFormat**, wrapped in a root node named **Countries**, to the **buildXmlString** method.

```
return "<Countries>" + countriesWithPhNoFormat + "</Countries>";
```

- Call the **IEnumerable(XElement).Select** method on the **countriesWithPhNoFormat** variable to convert each country into a string.

```
countriesWithPhNoFormat.Select()
```

- Convert all the values in the **countriesWithPhNoFormat** variable to strings by using the **ToString** method in an anonymous Lambda function, placed in the call to the **Select** method.

```
countriesWithPhNoFormat.Select(x => x.ToString())
```

- Convert the current output from the **Select** method to an array by using the **ToArray** method.

```
countriesWithPhNoFormat.Select(x => x.ToString()).ToArray()
```

- Join the current output from the **ToArray** method by using the **String.Join** method, specifying an empty string separator.

```
string.Join("", countriesWithPhNoFormat.Select(x =>
x.ToString()).ToArray())
```

- Assign the return value of the **buildXmlString** method to the **Data** property of the **CountriesXmlDataSource** control, passing the **countriesWithPhNoFormat** private member variable to the **buildXmlString** method.

```
CountriesXmlDataSource.Data =
buildXmlString(countriesWithPhNoFormat);
```

- Reset the **DataFile** property of the **CountriesXmlDataSource** control, ensuring that the **Data** property is used when rendering.

```
CountriesXmlDataSource.DataFile = "";
```

- Build the **ImportCountries** Web Form, and fix any errors.
- View the **ImportCountries** Web Form in the browser.
- Filter the countries from the **ImportCountries** Web Form.
- Close Internet Explorer.

Results: After completing this exercise, you will have loaded the content of the XML file manually, added a **Button** control to the Web Form, added code to the **Click** event for filtering the loaded countries, and displayed the filtered XML data in a **GridView** control.

Exercise 3: Saving Data by Using LINQ to Entities

The main tasks for this exercise are as follows:

1. Add a **Button** control to the Web Form.
2. Add an ADO.NET Entity Data Model project item to the Web site project.
3. Create an **ObjectContext** object.
4. Insert the filtered XML data in the local ObjectContext data.
5. Submit the local ObjectContext data to the database.

▶ Task 1: Add a Button control to the Web Form

- Open the **ImportCountries** Web Form in the Source view.
- Add a **Button** control next to the **FilterButton** control.
- Change the value for the **ID** attribute from **Button1** to **SaveButton**.

```
ID="SaveButton"
```

- Change the value for the **Text** property from **Button** to **Save Countries**.

```
Text="Save Countries"
```

- Open the **ImportCountries** Web Form in Design view, and double-click the **SaveButton** control.
- Save the **ImportCountries** code file.

▶ Task 2: Add an ADO.NET Entity Data Model project item to the Web site project

- Add an **ADO.NET Entity Data Model** project item named **CustomerManagement.edmx**.
- In the Entity Data Model Wizard, on the **Choose Model Contents** page, click **Generate from database**, and then click **Next**.

- On the **Choose Your Data Connection** page, on the **Which data connection should your application use to connect to the database?** list, click **CustomerManagementConnectionString (Settings)**, in the box below the **Save entity connection settings in Web.config as** check box, type **Entities**, and then click **Next**.
- On the **Choose Your Database Objects** page, on the **Which database objects do you want to include in your model?** list, expand **Tables**, click **Countries (dbo)**, click **Customers (dbo)**, click **Pluralize or singularize generated object names**, and then click **Finish**.
- Save the **ADO.NET Entity Data Model** item.
- Close the **ADO.NET Entity Data Model** item.

▶ Task 3: Create an ObjectContext object

- Create a private method named **saveCountries** in the **ImportCountries** class, for saving the filtered XML data.

Note: The **saveCountries** method does not return a value, but it takes an **IEnumerable(XElement)** parameter named **countriesWithPhNoFormat**.

```
private void saveCountries(IEnumerable<XElement>
countriesWithPhNoFormat)
{
}
```

- Add code to create and instantiate an **ObjectContext** of type **CustomerManagementModel.Entities** named **cmObjectContext**, retrieving the **CustomerManagementEntities** connection string from the **web.config** file by using the **System.Web.Configuration.WebConfigurationManager** class.

```
CustomerManagementModel.Entities cmObjectContext =
    new CustomerManagementModel.Entities(
        System.Web.Configuration.WebConfigurationManager.
        ConnectionStrings["Entities"].ConnectionString);
```

▶ Task 4: Insert the filtered XML data in the local ObjectContext data

- Add code to loop through the **Country** elements in the passed **countriesWithPhNoFormat** variable, by using a **For Each** construct.

```
// Loop through the filtered countries
foreach (XElement country in countriesWithPhNoFormat)
{
}
```

- Add code to insert each country in the **Countries** collection of the **cmObjectContext** object by using the **AddObject** method, and by passing the values from the filtered XML.

```
// Add the new country to the Countries collection
cmObjectContext.Countries.AddObject(new
CustomerManagementModel.Country
{
    ID = new Guid(country.Attribute("ID").Value),
    Name = country.Attribute("Name").Value,
    PhoneNoFormat = country.Attribute("PhoneNoFormat").Value,
    DialingCountryCode =
country.Attribute("DialingCountryCode").Value,
    InternationalDialingCode =
country.Attribute("InternationalDialingCode").Value,
    InternetTLD = country.Attribute("InternetTLD").Value
});
```

▶ Task 5: Submit the local ObjectContext data to the database

- Add a **try catch finally** construct for saving the local modifications to database. The **catch** code must catch all exceptions.

```
// Save to database
try
{
}
catch (Exception ex)
{
}
finally
{
}
```

- Add a call to submit the local modifications to the database by using the **SaveChanges** method of the **ObjectContext** object in the **try** part.

```
// Save to database
try
{
    cmObjectContext.SaveChanges();
}
catch (Exception ex)
{
}
finally
{
}
```

- Display successful import result message, by setting the **Text** property of the **ImportResultLabel** control to **Rows successfully exported to SQL Server table** in the **try** part.

```
// Display success message
ImportResultLabel.Text = "Rows successfully exported to SQL Server
table.";
```

- Add code to display a custom error message by setting the **Text** property of the **ImportResultLabel** control to **An error occurred exporting to SQL Server.
** in the **Catch** part. Append the value of the **Message** property of the caught **Exception**, and two additional line breaks **

. The text color must be set to red, by setting the **ForeColor property of the **Label** control to the value **System.Drawing.Color.Red**.

```
catch (Exception ex)
{
    // Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" + ex.Message + "<br/><br/>";
    ImportResultLabel.ForeColor = System.Drawing.Color.Red;
}
```

- Add a more specific **catch** to the **try catch finally** construct for catching exceptions that are thrown when errors occur during the actual update, by catching exceptions of type **System.Data.UpdateException**. The **catch** code must be added before the existing catch.

```
// Save to database
try
{
    cmObjectContext.SaveChanges();

    // Display success message
    ImportResultLabel.Text = "Rows successfully exported to SQL
Server table.";
}
catch (System.Data.UpdateException)
{
}
catch (Exception ex)
{
    // Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" + ex.Message + "<br/><br/>";
    ImportResultLabel.ForeColor = System.Drawing.Color.Red;
}
finally
{
}
```

- Add code to display a custom error message by setting the **Text** property of the **ImportResultLabel** control to **An error occurred exporting to SQL Server.
One or more of the rows already exist in the table.

** in the new **catch**. The text color must be set to red, by setting the **ForeColor** property of the **Label** control to the value **System.Drawing.Color.Red**.

```
catch (System.Data.UpdateException)
{
    // Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>One or more of the rows already exist in the
table.<br/><br/>";
    ImportResultLabel.ForeColor = System.Drawing.Color.Red;
}
```

- Add code to dispose the **ObjectContext** object in the **finally** part.

```
finally
{
    cmObjectContext.Dispose();
}
```

- Call the **saveCountries** method by passing the static **countriesWithPhNoFormat** private member variable from the **Click** event handler of the **SaveButton** control.

```
saveCountries(countriesWithPhNoFormat);
```

- Build the **ImportCountries** Web Form, and fix any errors.
- View the **ImportCountries** Web Form in the Solution Explorer browser.

Note: Notice that the CountriesGridView control displays the content of the XML data file.

- Filter the countries from the **ImportCountries** Web Form.
- Save the imported countries in the database.

Note: If you try to save the countries that already exist in the SQL database, you will receive an error message.

Results: After completing this exercise, you will have added a **Button** control to the Web Form, added an ADO.NET Entity Data Model project item, created an **ObjectContext** object, and saved the imported countries to the database.

▶ Task 6: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

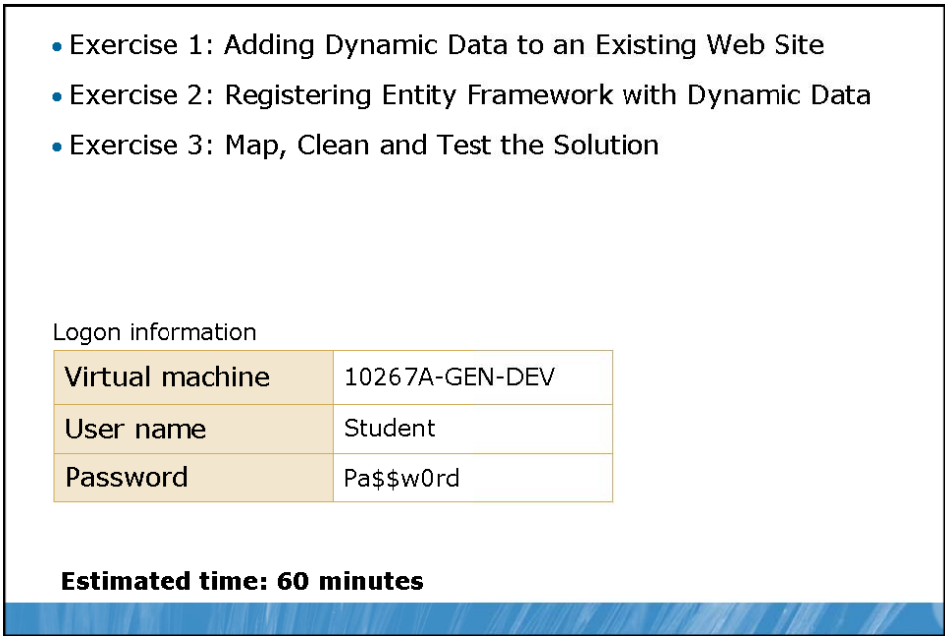
Note: The answers to the exercises are on the Course Companion CD.

Module 10

Lab Instructions: Managing Data by Using Microsoft® ASP.NET Dynamic Data (Visual Basic)

Contents:

Lab: Managing Data by Using ASP.NET Dynamic Data



Note: You can perform tasks in this lab by using either Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will add Dynamic Data to an existing Web site, and then register LINQ to Entities by using Dynamic Data. In addition, you will add metadata to the data model, and test the Dynamic Data functionality in the Web site.

Objectives

After completing this lab, you will be able to:

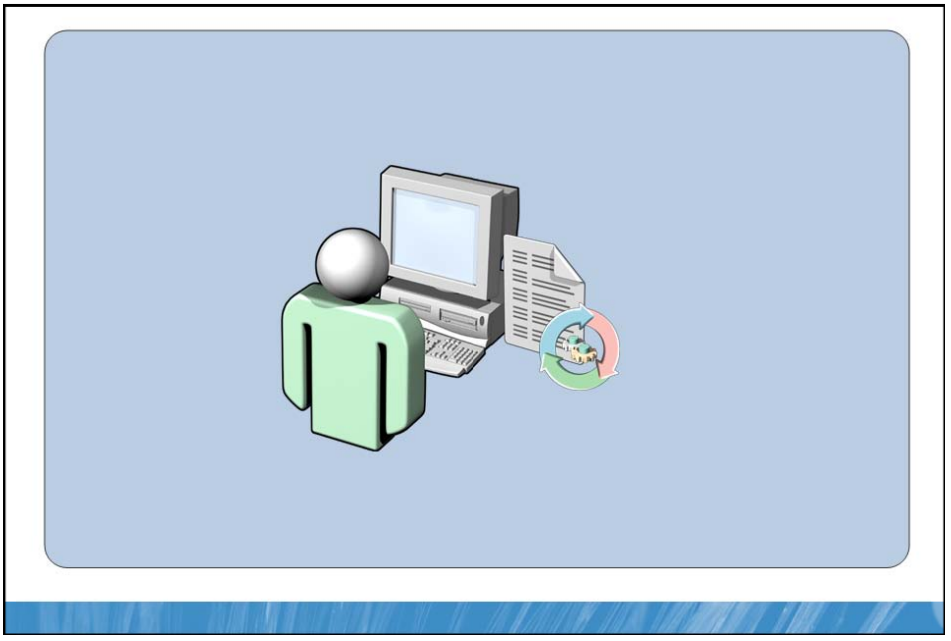
- Add dynamic data to an existing Web site.
- Register Entity Framework with Dynamic Data.
- Test the Dynamic Data functionality.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage its customer information.

The sales team of the organization wants to use the ASP.NET application to display and modify data contained in the CustomerManagement database. In addition, the sales team wants to add new records in the database by using Web Forms. To facilitate this, you need to add dynamic data to the Web site, and register the existing LINQ to Entities **ObjectContext** by using dynamic data.

Section 1: Visual Basic

Exercise 1: Adding Dynamic Data to an Existing Web Site

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Create a sample dynamic data Web site.
3. Add the dynamic data default files and folders.
4. Add the ScriptManager control to the master page.
5. Update dynamic data page templates to use an existing master page.
6. Copy dynamic data styles.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the CustomerManagement solution from the D:\Labfiles\Starter\M10\VB folder.

▶ Task 2: Create a sample dynamic data Web site

- Open a second instance of Visual Studio 2010.
- Create a new ASP.NET Dynamic Data Web site in the D:\Labfiles\Starter\M10\VB\SampleDDWebSite folder.
- Close the second instance of Visual Studio 2010.

▶ Task 3: Add the dynamic data default files and folders

- Add the **DynamicData** folder and default dynamic data content from the SampleDDWebSite Web site to the **CustomerManagement** Web site by copying the DynamicData folder from the path, **D:\Labfiles\Starter\M10\VB\SampleDDWebSite\DynamicData**.
- In Solution Explorer, refresh the Web site content to ensure the addition of the **DynamicData** folder.

▶ Task 4: Add the ScriptManager control to the master page

- Open the **Site.master** master page in Source view.
- Add a **ScriptManager** control before the **div** element with a value of **content** for the **class** attribute by adding an **asp:ScriptManager** element with an **ID** attribute value of **MainScriptManager**.
- Save the **Site.master** master page.
- Close the Site.master master page.

▶ Task 5: Update dynamic data page templates to use an existing master page

- Open the **Find and Replace** dialog box.
- Replace the **ContentPlaceHolder1** name that is referenced in the Dynamic Data Page Templates, with the name of the **ContentPlaceHolder** control that is used in the existing master page, **MainContentPlaceHolder**.
- In all of the opened and modified Dynamic Data page templates—**Details.aspx**, **Edit.aspx**, **Insert.aspx**, **List.aspx**, and **ListDetails.aspx**—remove the **Content** control with an **ID** attribute value of **headContent**.
- Save and close all modified files.

▶ Task 6: Copy dynamic data styles

- Copy the content of the **D:\Labfiles\Starter\M10\VB \SampleDDWebSite\Site.css** stylesheet.
- Add the copied content to the existing **Styles\Site.css** stylesheet.
- Save and close the **Site.css** file.

Results: After completing this exercise, you will have added dynamic data to an existing Web site.

Exercise 2: Registering Entity Framework with Dynamic Data

The main tasks for this exercise are as follows:

1. Add a Global Application Class file.
2. Import the required namespaces.
3. Get Entity Model namespace.
4. Register the ObjectContext with the metadata model.
5. Add a generic route to the routing table.

▶ Task 1: Add a Global Application Class file

- Add Global Application Class project item to the **CustomerManagement** Web site.

▶ Task 2: Import the required namespaces

- Add an **Import** directive to the Global Application Class with the **Namespace** attribute set to **System.Web.Routing**.

```
<%@ Import Namespace="System.Web.Routing" %>
```

▶ Task 3: Get Entity Model namespace

- Open the CustomerManagement Entity Model **CustomerManagement.edmx** that is located in the App_Code folder, in the ADO.NET Entity Data Model Designer (Entity Designer).
- Notice the namespace that is used for the code that is generated for the entity model.
- Close the CustomerManagement Entity Model.

▶ Task 4: Register the ObjectContext with the metadata model

- Create a public **Shared** procedure named **Register**, which takes a single parameter named **routes**, of type **RouteCollection**.

Note: The **Register** procedure does not return a value.

```
Public Shared Sub Register(ByVal routes As RouteCollection)
End Sub
```

- In the Global.asax file, in the **Application_Start** method, call the **Register** procedure by passing **RouteTable.Routes** as the only parameter.

```
Register(RouteTable.Routes)
```

- Add and initialize a new private shared variable of type **MetaModel**, named **defaultModel**. Initialize by using the default and parameterless constructor.

```
Private Shared defaultModel As New MetaModel()
```

- Add a new public shared property of type **MetaModel**, named **DefaultMetaModel**. Make the property read-only and have it return the private and shared variable, **defaultModel**.

```
Public Shared ReadOnly Property DefaultMetaModel As MetaModel
    Get
        Return defaultModel
    End Get
End Property
```

- Register the **CustomerManagementModel.Entities ObjectContext** in the **Register** procedure with scaffolding enabled for all tables by using the **RegisterContext** method of the **MetaModel** class.

```
DefaultMetaModel.RegisterContext(GetType(CustomerManagementModel.E
ntities), New ContextConfiguration() With

    .ScaffoldAllTables = True
})
```

▶ Task 5: Add a generic route to the routing table

- In the **Register** procedure, add a generic route of type **DynamicDataRoute** for the **List**, **Details**, **Edit**, and **Insert** actions, in that order. Use the DefaultMetaModel as the meta model to redirect to the page template of the same name, which is prefixed in the URL with the table name. Add the route by using the **Add** method of the passed **routes** parameter.

```
routes.Add(New DynamicDataRoute("{table}/{action}.aspx") With {
    .Constraints = New RouteValueDictionary(New With {.Action =
"List|Details|Edit|Insert"}),
    .Model = DefaultMetaModel
})
```

- Save the Global Application Class project item.
- Close the Global Application Class project item.

Results: After completing this exercise, you will have registered Entity Framework with Dynamic Data.

Exercise 3: Map, Clean, and Test the Solution

The main tasks for this exercise are as follows:

1. Map navigation to Dynamic Data page templates.
2. Remove superfluous solution and project items.
3. Test the Dynamic Data functionality.

▶ Task 1: Map navigation to Dynamic Data page templates

- Open the web.sitemap file.
- Map the **New** menu command in the **Customers** menu to the Insert page template, by modifying the **url** attribute for the **New** siteMapNode within the **Customers** siteMapNode, to a value of **~/Customers/Insert.aspx**.

```
<siteMapNode title="New" description="Create New Customer"
url="~/Customers/Insert.aspx" />
```

- Map the **All** menu command in the **Customers** menu to the List page template, by modifying the **url** attribute for the **All** siteMapNode within the **Customers** siteMapNode, to a value of **~/Customers/List.aspx**.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers/List.aspx" />
```

- Add a **New** menu command to the **Countries** menu to make it use the Insert page template, by adding a new siteMapNode within the **Countries** siteMapNode.

```
<siteMapNode title="New" description="Create New Country"
url="~/Countries/Insert.aspx" />
```

- Add an **All** menu command to the **Countries** menu to make it use the List page template, by adding a new siteMapNode within the **Countries** siteMapNode.

```
<siteMapNode title="All" description="View All Countries"
url="~/Countries/List.aspx" />
```

- Save and close the web.sitemap file.

▶ Task 2: Remove superfluous solution and project items

- Remove the **CustomerManagementEntities** project from the CustomerManagement solution.
- Delete the **Customers** Web Form in the CustomerManagement Web site.
- Delete the **InsertCustomer** Web Form in the CustomerManagement Web site.
- Delete the **Customer** user control in the CustomerManagement Web site.

▶ Task 3: Test the Dynamic Data functionality

- Run the **CustomerManagement** application.
- Show all customers, by clicking **All**, on the **Customers** menu.
- Close all open windows.

▶ Task 4: Turn off the virtual machine and revert the changes

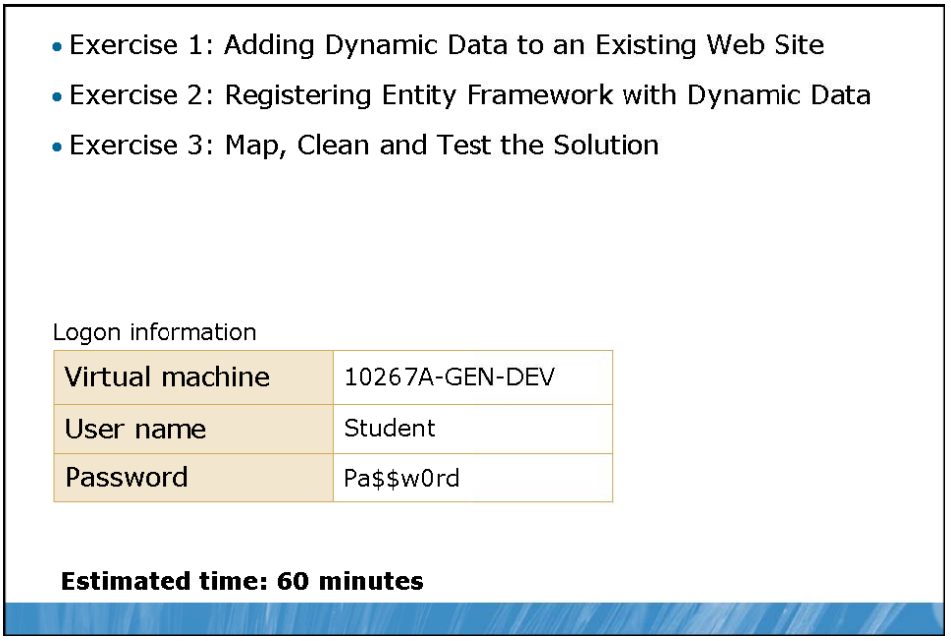
- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

Module 10

Lab Instructions: Managing Data by Using Microsoft® ASP.NET Dynamic Data (Visual C#)

Contents:

Lab: Managing Data by Using ASP.NET Dynamic Data



Note: You can perform tasks in this lab by using either Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will add Dynamic Data to an existing Web site, and then register LINQ to Entities by using Dynamic Data. In addition, you will add metadata to the data model, and test the Dynamic Data functionality in the Web site.

Objectives

After completing this lab, you will be able to:

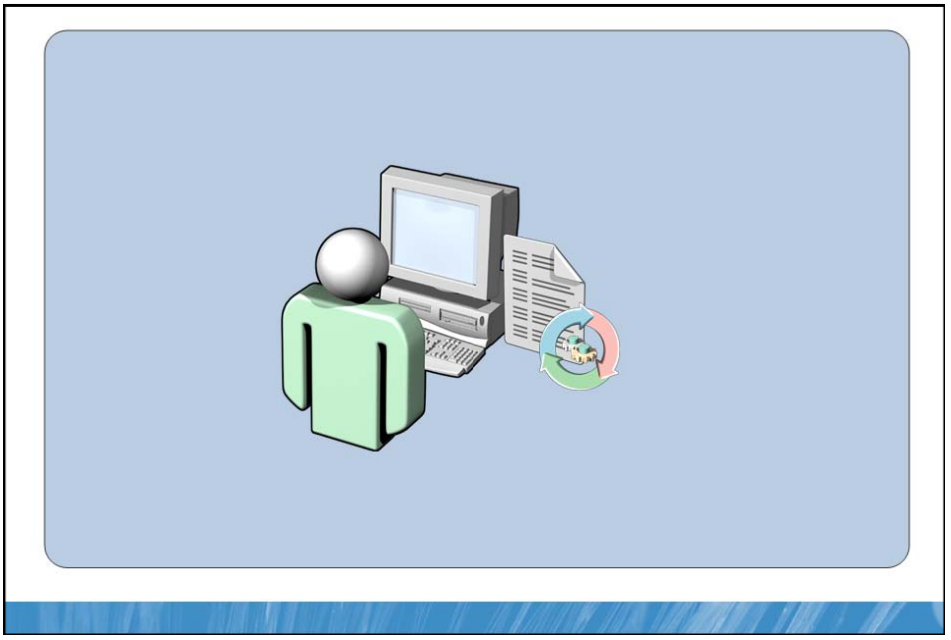
- Add dynamic data to an existing Web site.
- Register Entity Framework with Dynamic Data.
- Test the Dynamic Data functionality.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage its customer information.

The sales team of the organization wants to use the ASP.NET application to display and modify data contained in the CustomerManagement database. In addition, the sales team wants to add new records in the database by using Web Forms. To facilitate this, you need to add dynamic data to the Web site, and register the existing LINQ to Entities **ObjectContext** by using dynamic data.

Section 2: Visual C#

Exercise 1: Adding Dynamic Data to an Existing Web Site

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Create a sample Dynamic Data Web site.
3. Add the dynamic data default files and folders.
4. Add the **ScriptManager** control to the master page.
5. Update Dynamic Data page templates to use an existing master page.
6. Copy dynamic data styles.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the CustomerManagement solution from the D:\Labfiles\Starter\M10\CS folder.

▶ Task 2: Create a sample Dynamic Data Web site

- Open a second instance of Visual Studio 2010.
- Create a new ASP.NET Dynamic Data Web site in the D:\Labfiles\Starter\M10\CS\SampleDDWebSite folder.
- Close the second instance of Visual Studio 2010.

▶ Task 3: Add the dynamic data default files and folders

- Add the **DynamicData** folder and default dynamic data content from the **SampleDDWebSite** Web site to the **CustomerManagement** Web site by copying the DynamicData folder from the path, **D:\Labfiles\Starter\M10\CS\SampleDDWebSite\DynamicData**.
- In Solution Explorer, refresh the Web site content to ensure the addition of the **DynamicData** folder.

▶ Task 4: Add the ScriptManager control to the master page

- Open the **Site.master** master page in Source view.
- Add a **ScriptManager** control before the **div** element with a value of **content** for the class attribute by adding an **asp:ScriptManager** element with an **ID** attribute value of **MainScriptManager**.
- Save the **Site.master** master page.
- Close the **Site.master** master page.

▶ Task 5: Update the Dynamic Data page templates to use an existing master page

- Open the **Find and Replace** dialog box.
- Replace the **ContentPlaceHolder1** name that is referenced in the Dynamic Data Page Templates, with the name of the **ContentPlaceHolder** control used in the existing master page, **MainContentPlaceHolder**.
- In all of the opened and modified Dynamic Data page templates—**Details.aspx**, **Edit.aspx**, **Insert.aspx**, **List.aspx**, and **ListDetails.aspx**—remove the **Content** control with an **ID** attribute value of **headContent**.
- Save and close all modified files.

▶ Task 6: Copy dynamic data styles

- Copy the content of the **D:\Labfiles\Starter\M10\CS\SampleDDWebSite\Site.css** stylesheet.
- Add the copied content to the existing **Styles\Site.css** stylesheet.
- Save and close the **Site.css** file.

Results: After completing this exercise, you will have added dynamic data to an existing Web site.

Exercise 2: Registering Entity Framework with Dynamic Data

The main tasks for this exercise are as follows:

1. Add a Global Application Class file.
2. Import the required namespaces.
3. Get **Entity Model** namespace.
4. Register the **ObjectContext** with the metadata model.
5. Add a generic route to the routing table.

▶ Task 1: Add a Global Application Class file

- Add Global Application Class project item to the **CustomerManagement** Web site.

▶ Task 2: Import the required namespaces

- Add an **Import** directive to the Global Application Class with the **Namespace** attribute set to **System.Web.Routing**.

```
<%@ Import Namespace="System.Web.Routing" %>
```

▶ Task 3: Get Entity Model namespace

- Open the CustomerManagement Entity Model **CustomerManagement.edmx** that is located in the App_Code folder, in the ADO.NET Entity Data Model Designer (Entity Designer).
- Notice the namespace that is used for the code that is generated for the entity model.
- Close the CustomerManagement Entity Model.

▶ Task 4: Register the ObjectContext with the metadata model

- Create a public **static** procedure named **Register**, which takes a single parameter named **routes**, of type **RouteCollection**.

Note: The **Register** procedure does not return a value.

```
public static void Register(RouteCollection routes)
{
}
```

- In the Global.asax file, in the **Application_Start** method, call the **Register** procedure by passing **RouteTable.Routes** as the only parameter.

```
Register(RouteTable.Routes);
```

- Add and initialize a new private static variable of type **MetaModel**, named **defaultModel**. Initialize by using the default and parameterless constructor.

```
private static MetaModel defaultModel = new MetaModel();
```

- Add a new public static property of type **MetaModel**, named **DefaultMetaModel**. Make the property read-only, and have it return the private and static variable, **defaultModel**.

```
public static MetaModel DefaultMetaModel
{
    get
    {
        return defaultModel;
    }
}
```

- Register the **CustomerManagementModel.Entities ObjectContext** in the **Register** procedure with scaffolding enabled for all tables by using the **RegisterContext** method of the **MetaModel** class.

```
DefaultMetaModel.RegisterContext(typeof(CustomerManagementModel.En
tities), new ContextConfiguration()
{
    ScaffoldAllTables = true
});
```

▶ Task 5: Add a generic route to the routing table

- In the **Register** procedure, add a generic route of type **DynamicDataRoute**, for the **List**, **Details**, **Edit**, and **Insert** actions, in that order. Use the DefaultMetaModel as the meta model to redirect to the page template of the same name, which is prefixed in the URL with the table name. Add the route by using the **Add** method of the passed **routes** parameter.

```
routes.Add(new DynamicDataRoute("{table}/{action}.aspx") {
    Constraints = new RouteValueDictionary(new { action =
"List|Details|Edit|Insert" }),
    Model = DefaultMetaModel
});
```

- Save the Global Application Class project item.
- Close the Global Application Class project item.

Results: After completing this exercise, you will have registered Entity Framework with Dynamic Data.

Exercise 3: Map, Clean, and Test the Solution

The main tasks for this exercise are as follows:

1. Map navigation to Dynamic Data page templates.
2. Remove superfluous solution and project items.
3. Test the Dynamic Data functionality.

▶ Task 1: Map navigation to Dynamic Data page templates

- Open the web.sitemap file.
- Map the **New** menu command in the **Customers** menu to the Insert page template, by modifying the **url** attribute for the **New** siteMapNode within the **Customers** siteMapNode, to a value of **~/Customers/Insert.aspx**.

```
<siteMapNode title="New" description="Create New Customer"
url="~/Customers/Insert.aspx" />
```

- Map the **All** menu command in the **Customers** menu to the List page template, by modifying the **url** attribute for the **All** siteMapNode within the **Customers** siteMapNode, to a value of **~/Customers/List.aspx**.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers/List.aspx" />
```

- Add a **New** menu command to the **Countries** menu to make it use the Insert page template, by adding a new **siteMapNode** within the **Countries** siteMapNode.

```
<siteMapNode title="New" description="Create New Country"
url="~/Countries/Insert.aspx" />
```

- Add an **All** menu command to the **Countries** menu to make it use the List page template, by adding a new siteMapNode within the **Countries** siteMapNode.

```
<siteMapNode title="All" description="View All Countries"
url="~/Countries/List.aspx" />
```

- Save and close the web.sitemap file.

▶ Task 2: Remove superfluous solution and project items

- Remove the **CustomerManagementEntities** project from the CustomerManagement solution.
- Delete the **Customers** Web Form in the CustomerManagement Web site.
- Delete the **InsertCustomer** Web Form in the CustomerManagement Web site.
- Delete the **Customer** user control in the CustomerManagement Web site.

▶ Task 3: Test the Dynamic Data functionality

- Run the **CustomerManagement** application.
- In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **All**.
- Close all open windows.

▶ Task 4: Turn off the virtual machine and revert the changes

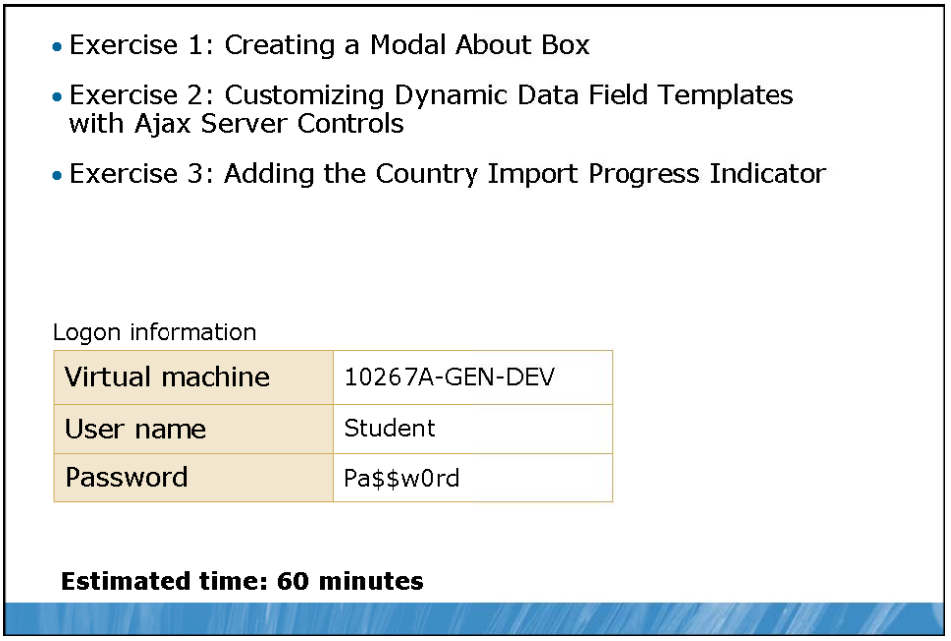
- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

Module 11

Lab Instructions: Creating a Microsoft® ASP.NET Ajax-enabled Web Forms Application (Visual Basic)

Contents:

Lab: Creating a Microsoft ASP.NET Ajax-Enabled Web Forms Application



Note: You can perform tasks in this lab either by using the Microsoft Visual Basic® or Microsoft Visual C#® programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will implement partial-page updates by using ASP.NET Ajax, create a modal dialog, and customize Dynamic Data field templates by using Ajax server controls. In addition, you will display progress when updating a page with partial rendering.

Objectives

After completing this lab, you will be able to:

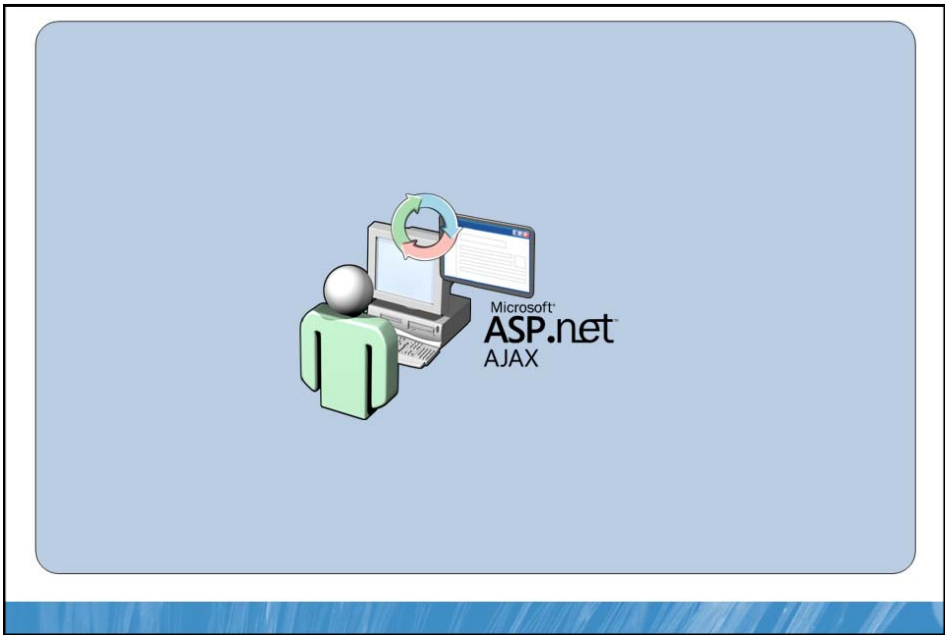
- Implement partial-page updates by using ASP.NET Ajax.
- Display a modal dialog.
- Display a client-side controlled Calendar.
- Display progress when updating a page with partial rendering.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses.NET applications to create, customize, and manage its customer information.

You are assigned the task of improving the responsiveness of the Web application user interface. You also need to avoid full-page updates where possible, and display a message when performing a partial update. For this purpose, you need to implement ASP.NET Ajax in the Customer Management application. You also need to ensure that partial page updates are possible without writing any client-side coding. In addition, you need to create a modal **About** box, add a client-side controlled **Calendar** control to a Dynamic Data field template user control, and add a progress indicator to the ImportCountries Web Form.

Section 1: Visual Basic

Exercise 1: Creating a Modal About Box

The main tasks for this exercise are as follows:

1. Open an existing Web site.
2. Add a new Web Form.
3. Add a reference to the Ajax Control Toolkit assembly.
4. Register the Ajax Toolkit assembly, namespace, and TagPrefix.
5. Change the **ScriptManager** control in the master page.
6. Add the **ScriptManagerProxy** control.
7. Add the HTML and UI controls.
8. Add the event handler and code.
9. Map navigation to the About Web Form.
10. Test the modal About box.

▶ Task 1: Open an existing Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M11\VB folder.

▶ Task 2: Add a new Web Form

- Add a new Web Form named **About.aspx**, based on the master page **Site.master**.
- Change the **ID** property of the **Content** control to **AboutContent**.

```
<asp:Content ID="AboutContent"
ContentPlaceHolderID="MainContentPlaceHolder" Runat="Server">
```

▶ Task 3: Add a reference to the Ajax Control Toolkit assembly

- Add a reference to the **C:\Program Files\ASP.NET Ajax Library \WebForms\Debug\AjaxControlToolkit.dll** assembly.

▶ Task 4: Register the Ajax Toolkit assembly, namespace, and TagPrefix

- Add a **Register** directive to the **web.config** file by using the following attribute values:
  - assembly: **AjaxControlToolkit**
  - namespace: **AjaxControlToolkit**
  - tagPrefix: **ajaxToolKit**

```
<pages>
    <controls>
        <add assembly="AjaxControlToolkit"
namespace="AjaxControlToolkit" tagPrefix="ajaxToolKit" />
    </controls>
</pages>
```

- Save and close the web.config file.

▶ Task 5: Change the ScriptManager control in the master page

- Open the **Site.master** master page in Source view.
- Locate the **ScriptManager** control.

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

- Change the **ScriptManager** control to a **ToolkitScriptManager**.

```
<ajaxToolKit:ToolkitScriptManager ID="MainScriptManager"
runat="server" />
```

- Save the **Site.master** master page.
- Close the Site.master master page.

▶ Task 6: Add the ScriptManagerProxy control

- Add a **ScriptManagerProxy** control named **AboutScriptManagerProxy**, to the **AboutContent** control.

```
<asp:ScriptManagerProxy runat="server"
ID="AboutScriptManagerProxy" />
```

▶ Task 7: Add the HTML and UI controls

- Append a **Panel** control named **AboutPanel**, to the **Content** control. The panel can be made invisible by using an inline style of **display: none**.

```
<asp:Panel ID="AboutPanel" runat="server" Style="display: none">
</asp:Panel>
```

- Add an **UpdatePanel** control named **AboutUpdatePanel**, with an empty **ContentTemplate** element, to the **AboutPanel** control.

```
<asp:UpdatePanel ID="AboutUpdatePanel" runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

- Add a **Panel** control named **AboutPopupPanel**, with a cascading style sheet (CSS) class of **aboutPopupPanel**, to the **ContentTemplate** element in the **AboutUpdatePanel** control.

```
<asp:Panel runat="server" ID="AboutPopupPanel"
CssClass="aboutPopupPanel">
</asp:Panel>
```

- Add the following HTML tags and text within the **AboutPopupPanel** control.

```
<br />
 <b>About Contoso Customer Management</b> <br />
<br />
 Copyright © 2010 by Contoso <br />
<br />
```

- Append a **Button** control named **ShowModalButton**, with a CSS class of **displayNone**, to the **AboutPopupPanel** control.

```
<asp:Button ID="ShowModalButton" runat="server"
CssClass="displayNone" />
```

- Append a **ModalPopupExtender** control named **AboutModalPopupExtender**, with the following attribute values to the **AboutPopupPanel** control:
  - TargetControlID: **ShowModalButton**
  - PopupControlID: **AboutPanel**
  - CancelControlID: **CloseButton**
  - BackgroundCssClass: **modalPopup**

```
<ajaxToolKit:ModalPopupExtender ID="AboutModalPopupExtender"
runat="server" TargetControlID="ShowModalButton"
    PopupControlID="AboutPanel" CancelControlID="CloseButton"
BackgroundCssClass="modalPopup" />
```

- Append a **Button** control named **CloseButton**, which does not trigger validation, set the **OnClick** event to **CloseButton_Click**, set the **Text** property to **Close**, and add two HTML line break elements to the **AboutPopupPanel** control.

```
<asp:Button ID="CloseButton" runat="server"
CausesValidation="False" OnClick="CloseButton_Click" Text="Close"
/>
<br />
<br />
```

- Add the following styles to the Styles/Site.css stylesheet.

```
div.aboutPopupPanel
{
    background-color: White;
    text-align: center;
    border: 2px solid Black;
}
.modalPopup
{
    position: relative;
    background-color: Gray;
    -ms-filter: alpha(opacity=70);
    z-index: 1;
}
.displayNone
{
    display: none;
}
```

- Save and close the **Site.css** file.

▶ Task 8: Add the event handler and code

- View the code-behind file of the **About.aspx** Web Form.
- Show the **AboutModalPopupExtender** control in the **Page.Load** event.

```
Protected Sub CloseButton_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles CloseButton.Click
    ' Close about box
    AboutModalPopupExtender.Hide()
End Sub
```

- Hide the **AboutModalPopupExtender** control in the **CloseButton_Click** event handler.

```
Protected Sub CloseButton_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles CloseButton.Click
    ' Close about box
    AboutModalPopupExtender.Hide()
End Sub
```

▶ Task 9: Map navigation to the About Web Form

- Open the **web.sitemap** file.
- Add an **About** menu command to the **Help** menu, making it use the **About** Web Form, by adding a new siteMapNode within the **Help** siteMapNode.

```
<siteMapNode title="About" description="" url="~/About.aspx" />
```

- Save and close the **web.sitemap** file.

▶ Task 10: Test the modal About box

- Build the solution.
- Run the CustomerManagement Web application.
- Close the **About** box.
- Show the **About** box, using the menu.
- Close the **About** box.
- Close the CustomerManagement Web application.
- Close the About.aspx.vb code file.
- Close the About Web Form.

Results: After completing this exercise, you will have added a new Web Form content page, and included a **ScriptManagerProxy**, two Panel controls, one **UpdatePanel** control, and a **ModalPopupExtender** control to the Web Form. In addition, you will have added event handlers and code to create a modal **About** box, and then tested the **About** box.

Exercise 2: Customizing Dynamic Data Field Templates with Ajax Server Controls

The main tasks for this exercise are as follows:

1. Add the **CalendarExtender** control to the DateTime_Edit field template.
2. Test the DateTime_Edit field template.

▶ Task 1: Add the CalendarExtender control to the DateTime_Edit field template

- Open the **DateTime_Edit.ascx** user control Dynamic Data field template in the Source view.
- Add a **CalendarExtender** control named **DateTimeEditCalendarExtender**, to the user control by using the following attribute values:
  - TargetControlID: **TextBox1**
  - PopupPosition: **Right**

```
<ajaxToolKit:CalendarExtender
ID="DateTimeEditCalendarExtender" runat="server"
    TargetControlID="TextBox1" PopupPosition="Right" />
```

- Save and close the **DateTime_Edit.ascx** user control.

▶ Task 2: Test the DateTime_Edit field template

- Build the solution.
- Run the CustomerManagement Web application.
- Test the **CalendarExtender** control by adding a new entry to the **Customers** table, click the **ModifiedDate** box, and then select a date from the calendar.
- Cancel the new customer entry.
- Close the CustomerManagement Web application.

Results: After completing this exercise, you will have added a **CalendarExtender** control to the dynamic data **DateTime_Edit** field template user control, and tested the template.

Exercise 3: Adding the Country Import Progress Indicator

The main tasks for this exercise are as follows:

1. Add the **ScriptManagerProxy** control.
2. Add the HTML and UI controls.
3. Test the update progress.

▶ Task 1: Add the ScriptManagerProxy control

- Open the **ImportCountries** Web Form in the Source view.
- Rename the **Content** control from **Content1** to **ImportCountriesContent**.
- Add a **ScriptManagerProxy** control named **ImportCountriesScriptManagerProxy**, to the **Content** control.

```
<asp:ScriptManagerProxy runat="server"
ID="ImportCountriesScriptManagerProxy" />
```

▶ Task 2: Add the HTML and UI controls

- Add an **UpdatePanel** control named **ImportCountriesUpdatePanel**, to the **ImportCountriesContent** control, with an empty **ContentTemplate** element, after the **ImportCountriesScriptManagerProxy** control.

```
<asp:UpdatePanel ID="ImportCountriesUpdatePanel" runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

- Move the existing content of the **ImportCountriesContent** control to the **ContentTemplate** element.
- Format the document markup.
- Add an **UpdateProgress** control named **ImportCountriesUpdateProgress**, to the **ImportCountriesContent** control, after the **ImportCountriesScriptManagerProxy** control, by using the following attribute values:
  - AssociatedUpdatePanelID: **ImportCountriesUpdatePanel**
  - DisplayAfter: **0**

- Add an opening and a closing **ProgressTemplate** tag with the text, "**Processing...**" to the **ImportCountriesUpdateProgress** control.
- Save and close the **ImportCountries.aspx** Web Form.

▶ Task 3: Test the update progress

- Build the solution.
- Debug the application.
- Filter the countries

Note: Notice that the text, "Processing...", displays in the upper-left corner of the Web site for a short period of time.

▶ Task 4: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

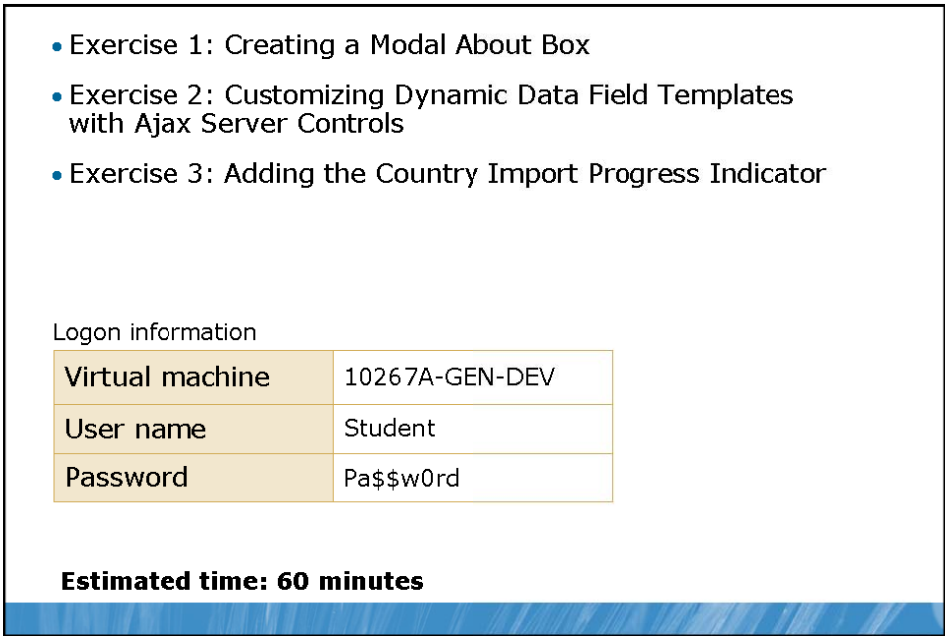
Results: After completing this exercise, you will have added a **ScriptManagerProxy**, an **UpdatePanel**, and an **UpdateProgress** control, and tested the update progress of the **ImportCountries** Web form.

Module 11

Lab Instructions: Creating a Microsoft® ASP.NET Ajax-enabled Web Forms Application (Visual C#)

Contents:

Lab: Creating a Microsoft ASP.NET Ajax-Enabled Web Forms Application



Note: You can perform tasks in this lab either by using the Microsoft Visual Basic® or Microsoft Visual C#® programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab document. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will implement partial-page updates by using ASP.NET Ajax, create a modal dialog, and customize Dynamic Data field templates by using Ajax server controls. In addition, you will display progress when updating a page with partial rendering.

Objectives

After completing this lab, you will be able to:

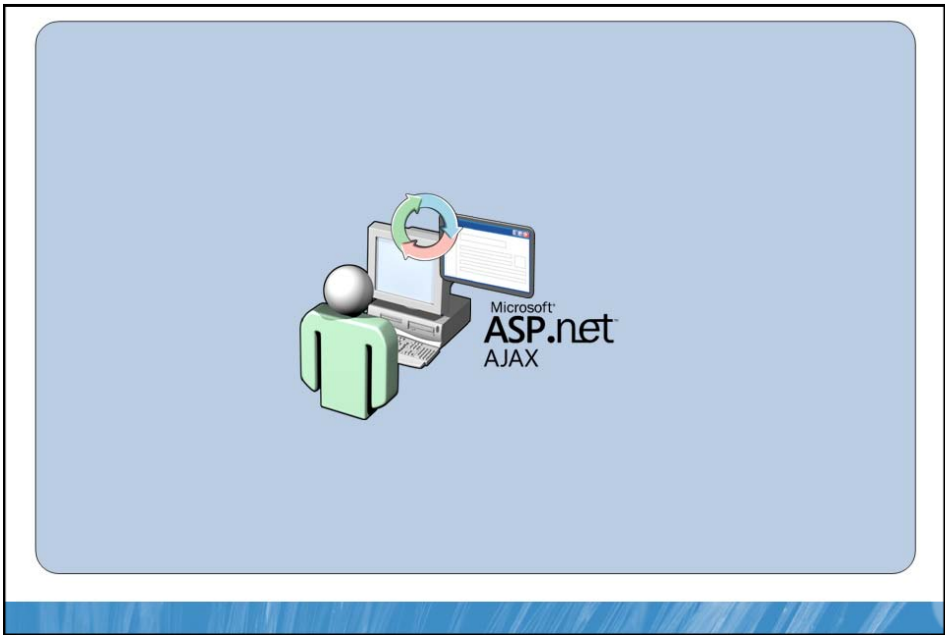
- Implement partial-page updates by using ASP.NET Ajax.
- Display a modal dialog.
- Display a client-side controlled Calendar.
- Display progress when updating a page with partial rendering.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses.NET applications to create, customize, and manage its customer information.

You are assigned the task of improving the responsiveness of the Web application user interface. You also need to avoid full-page updates where possible, and display a message when performing a partial update. For this purpose, you need to implement ASP.NET Ajax in the Customer Management application. You also need to ensure that partial page updates are possible without writing any client-side coding. In addition, you need to create a modal **About** box, add a client-side controlled **Calendar** control to a Dynamic Data field template user control, and add a progress indicator to the ImportCountries Web Form.

Section 2: Visual C#

Exercise 1: Creating a Modal About Box

The main tasks for this exercise are as follows:

1. Open an existing Web site.
2. Add a new Web Form.
3. Add a reference to the Ajax Control Toolkit assembly.
4. Register the Ajax Toolkit assembly, namespace, and TagPrefix.
5. Change the **ScriptManager** control in the master page.
6. Add the **ScriptManagerProxy** control.
7. Add the HTML and UI controls.
8. Add the event handler and code.
9. Map navigation to the About Web Form.
10. Test the modal About box.

▶ Task 1: Open an existing Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M11\CS folder.

▶ Task 2: Add a new Web Form

- Add a new Web Form named **About.aspx**, based on the master page, **Site.master**.
- Change the **ID** property of the **Content** control to **AboutContent**.

```
<asp:Content ID="AboutContent"
ContentPlaceHolderID="MainContentPlaceHolder" Runat="Server">
```

▶ Task 3: Add a reference to the Ajax Control Toolkit assembly

- Add a reference to the **C:\Program Files\ASP.NET Ajax Library \WebForms\Debug\AjaxControlToolkit.dll** assembly.

▶ Task 4: Register the Ajax Toolkit assembly, namespace, and TagPrefix

- Add a **Register** directive to the **web.config** file by using the following attribute values:
  - assembly: **AjaxControlToolkit**
  - namespace: **AjaxControlToolkit**
  - tagPrefix: **ajaxToolKit**

```
<pages>
    <controls>
        <add assembly="AjaxControlToolkit"
namespace="AjaxControlToolkit" tagPrefix="ajaxToolKit" />
    </controls>
</pages>
```

- Save and close the web.config file.

▶ Task 5: Change the ScriptManager control in the master page

- Open the **Site.master** master page in Source view.
- Locate the **ScriptManager** control.

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

- Change the **ScriptManager** control to a **ToolkitScriptManager**.

```
<ajaxToolKit:ToolkitScriptManager ID="MainScriptManager"
runat="server" />
```

- Save the **Site.master** master page.
- Close the Site.master master page.

▶ Task 6: Add the ScriptManagerProxy control

- Add a **ScriptManagerProxy** control named **AboutScriptManagerProxy**, to the **AboutContent** control.

```
<asp:ScriptManagerProxy runat="server"
ID="AboutScriptManagerProxy" />
```

▶ Task 7: Add the HTML and UI controls

- Append a **Panel** control named **AboutPanel**, to the **Content** control. The panel can be made invisible by using an inline style of **display: none**.

```
<asp:Panel ID="AboutPanel" runat="server" Style="display: none">
</asp:Panel>
```

- Add an **UpdatePanel** control named **AboutUpdatePanel**, with an empty **ContentTemplate** element to the **AboutPanel** control.

```
<asp:UpdatePanel ID="AboutUpdatePanel" runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

- Add a **Panel** control named **AboutPopupPanel**, with a cascading style sheet (CSS) class of **aboutPopupPanel**, to the **ContentTemplate** element in the **AboutUpdatePanel** control.

```
<asp:Panel runat="server" ID="AboutPopupPanel"
CssClass="aboutPopupPanel">
</asp:Panel>
```

- Add the following HTML tags and text within the **AboutPopupPanel** control.

```
<br />
 <b>About Contoso Customer Management</b> <br />
<br />
 Copyright © 2010 by Contoso <br />
<br />
```

- Append a **Button** control named **ShowModalButton**, with a CSS class of **displayNone** to the **AboutPopupPanel** control.

```
<asp:Button ID="ShowModalButton" runat="server"
CssClass="displayNone" />
```

- Append a **ModalPopupExtender** control named **AboutModalPopupExtender**, with the following attribute values to the **AboutPopupPanel** control:
  - TargetControlID: **ShowModalButton**
  - PopupControlID: **AboutPanel**
  - CancelControlID: **CloseButton**
  - BackgroundCssClass: **modalPopup**

```
<ajaxToolKit:ModalPopupExtender ID="AboutModalPopupExtender"
runat="server" TargetControlID="ShowModalButton"
    PopupControlID="AboutPanel" CancelControlID="CloseButton"
BackgroundCssClass="modalPopup" />
```

- Append a **Button** control named **CloseButton**, which does not trigger validation and with the **OnClick** event set to **CloseButton_Click** and the **Text** property set to **Close**, and two HTML line break elements to the **AboutPopupPanel** control.

```
<asp:Button ID="CloseButton" runat="server"
CausesValidation="False" OnClick="CloseButton_Click" Text="Close"
/>
<br />
<br />
```

- Add the following styles to the Styles/Site.css stylesheet.

```
div.aboutPopupPanel
{
    background-color: White;
    text-align: center;
    border: 2px solid Black;
}
.modalPopup
{
    position: relative;
    background-color: Gray;
    -ms-filter: alpha(opacity=70);
    z-index: 1;
}
.displayNone
{
    display: none;
}
```

- Save and close the **Site.css** file.

▶ Task 8: Add the event handler and code

- View the code-behind file of the **About.aspx** Web Form.
- Show the **AboutModalPopupExtender** control in the **Page.Load** event.

```
// Show about box
AboutModalPopupExtender.Show();
```

- Hide the **AboutModalPopupExtender** control in the **CloseButton_Click** event handler.

```
protected void CloseButton_Click(object sender, EventArgs e)
{
    // Close about box
    AboutModalPopupExtender.Hide();
}
```

▶ Task 9: Map navigation to the About Web Form

- Open the web.sitemap file.
- Add an **About** menu command to the **Help** menu, making it use the **About** Web Form, by adding a new siteMapNode within the **Help** siteMapNode.

```
<siteMapNode title="About" description="" url="~/About.aspx" />
```

- Save and close the web.sitemap file.

▶ Task 10: Test the modal About box

- Build the solution.
- Run the CustomerManagement Web application.
- Close the **About** box.
- Show the **About** box using the menu.
- Close the **About** box.
- Close the CustomerManagement Web application.
- Close the About.aspx.cs code file.
- Close the About Web Form.

Results: After completing this exercise, you will have added a new Web Form content page, and included a **ScriptManagerProxy**, two **Panel** controls, one **UpdatePanel** control, and a **ModalPopupExtender** control to the Web Form. In addition, you will have added event handlers and code to create a modal **About** box, and then tested the **About** box.

Exercise 2: Customizing Dynamic Data Field Templates with Ajax Server Controls

The main tasks for this exercise are as follows:

1. Add the CalendarExtender control to the DateTime_Edit field template.
2. Test the DateTime_Edit field template.

▶ Task 1: Add the CalendarExtender control to the DateTime_Edit field template

- Open the **DateTime_Edit.ascx** user control Dynamic Data field template in the Source view.
- Add a **CalendarExtender** control named **DateTimeEditCalendarExtender**, to the user control by using the following attribute values:
  - TargetControlID: **TextBox1**
  - PopupPosition: **Right**

```
<ajaxToolKit:CalendarExtender
ID="DateTimeEditCalendarExtender" runat="server"
    TargetControlID="TextBox1" PopupPosition="Right" />
```

- Save and close the **DateTime_Edit.ascx** user control.

▶ Task 2: Test the DateTime_Edit field template

- Build the solution.
- Run the CustomerManagement Web application.
- Test the **CalendarExtender** control by adding a new entry to the **Customers** table, click the **ModifiedDate** box, and then select a date from the calendar.
- Cancel the new customer entry.
- Close the CustomerManagement Web application.

Results: After completing this exercise, you will have added a **CalendarExtender** control to the dynamic data **DateTime_Edit** field template user control and tested the template.

Exercise 3: Adding the Country Import Progress Indicator

The main tasks for this exercise are as follows:

1. Add the ScriptManagerProxy control.
2. Add the HTML and UI controls.
3. Test the update progress.

▶ Task 1: Add the ScriptManagerProxy control

- Open the **ImportCountries** Web Form in the Source view.
- Rename the **Content** control from **Content1** to **ImportCountriesContent**.
- Add a **ScriptManagerProxy** control named **ImportCountriesScriptManagerProxy**, to the **Content** control.

```
<asp:ScriptManagerProxy runat="server"
ID="ImportCountriesScriptManagerProxy" />
```

▶ Task 2: Add the HTML and UI controls

- Add an **UpdatePanel** control named **ImportCountriesUpdatePanel**, to the **ImportCountriesContent** control, with an empty **ContentTemplate** element, after the **ImportCountriesScriptManagerProxy** control.

```
<asp:UpdatePanel ID="ImportCountriesUpdatePanel" runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

- Move the existing content of the **ImportCountriesContent** control to the **ContentTemplate** element.
- Format the document markup.
- Add an **UpdateProgress** control named **ImportCountriesUpdateProgress**, to the **ImportCountriesContent** control, after the **ImportCountriesScriptManagerProxy** control by using the following attribute values:
  - AssociatedUpdatePanelID: **ImportCountriesUpdatePanel**
  - DisplayAfter: **0**

- Add an opening and a closing **ProgressTemplate** tag with the text, "**Processing...**" to the **ImportCountriesUpdateProgress** control.
- Save and close the **ImportCountries.aspx** Web Form.

▶ Task 3: Test the update progress

- Build the solution.
- Debug the application.
- Filter the countries.

Note: Notice that the text, "Processing...", displays in the upper-left corner of the Web site for a short period of time.

▶ Task 4: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

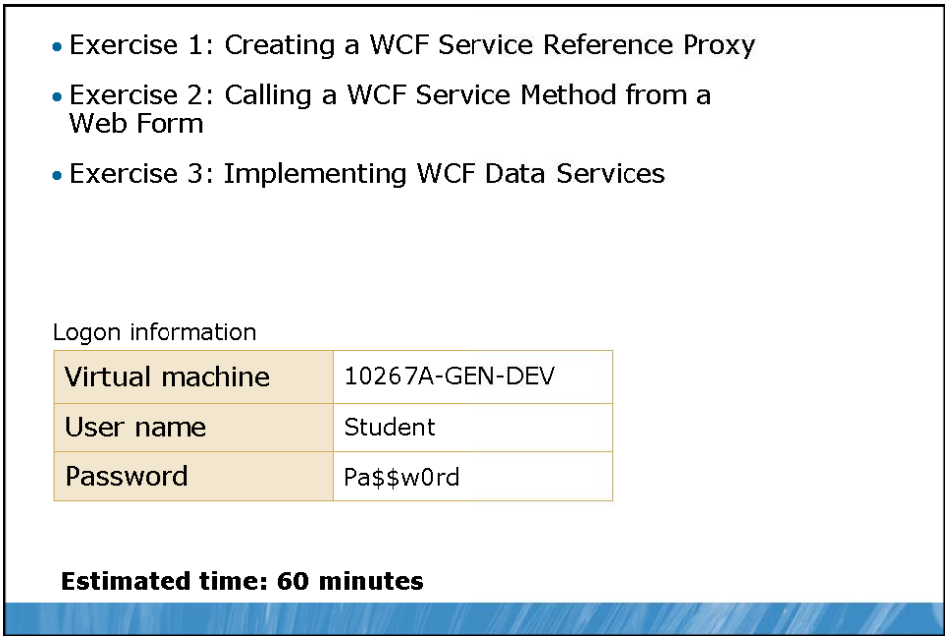
Results: After completing this exercise, you will have added a **ScriptManagerProxy**, an **UpdatePanel**, and an **UpdateProgress** control, and tested the update progress of the **ImportCountries** Web form.

Module 12

Lab Instructions: Consuming Microsoft® Windows Communication Foundation Services (Visual Basic)

Contents:

Lab: Consuming Windows Communication Foundation Services



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab page. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab page.

Introduction

In this lab, you will discover the WCF service at a specific address by using a .svc file. In addition, you will create a service reference proxy for the Customers WCF service, and call the WCF service method from a Web Form. Finally, you will implement and test a WCF Data Service.

Objectives

After completing this lab, you will be able to:

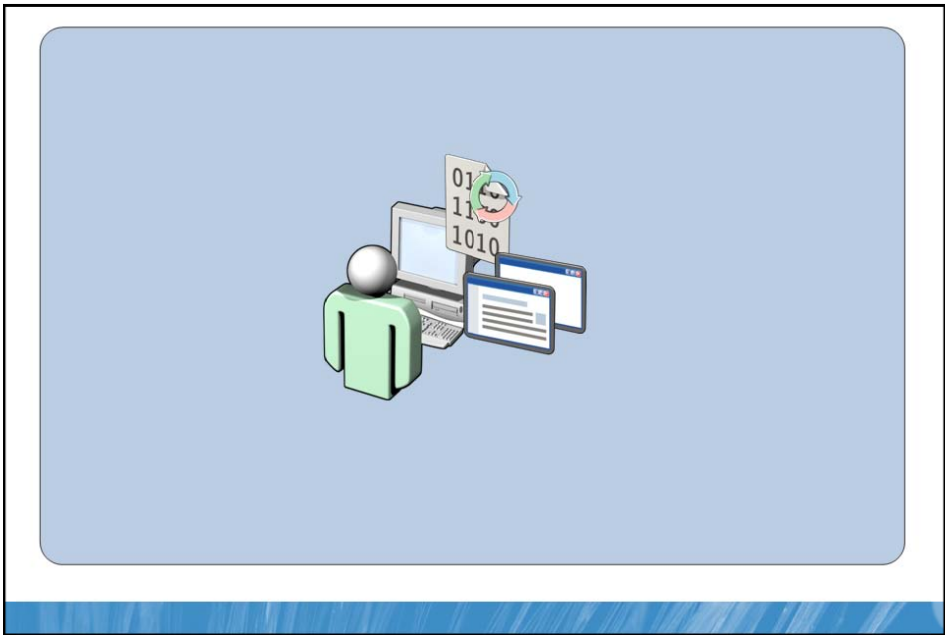
- Reuse existing functionality exposed by using WCF Services.
- Discover WCF Services.
- Create a Service reference proxy for a WCF service.
- Call the WCF Service method from a Web Form by using the Service reference proxy.
- Implement and test a WCF Data Service.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. You need to implement a Web Form that can be called by external customers to retrieve the information in your database. The senior developer has created a WCF Web service that exposes a method that returns the countries, and the WCF service method will be available to some customers. However, other customers need to see the country data on a Web page. Therefore, it is your responsibility to discover the WCF service from within your Web site, create a proxy object to call the WCF service method from a new Web Form, and display the returned countries. Finally, you need to implement and test a WCF Data Service.

Section 1: Visual Basic

Exercise 1: Creating a WCF Service Reference Proxy

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Discover the WCF services at a specific address by using a .svc file.
3. Examine the Web reference files.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M12\VB folder.

▶ Task 2: Discover the WCF services at a specific address by using a .svc file

- Open the **Add Service Reference** dialog box.
- Browse the WCF services at a specific address, **http://localhost:1112/Services/Customers.svc**.
- Open the Customers WCF services.
- View the operations of the **Customers** WCF service.

Note: Notice the single method, **GetCountries**, in the Operations pane.

- Specify the service reference name as **CustomersService**.

▶ Task 3: Examine the Web reference files

- Open the **Reference.svcmap** file, and examine the Web reference files.

Note: Notice that the **MetadataFile** elements and **ExtensionFile** elements point to the Customers.wsdl, Customers.xsd, Customers1.xsd, Customers.disco, and Customers2.xsd files, and configuration91.svcinfo and configuration.svcinfo files respectively.

- Close the Reference.svcmap file.
- Open the **Customers.disco** file, and examine the Web reference files.

Note: Notice the discovery file contains the **contractRef** element that links to other documents, including the WSDL, and documentation on the server.

- Close the Customers.disco file.
  - In the App_WebReferences/CustomersService/Customers.disco window, click the **Close** button.
- Open the **Customers.wsdl** file, and examine the Web reference files.
- In Solution Explorer, right-click **Customers.wsdl**, and then click **Open**.

Note: View the WSDL for the WCF service, its methods, and the bindings.

- Close the **Customers.wsdl** file.

Results: After completing this exercise, you will have discovered a WCF service and seen its methods, by using the WCF service file with the extension, .svc.

Exercise 2: Calling a WCF Service Method from a Web Form

The main tasks for this exercise are as follows:

1. Create the **Customers** Web Form.
2. Add controls for specifying countries to return.
3. Add a **GridView** control.
4. Call the WCF service.
5. Test the **Customers** Web Form.

▶ Task 1: Create the Customers Web Form

- Create a folder named **Services**.
- Add a Web Form named **Customers**, to the **Services** folder. The Web Form should not be based on a master page.

Note: The **Customers** Web Form should not be based on a master page.

- Set the page title to **Countries by Contoso**.
- Link the **Styles/Site.css** stylesheet to the **Customers** Web Form.

```
<head runat="server">
    <title>Countries by Contoso</title>
    <link href="~/Styles/Site.css" rel="stylesheet"
type="text/css" />
</head>
```

▶ Task 2: Add controls for specifying countries to return

- Add a **Label** control named **StartingLettersLabel**, to the **div** element of the **Customers** Web Form.
- Add a **TextBox** control named **StartingLettersTextBox**, to the **div** element of the **Customers** Web Form, for specifying the starting letters of the countries to return.

- Add a **RequiredFieldValidator** control named **StartingLettersRequiredFieldValidator**, for requiring input in the **StartingLettersTextBox** control.
- Add a **Button** control named **GetCountriesButton**, to the **div** element of the **Customers** Web Form. Create the **Click** event handler.

▶ Task 3: Add a GridView control

- Add a **GridView** control named **CustomersGridView**, to the **div** element of the **Customers** Web Form, using a style of **DDGridView**.
- Save the **Customers** Web Form.
- Close the Customers Web Form.

▶ Task 4: Call the WCF service

- Import the **CustomersService** namespace to the **Customers** Web Form.

```
Imports CustomersService
```

- In the **GetCountriesButton.Click** event, declare and instantiate an instance of the **CustomersClient** class, named **customersService**.

```
Dim customersService As New CustomersClient()
```

- Call the **GetCountries** WCF service method, passing the content of the **Text** property of the **StartingLettersTextBox TextBox** control, and assign the return value to the **DataSource** property of the **CustomersGridView** control.

```
CustomersGridView.DataSource =
customersService.GetCountries(StartingLettersTextBox.Text)
```

- Implement data binding for the **CustomersGridView** control.

```
CustomersGridView.DataBind()
```

- Save the code-behind file of the **Customers** Web Form.

▶ Task 5: Test the Customers Web Form

- View the **Customers** Web Form in the browser.
- Get all countries beginning with the letter **a**.
- Close the Internet Explorer browser.
- Close the **Customers** Web Form.

Result: After completing this exercise, you will have created a Customers Web Form, and added controls for specifying countries to return. Additionally, you will have added a **GridView** control, and called the WCF service. Finally, you will have tested the Customers Web Form.

Exercise 3: Implementing WCF Data Services

The main tasks for this exercise are as follows:

1. Add a WCF Data Service.
2. Test the Data Service.

▶ Task 1: Add a WCF Data Service

- Add a new WCF Data Service in the **Services** folder named **CustomersWcfDS**, by using the **Add New Item** dialog box and the **WCF Data Service** template.
- Add the existing EDM namespace and class name to the declaration of the generic **CustomerManagementModel.Entities**.

```
Public Class CustomersWcfDS
    Inherits DataService(Of CustomerManagementModel.Entities)
```

- Allow read access to the **Countries** and **Customers** entities of the EDM by using the **IDataServiceConfiguration.SetEntitySetAccessRule** method in the Shared **InitializeService** method of the **DataService** class.

```
config.SetEntitySetAccessRule("Countries",
EntitySetRights.AllRead)
config.SetEntitySetAccessRule("Customers",
EntitySetRights.AllRead)
```

- Save and close the **CustomersWcfDS** code-behind file.
- Move the **CustomersWcfDS** code-behind file from the **Services\App_Code** folder to the **App_Code** folder in the root folder.
- Delete the **App_Code** folder from the **Services** folder.

▶ Task 2: Test the Data Service

- Run the application.
- View the entities available with the Customers Data Service by browsing to the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc**.

Note: Notice that the XML that was returned from the service contains both the **Countries** and **Customers** entities.

- View the **Countries** entity by browsing to the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries**.

Note: Notice that the XML that was returned from the service contains all of the Countries from the data model.

- View the **Customers** entity by browsing to the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Customers**.

Note: Notice the XML that was returned from the service contains all of the Customers from the data model.

- View the information for the country, **India**, by browsing to the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=Name eq 'India'**.

Note: Notice the XML that was returned from the service contains a single country, **India**.

- View the countries for which the **InternetTLD** starts with less than **B** by using the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=InternetTLD lt 'B'**.

Note: Notice the XML that was returned from the service contains all countries for which the **InternetTLD** starts with less than B.

- Close Internet Explorer.

Task 3: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

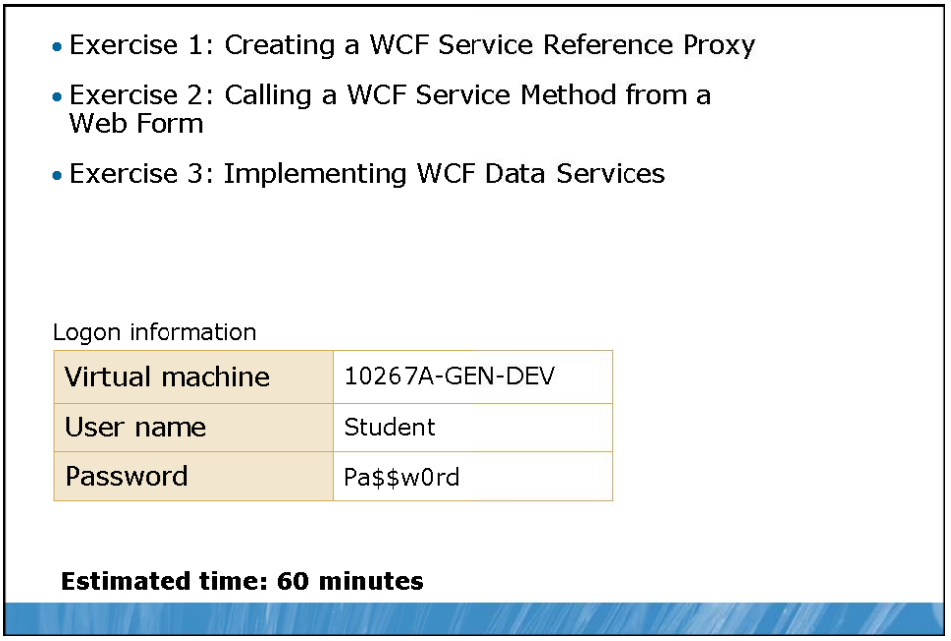
Results: After completing this exercise, you will have added and tested a WCF Data Service.

Module 12

Lab Instructions: Consuming Microsoft® Windows Communication Foundation Services (Visual C#)

Contents:

Lab: Consuming Windows Communication Foundation Services



Note: You can perform tasks in this lab either by using the Visual Basic or Visual C# programming language. If you are using Visual Basic as your programming language, refer to the steps provided in the Section 1 of the lab page. If you are using Visual C# as your programming language, refer to the steps provided in Section 2 of the lab page.

Introduction

In this lab, you will discover the WCF service at a specific address by using a .svc file. In addition, you will create a service reference proxy for the Customers WCF service, and call the WCF service method from a Web Form. Finally, you will implement and test a WCF Data Service.

Objectives

After completing this lab, you will be able to:

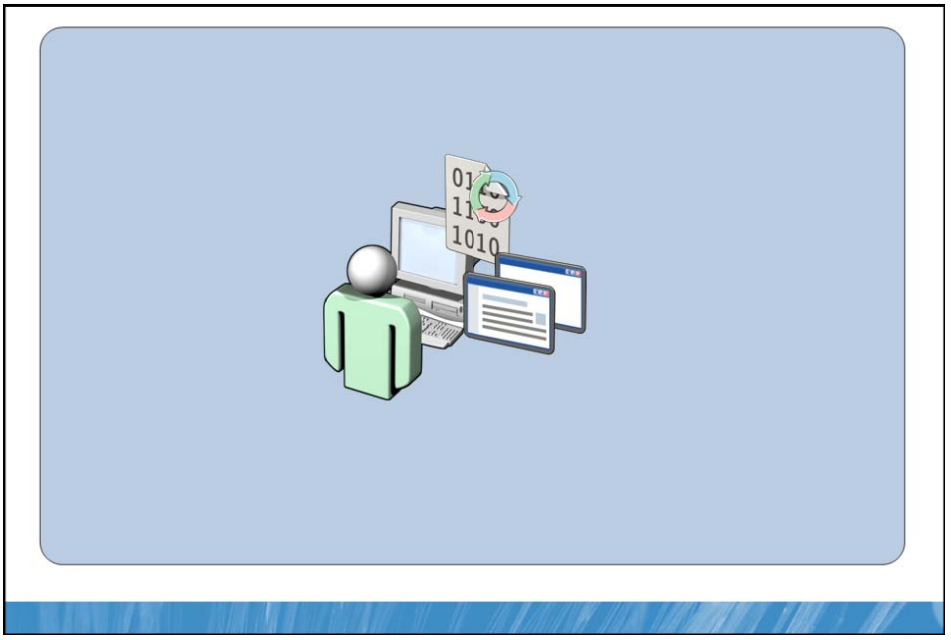
- Reuse existing functionality exposed by using WCF Services.
- Discover WCF Services.
- Create a Service reference proxy for a WCF service.
- Call the WCF Service method from a Web Form by using the Service reference proxy.
- Implement and test a WCF Data Service.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. You need to implement a Web Form that can be called by external customers to retrieve the information in your database. The senior developer has created a WCF Web service that exposes a method that returns the countries, and the WCF service method will be available to some customers. However, other customers need to see the country data on a Web page. Therefore, it is your responsibility to discover the WCF service from within your Web site, create a proxy object to call the WCF service method from a new Web Form, and display the returned countries. Finally, you need to implement and test a WCF Data Service.

Section 2: Visual C#

Exercise 1: Creating a WCF Service Reference Proxy

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Discover the WCF services at a specific address by using a .svc file.
3. Examine the Web reference files.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M12\CS folder.

▶ Task 2: Discover the WCF services at a specific address by using a .svc file

- Open the **Add Service Reference** dialog box.
- Browse the WCF services at **http://localhost:1112/Services/Customers.svc**.
- Open the **Customers** WCF services.
- View the operations of the **Customers** WCF service.

Note: Notice the single method, **GetCountries**, in the Operations pane.

- Specify the service reference name as **CustomersService**.

▶ Task 3: Examine the Web reference files

- Open the **Reference.svcmap** file and examine the Web reference files.

Note: Notice that the MetadataFile elements, and ExtensionFile elements, point to the Customers.wsdl, Customers.xsd, Customers1.xsd, Customers.disco, and Customers2.xsd files, and configuration91.svcinfo and configuration.svcinfo files respectively.

- Close the **Reference.svcmap** file.
- Open the **Customers.disco** file, and examine the Web reference files.

Note: Notice the discovery file contains the **contractRef** element that link to other documents, including the WSDL and documentation on the server.

- Close the **Customers.disco** file.
  - In the App_WebReferences/CustomersService/Customers.disco window, click the **Close** button.
- Open the **Customers.wsdl** file, and examine the Web reference files.
  - In Solution Explorer, right-click **Customers.wsdl**, and then click **Open**.

Note: View the WSDL for the WCF service, its methods, and the bindings.

- Close the **Customers.wsdl** file.

Results: After completing this exercise, you will have discovered a WCF service and viewed its methods, by using the WCF service file with the extension, .svc.

Exercise 2: Calling a WCF Service Method from a Web Form

The main tasks for this exercise are as follows:

1. Create the **Customers** Web Form.
2. Add controls for specifying countries to return.
3. Add a **GridView** control.
4. Call the WCF service.
5. Test the **Customers** Web Form.

▶ Task 1: Create the Customers Web Form

- Create a folder named **Services**.
- Add a Web Form named **Customers**, to the **Services** folder. The Web Form should not be based on a master page.

Note: The Customers Web Form should not be based on a master page.

- Set the page title to **Countries by Contoso**.
- Link the **Styles/Site.css** stylesheet to the **Customers** Web Form.

```
<head runat="server">
    <title>Countries by Contoso</title>
    <link href="~/Styles/Site.css" rel="stylesheet"
type="text/css" />
</head>
```

▶ Task 2: Add controls for specifying countries to return

- Add a **Label** control named **StartingLettersLabel**, to the **div** element of the **Customers** Web Form.
- Add a **TextBox** control named **StartingLettersTextBox**, to the **div** element of the **Customers** Web Form, for specifying the starting letters for the countries to return.

- Add a **RequiredFieldValidator** control named **StartingLettersRequiredFieldValidator**, for requiring input in the **StartingLettersTextBox** control.
- Add a **Button** control named **GetCountriesButton**, to the **div** element of the **Customers** Web Form. Create the **Click** event handler.

▶ Task 3: Add a GridView control

- Add a **GridView** control named **CustomersGridView**, using a style of **DDGridView**, to the **div** element of the **Customers** Web Form.
- Save the **Customers** Web Form.
- Close the Customers Web Form.

▶ Task 4: Call the WCF service

- Import the **CustomersService** namespace to the **Customers** Web Form.

```
using CustomersService;
```

- In the **GetCountriesButton.Click** event, declare and instantiate an instance of the **CustomersClient** class, named **customersService**.

```
CustomersClient customersService = new CustomersClient();
```

- Call the **GetCountries** WCF service method, passing the content of the **Text** property of the **StartingLettersTextBox TextBox** control, and assign the return value to the **DataSource** property of the **CustomersGridView** control.

```
CustomersGridView.DataSource =
customersService.GetCountries(StartingLettersTextBox.Text);
```

- Implement data binding for the **CustomersGridView** control.

```
CustomersGridView.DataBind();
```

- Save the code-behind file of the **Customers** Web Form.

▶ Task 5: Test the Customers Web Form

- View the **Customers** Web Form in the browser.
- Get all countries beginning with the letter **a**.
- Close the Internet Explorer browser.
- Close the Customers Web Form.

Result: After completing this exercise, you will have created a Customers Web Form, and added controls for specifying countries to return. Additionally, you will have added a **GridView** control, and called the WCF service. Finally, you will have tested the Customers Web Form.

Exercise 3: Implementing WCF Data Services

The main tasks for this exercise are as follows:

1. Add a WCF Data Service.
2. Test the Data Service.

▶ Task 1: Add a WCF Data Service

- Add a new WCF Data Service in the **Services** folder named **CustomersWcfDS**, by using the **Add New Item** dialog box and the **WCF Data Service** template.
- Add the existing EDM namespace and class name to the declaration of the generic **CustomerManagementModel.Entities**.

```
public class Customers :
DataService<CustomerManagementModel.Entities>
```

- Allow read access to the **Countries** and **Customers** entities of the EDM by using the **IDataServiceConfiguration.SetEntitySetAccessRule** method in the Shared **InitializeService** method of the **DataService** class.

```
config.SetEntitySetAccessRule("Countries",
EntitySetRights.AllRead);
config.SetEntitySetAccessRule("Customers",
EntitySetRights.AllRead);
```

- Save and close the **CustomersWcfDS** code-behind file.
- Move the **CustomersWcfDS** code-behind file from the **Services\App_Code** folder to the **App_Code** folder in the root folder.
- Delete the **App_Code** folder from the **Services** folder.

▶ Task 2: Test the Data Service

- Run the application.
- View the entities available with the Customers Data Service by browsing to the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc**.

Note: Notice that the XML returned from the service contains both the **Countries** and **Customers** entities.

- View the **Countries** entity by browsing to the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries**.

Note: Notice that the XML returned from the service contains all of the Countries from the data model.

- View the **Customers** entity by browsing to the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Customers**.

Note: Notice the XML returned from the service, it contains all of the Customers from the data model.

- View the information for the country, **India**, by browsing to the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=Name eq 'India'**.

Note: Notice the XML returned from the service, it contains a single country, **India**.

- View the countries for which the **InternetTLD** starts with less than **B** by using the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=InternetTLD lt 'B'**.

Note: Notice the XML returned from the service, it contains all countries for which the **InternetTLD** starts with less than **B**.

- Close Internet Explorer.

▶ Task 3: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

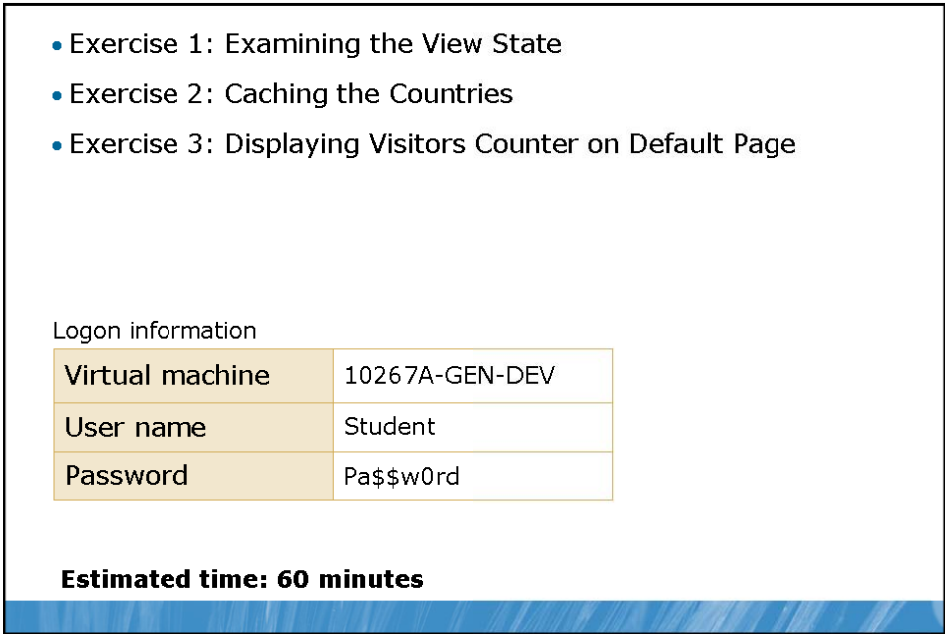
Results: After completing this exercise, you will have added and tested a WCF Data Service.

Module 13

Lab Instructions: Managing State in Web Applications (Visual Basic)

Contents:

Lab: Managing State in Web Applications



Note: You can perform tasks in this lab either by using Visual Basic or Visual C#. If you are using Visual Basic, refer to the steps provided in **Section 1** of the lab page. If you are using Visual C#, refer to the steps provided in **Section 2** of the lab page.

Objectives

After completing this lab, you will be able to:

- Examine how View state helps to maintain server control state.
- Cache an item and retrieve a cached item from the Cache object.
- Create an application variable and use it to maintain state.

Introduction

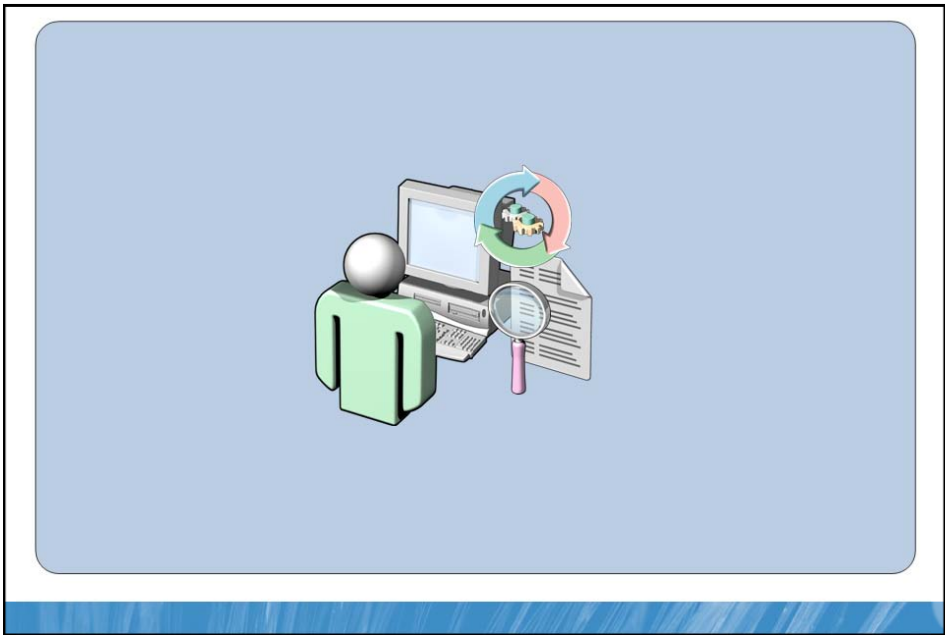
In this lab, you will examine how you use View state to maintain server control values between server round-trips, cache a DataTable object on a page request of the Customers Web Form, and retrieve the data on subsequent requests. You will also add a value to the Application state, use the value to track the number of visitors to the application, and display it on the Default Web Form.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. The senior developer wants you to understand the concept of state management, and why it is important to maintain state to provide a user-friendly experience. You need to understand how most of the server controls keep state between round trips. The senior developer has requested that you to look at how View state is used for this purpose to manage state for the CustomerManagement application. You also need to cache a DataTable object that is retrieved from a Web Service method, and then retrieve it again. Finally, you need to create visitors counter on the Default Web Form.

Section 1: Visual Basic

Exercise 1: Examining the View State

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Set the start page.
3. Notice the View state size.
4. Disable the View state.
5. Notice the View state size.
6. Enable the View state.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M13\VB folder.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.

▶ Task 3: Notice the View state size

- Run the application.
- View all customers.

Note: Notice that the **GridView** control displays with the customers in the database.

- View the page source by using the **View Source** option.

Note: You can also view the page source by using the **View Source** option on the **Page** menu.

Note: The built-in viewer of the Developer Tools in Windows® Internet Explorer® 8 opens in the http://localhost:1111/CustomerManagement/Customers/List.aspx – Original Source window.

- Locate the page View state in the hidden field named **__VIEWSTATE**, and view the space occupied by the **__VIEWSTATE** element.

Note: Notice that about 10–15 percent of the space of the http://localhost:1111 /CustomerManagement/Customers/List.aspx – Original Source window is occupied by the **__VIEWSTATE** element.

- Close the built-in viewer of the Internet Explorer 8 Developer Tools.
- Close Internet Explorer.

▶ Task 4: Disable the View state

- In Solution Explorer, navigate to **DynamicData\PageTemplates**, and then open the **List.aspx** Web Form in the Source view.
- Disable the View state for the **List** Web Form by using the **EnableViewState** page attribute.
- Save the **List** Web Form.

▶ Task 5: Notice the View state size

- Run the application.
- View all customers, and note that there is no change in the number of customers.

Note: Notice that the **GridView** control displays with the same number of customers.

- View the page source by using the **View Source** option.

Note: You can also view the page source by using the **View Source** option on the **Page** menu.

Note: The Developer Tools built-in viewer in Internet Explorer 8 opens in the http://localhost:1111/CustomerManagement/Customers/List.aspx – Original Source window.

- Locate the page View state in the hidden field named **__VIEWSTATE**, and view the space occupied by the **__VIEWSTATE** element.

Note: Notice that about 6–8 percent of the space of the http://localhost:1111 /CustomerManagement/Customers/List.aspx – Original Source window is occupied by the **__VIEWSTATE** element.

- Close the built-in viewer of the Internet Explorer 8 Developer Tools.
- Close Internet Explorer.

▶ Task 6: Enable the View state

- Enable the View state for the **List** Web Form by removing the **EnableViewState** page attribute.
- Save and close the **List** Web Form.

Results: After completing this exercise, you will have displayed the list view of all customers with View state enabled, disabled View state for the **List.aspx** Web Form, and then displayed the List view of all customers with View state disabled.

Exercise 2: Caching the Countries

The main tasks for this exercise are as follows:

1. Cache a DataTable.
2. Test the cache storage.

▶ Task 1: Cache a DataTable

- In the **Services_Customers** class, create a private member variable named **countryDataTable**, of type **System.Data.DataTable**, to hold the countries retrieved from the database and initialize the variable to **Nothing**.

```
Private countryDataTable As System.Data.DataTable = Nothing
```

- Assign the **Countries** cache item value to the **countryDataTable** private variable, casting it to the data type **System.Data.DataTable**, upon page load.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Retrieve DataTable from cache
    countryDataTable =
        CType(Cache("Countries"), System.Data.DataTable)
End Sub
```

- Check whether an item has been retrieved from the cache by comparing the **countryDataTable** variable with a null value, in an **If** construct, when the user clicks the **Get Countries** button.

```
' Does cached item exist?
If countryDataTable Is Nothing Then

End If
```

- Assign the value of the **countryDataTable** local variable to the **DataSource** property of the **CustomersGridView** control.

```
' Set GridView DataSource
CustomersGridView.DataSource = countryDataTable
```

- If an item is not retrieved from the cache, move the existing line of code from after the **If** construct that declares and instantiates the local **customersService** variable.

```
Dim customersService As New CustomersClient()
```

- If an item is not retrieved from the cache, assign the result of the call to the **GetCountries** WCF service method, to the local **countryDataTable** variable.

```
' Retrieve DataTable from WCF Service
countryDataTable =
customersService.GetCountries(StartingLettersTextBox.Text)
```

- Add a new variable named **Countries** to the cache object, and assign the value of the local **countryDataTable** variable.

```
' Save DataTable to cache
Cache("Countries") = countryDataTable
```

- Delete the existing line of code that assigns the result of the call to the **GetCountries** WCF service method to the **DataSource** property of the **CustomersGridView** control.

```
CustomersGridView.DataSource =
customersService.GetCountries(StartingLettersTextBox.Text)
```

- Save the changes to the Customers.aspx.vb file.

▶ Task 2: Test the cache storage

- Set a breakpoint on the line that retrieves the **Countries** item from the **Cache** object.

```
If countryDataTable Is Nothing Then
```

- Run the application in the debug mode.
- View the **Customers** Web Form by typing in the URL, **http://localhost:1111/CustomerManagement/Services/Customers.aspx** in the Address bar.
- Retrieve the countries starting with the letter **b**.

Note: The Customers Web Form code now opens in the debugger, and the current line of code will now check whether you have retrieved an item from the cache.

- Press F10 to go to the next statement.

Note: You have not retrieved an item from the cache, so you need to create and store an item in the cache.

- Press F5 to continue running the application.

Note: Notice that the countries now display on the Web page.

- Close Internet Explorer.
- Run the application in the debug mode.
- View the **Customers** Web Form in browser by using the URL **http://localhost:1111/CustomerManagement/Services/Customers.aspx**.
- Retrieve the countries starting with the letter **c**.

Note: The **Customers** Web Form code now opens in the debugger, and the current line of code will now check whether you have retrieved an item from the cache.

- Press F10 to go to the next statement.

Note: You have retrieved an item from the cache. Now, you need to assign the retrieved item to the CustomersGridView data source property.

- Press F5 to continue running the application.

Note: Notice that the countries display on the Web page.

- Close Internet Explorer.

- Remove the breakpoint from the code that retrieves the **Countries** item from the **Cache** object.

```
If countryDataTable Is Nothing Then
```

- Close the **Customers.aspx.vb** file.

Results: After completing this exercise, you will have retrieved a DataTable from a Web Service method, and saved the DataTable in the cache. You will also have added code to retrieve the cached item on subsequent page requests. In addition, you will have tested the Web Form to view the countries that are saved and retrieved in the cache.

Exercise 3: Displaying a Visitors Counter on the Default Page

The main tasks for this exercise are as follows:

1. Initialize an application variable.
2. Add a visitors counter to the default page.
3. Update the visitors counter on the new session.
4. Test the visitors counter.

▶ Task 1: Initialize an application variable

- Add a variable named **Visitors**, with a value of **0** to the Application state.

```
' Save application variable
Application("Visitors") = 0
```

- Save the Global.asax file.

▶ Task 2: Add a visitors counter to the default page

- Add a **div** element with a **class** attribute set to the value of **footer** to the **MainContent** control of the **Default** Web Form.

```
<div class="footer">
</div>
```

- Add a **Literal** control named **VisitorLiteral**, to the **div** element.

```
<asp:Literal ID="VisitorLiteral" runat="server" />
```

- Save and close the **Default** Web Form.

- Add the following style to the **Styles/Site.css** stylesheet.

```
div.footer
{
    background: #dbddff;
    text-align: center;
    height: 15px; /* Height of the footer */
    width: 100%;
    position: fixed;
    bottom: 0;
}
```

- Save and close the **Site.css** file.
- In the **Page_Load** event handler of the **Default** Web Form, create a local variable named **numVisitors**, of type **Long**, to hold the number of visitors and initialize the variable with a value of **0**.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    Dim numVisitors As Long = 0
End Sub
```

- In the **Page_Load** event handler, check if the **Visitors** Application variable exists, and if so, assign the value to **numVisitors** local variable, casting it to data type **long**.

```
' Check if Application variable exists
If Not Application("Visitors") Is Nothing Then
    numVisitors = Long.Parse(Application("Visitors").ToString())
End If
```

- In the **Page_Load** event handler, assign the text, "**Number of visitors:** ", and the value of the **numVisitors** variable, to the **Text** property of the **VisitorLiteral** control.

Note: The **Visitors** application variable must be converted by using the **Parse** method of the **Long** data type.

```
VisitorLiteral.Text = "Number of visitors: " &
numVisitors.ToString()
```

- Save and close the **Default.aspx.vb** file.

▶ Task 3: Update the visitors counter on the new session

- Each time a new user session is started, increment the **Visitors** application variable by **1**, by appending the following code to the **Session_Start** method in the Global.asax file.

```
' Increment Visitors counter
Application("Visitors") =
    Long.Parse(Application("Visitors").ToString()) + 1
```

Note: The **Visitors** application variable must be converted by using the **Parse** method of the data type.

- Save and close the Global.asax file.

▶ Task 4: Test the visitors counter

- View the **Default** Web Form in a browser.

Note: Notice that the Number of visitors is set to **1**.

- Close Internet Explorer.
- View the **Default** Web Form in a browser.

Note: Notice that the Number of visitors is set to **2**.

- Close Internet Explorer.

▶ Task 5: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

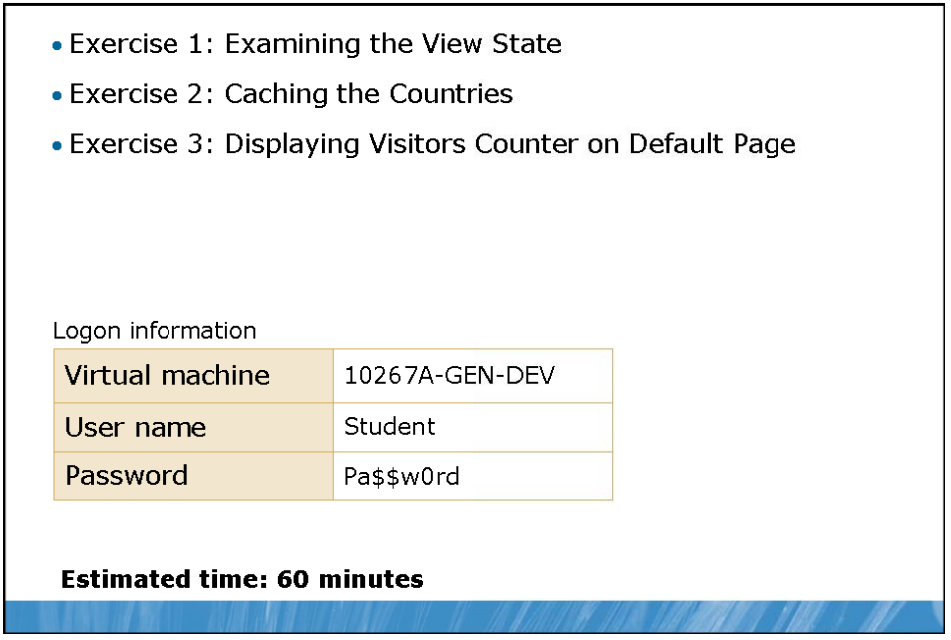
Results: After completing this exercise, you will have added and initialized a new Application variable, added visitors counter to the Default Web Form, and added code to increment the visitors counter on each new session. In addition, you should have tested the Web Form and verified the visitors counter.

Module 13

Lab Instructions: Managing State in Web Applications (Visual C#)

Contents:

Lab: Managing State in Web Applications



Note: You can perform tasks in this lab either by using Visual Basic or Visual C#. If you are using Visual Basic, refer to the steps provided in **Section 1** of the lab page. If you are using Visual C#, refer to the steps provided in **Section 2** of the lab page.

Objectives

After completing this lab, you will be able to:

- Examine how View state helps to maintain server control state.
- Cache an item and retrieve a cached item from the Cache object.
- Create an application variable and use it to maintain state.

Introduction

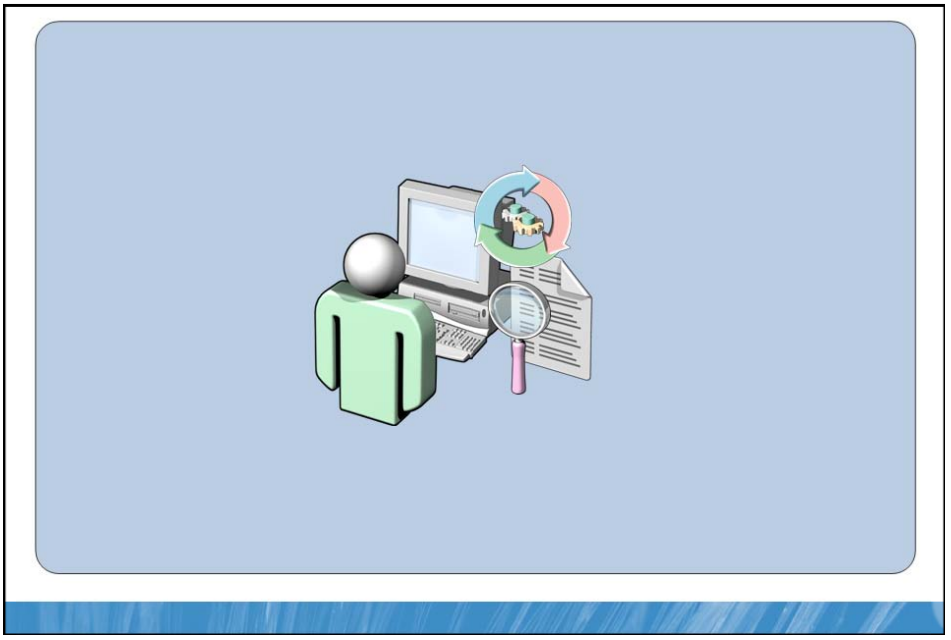
In this lab, you will examine how you use View state to maintain server control values between server round-trips, cache a DataTable object on a page request of the Customers Web Form, and retrieve the data on subsequent requests. You will also add a value to the Application state, use the value to track the number of visitors to the application, and display it on the Default Web Form.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Web site to manage its customer information. The senior developer wants you to understand the concept of state management, and why it is important to maintain state to provide a user-friendly experience. You need to understand how most of the server controls keep state between round trips. The senior developer has requested that you to look at how View state is used for this purpose to manage state for the CustomerManagement application. You also need to cache a DataTable object that is retrieved from a Web Service method, and then retrieve it again. Finally, you need to create visitors counter on the Default Web Form.

Section 2: Visual C#

Exercise 1: Examining the View State

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Set the start page.
3. Notice the View state size.
4. Disable the View state.
5. Notice the View state size.
6. Enable the View state.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M13\CS folder.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.

▶ Task 3: Notice the View state size

- Run the application.
- View all customers.

Note: Notice that the **GridView** control displays with the customers in the database.

- View the page source by using the **View Source** option.

Note: You can also view the page source by using the **View Source** option on the **Page** menu.

Note: The built-in viewer of the Developer Tools in Windows® Internet Explorer® 8 opens in the http://localhost:1111/CustomerManagement/Customers/List.aspx – Original Source window.

- Locate the page View state in the hidden field named **__VIEWSTATE**, and view the space occupied by the **__VIEWSTATE** element.

Note: Notice that about 10–15 percent of the space of the http://localhost:1111 /CustomerManagement/Customers/List.aspx – Original Source window is occupied by the **__VIEWSTATE** element.

- Close the built-in viewer of the Internet Explorer 8 Developer Tools.
- Close Internet Explorer.

▶ Task 4: Disable the View state

- In Solution Explorer, navigate to **DynamicData\PageTemplates**, and then open the **List.aspx** Web Form in the Source view.
- Disable the View state for the **List** Web Form by using the **EnableViewState** page attribute.
- Save the **List** Web Form.

▶ Task 5: Notice the View state size

- Run the application.
- View all customers and note that there is no change in the number of customers.

Note: Notice that the **GridView** control displays with the same number of customers.

- View the page source by using the **View Source** option.

Note: You can also view the page source by using the **View Source** option on the **Page** menu.

Note: The Developer Tools built-in viewer in Internet Explorer 8 opens in the http://localhost:1111/CustomerManagement/Customers/List.aspx – Original Source window.

- Locate the page View state in the hidden field named **__VIEWSTATE**, and view the space occupied by the **__VIEWSTATE** element.

Note: Notice that about 6–8 percent of the space of the http://localhost:1111 /CustomerManagement/Customers/List.aspx – Original Source window is occupied by the **__VIEWSTATE** element.

- Close the built-in viewer of the Internet Explorer 8 Developer Tools.
- Close Internet Explorer.

▶ Task 6: Enable the View state

- Enable the View state for the **List** Web Form by removing the **EnableViewState** page attribute.
- Save and close the **List** Web Form.

Results: After completing this exercise, you will have displayed the list view of all customers with View state enabled, disabled View state for the **List.aspx** Web Form, and then displayed the List view of all customers with View state disabled.

Exercise 2: Caching the Countries

The main tasks for this exercise are as follows:

1. Cache a DataTable.
2. Test the cache storage.

▶ Task 1: Cache a DataTable

- In the **Services_Customers** class, create a private member variable named **countryDataTable**, of type **System.Data.DataTable** to hold the countries retrieved from the database and initialize the variable to **null**.

```
private System.Data.DataTable countryDataTable = null;
```

- Assign the **Countries** cache item value to the **countryDataTable** private variable, casting it to the data type **System.Data.DataTable**, upon page load.

```
// Retrieve DataTable from cache
countryDataTable =
    (System.Data.DataTable) Cache["Countries"];
```

- Check whether an item has been retrieved from the cache by comparing the **countryDataTable** variable with a **null** value, in an **if** construct, when the user clicks the **Get Countries** button.

```
// Does cached item exist?
if (countryDataTable == null)
{

}
```

- Assign the value of the **countryDataTable** local variable to the **DataSource** property of the **CustomersGridView** control.

```
// Set GridView DataSource
CustomersGridView.DataSource = countryDataTable;
```

- If an item is not retrieved from the cache, move the existing line of code from after the **if** construct that declares and instantiates the local **customersService** variable.

```
CustomersClient customersService = new CustomersClient();
```

- If an item is not retrieved from the cache, assign the result of the call to the **GetCountries** WCF service method, to the local **countryDataTable** variable.

```
// Retrieve DataTable from WCF Service
countryDataTable =
customersService.GetCountries(StartingLettersTextBox.Text);
```

- Add a new variable named **Countries**, to the cache object, and assign the value of the local **countryDataTable** variable.

```
// Save DataTable to cache
Cache["Countries"] = countryDataTable;
```

- Delete the existing line of code that assigns the result of the call to the **GetCountries** WCF service method to the **DataSource** property of the **CustomersGridView** control.

```
CustomersGridView.DataSource =
customersService.GetCountries(StartingLettersTextBox.Text);
```

- Save the changes to the Customers.aspx.cs file.

▶ Task 2: Test the cache storage

- Set a breakpoint on the line that retrieves the **Countries** item from the **Cache** object.

```
if (countryDataTable == null)
```

- Run the application in the debug mode.
- View the **Customers** Web Form by typing in the URL **http://localhost:1110/CustomerManagement/Services/Customers.aspx** in the Address bar.
- Retrieve the countries starting with the letter **b**.

Note: The **Customers** Web Form code is now opened in the debugger, and the current line of code will now check whether you have retrieved an item from the cache.

- Press the F10 key to go to the next statement.

Note: You have not retrieved an item from the cache, so you need to create and store an item in the cache.

- Press the F5 key to continue running the application.

Note: Notice that the countries now display on the Web page.

- Close Internet Explorer.
- Run the application in the debug mode.
- View the **Customers** Web Form in browser, by using the URL, **http://localhost:1110/CustomerManagement/Services/Customers.aspx**.
- Retrieve the countries starting with the letter **c**.

Note: The **Customers** Web Form code is now opened in the debugger, and the current line of code will now check whether you have retrieved an item from the cache.

- Press the F10 key to go to the next statement.

Note: You have retrieved an item from the cache. Now, you need to assign the retrieved item to the CustomersGridView data source property.

- Press F5 to continue running the application.

Note: Notice that the countries display on the Web page.

- Close Internet Explorer.

- Remove the breakpoint from the code that retrieves the **Countries** item from the **Cache** object.

```
if (countryDataTable == null)
```

- Close the **Customers.aspx.cs** file.

Results: After completing this exercise, you will have retrieved a DataTable from a Web Service method, and saved the DataTable in the cache. You will also have added code to retrieve the cached item on subsequent page requests. In addition, you will have tested the Web Form to view the countries that are saved and retrieved in the cache.

Exercise 3: Displaying a Visitors Counter on the Default Page

The main tasks for this exercise are as follows:

1. Initialize an application variable.
2. Add a visitors counter to the default page.
3. Update the visitors counter on the new session.
4. Test the visitors counter.

▶ Task 1: Initialize an application variable

- Add a variable named **Visitors**, with a value of **0**, to the Application state.

```
// Save application variable
Application["Visitors"] = 0;
```

- Save the Global.asax file.

▶ Task 2: Add a visitors counter to the default page

- Add a **div** element with a **class** attribute set to the value of **footer** to the **MainContent** control of the **Default** Web Form.

```
<div class="footer">
</div>
```

- Add a **Literal** control named **VisitorLiteralVisitorLiteral**, to the **div** element.

```
<asp:Literal ID="VisitorLiteral" runat="server" />
```

- Save and close the **Default** Web Form.

- Add the following style to the **Styles/Site.css** stylesheet.

```
div.footer
{
    background: #dbddff;
    text-align: center;
    height: 15px; /* Height of the footer */
    width: 100%;
    position: fixed;
    bottom: 0;
}
```

- Save and close the **Site.css** file.
- In the **Page_Load** event handler of the **Default** Web Form, create a local variable named **numVisitors**, of type **long**, to hold the number of visitors and initialize the variable with a value of **0**.

```
long numVisitors = 0;
```

- In the **Page_Load** event handler, check if the **Visitors** Application variable exists, and if so, assign the value to **numVisitors** local variable, casting it to data type **long**.

```
// Check if Application variable exists
if (Application["Visitors"] != null)
{
    numVisitors = long.Parse(Application["Visitors"].ToString());
}
```

- In the **Page_Load** event handler, assign the text "**Number of visitors:** ", and the value of the **numVisitors** variable, to the **Text** property of the **VisitorLiteral** control.

Note: The **Visitors** application variable must be converted by using the Parse method of the long data type.

```
VisitorLiteral.Text = "Number of visitors: " +
numVisitors.ToString();
```

- Save and close the Default.aspx.cs file.

▶ Task 3: Update the visitors counter on the new session

- Each time a new user session is started, increment the **Visitors** application variable by **1**, by appending the following code to the **Session_Start** method in the Global.asax file.

```
// Increment Visitors counter
Application["Visitors"] =
    long.Parse(Application["Visitors"].ToString()) + 1;
```

Note: The **Visitors** application variable must be converted by using the **Parse** method of the data type.

- Save and close the **Global.asax** file.

▶ Task 4: Test the visitors counter

- View the **Default** Web Form in a browser.

Note: Notice that the Number of visitors is set to **1**.

- Close Internet Explorer.
- View the **Default** Web Form in a browser.

Note: Notice that the Number of visitors is set to **2**.

- Close Internet Explorer.

▶ Task 5: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

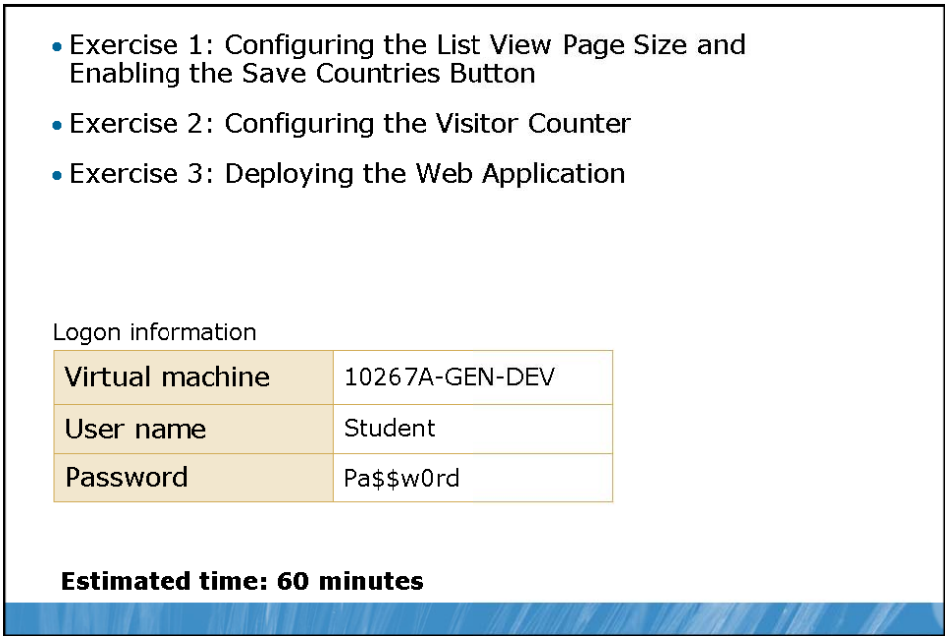
Results: After completing this exercise, you will have added and initialized a new Application variable, added visitors counter to the Default Web Form, and added code to increment the visitors counter on each new session. In addition, you will have tested the Web Form and verified the visitors counter.

Module 14

Lab Instructions: Configuring and Deploying a Microsoft® ASP.NET Web Application (Visual Basic)

Contents:

Lab: Configuring and Deploying a Microsoft ASP.NET Web Application



Note: You can perform tasks in this lab by using either Microsoft Visual Basic® or Microsoft Visual C#®. If you are using Visual Basic, refer to the steps provided in Section 1 of the lab document. If you are using Visual C#, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will configure an ASP.NET Web application by using the web.config file. In addition, you will deploy and publish the Web application to a Web site.

Objectives

After completing this lab, you will be able to:

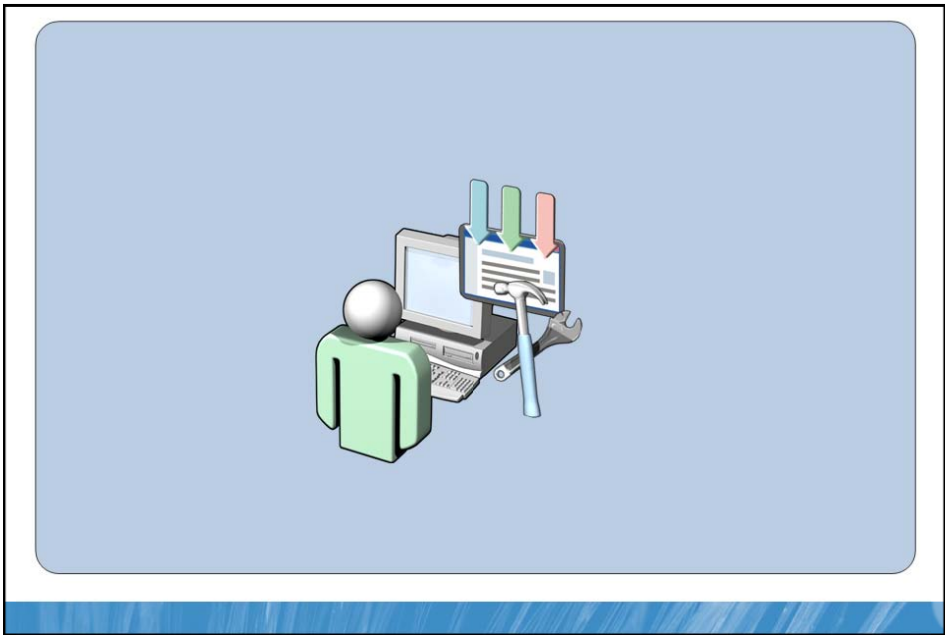
- Configure an ASP.NET Web application by using the web.config file.
- Deploy and publish an ASP.NET Web application to a Web site.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Customer Management application to create, customize, and manage its customer information. After configuring the Customer Management application, you need to deploy the Web site to the local IIS to help the sales team in different branches of Contoso, Ltd access the application. Before deploying the application to provide access for the branch offices, you need to test the application on the server. To do this, you need to save both static and dynamic values in the web.config file, because the web.config file accompanies the Customer Management application during deployment. In addition, you need to deploy the Web application to IIS.

Section 1: Visual Basic

Exercise 1: Configuring the List View Page Size and Enabling the Save Countries Button

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Set the start page.
3. View the default page size in the List view.
4. Set the default List view page size.
5. View the new page size in the List view.
6. View the Country Import Page.
7. Set the default value for enabling saving of the filtered countries.
8. View the Country Import Page.
9. Set the default value for enabling saving of the filtered countries.
10. View the Country Import Page.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft® Visual Studio® 2010 as an Administrator.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M14\VB folder.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.

▶ Task 3: View the default page size in the List view

- Run the application.
- View all the countries.

Note: Notice that by default, 10 items are displayed in the table. This is because the page size is set to 10.

- Close Windows® Internet Explorer®.

▶ Task 4: Set the default List view page size

- Open the **web.config** file, and add the **appSettings** element after the opening **configuration** tag.

```
<appSettings>
</appSettings>
```

- Add a self-closing **add** element with a **key** attribute value of **ListViewPagerSize**, and a **value** attribute value of **5**, to the **appSettings** element.

```
<add key="ListViewPagerSize" value="5" />
```

- Save and close the web.config file.
- Navigate to **DynamicData/PageTemplates**, and open the **List** Web Form in the Code view.
- In the **Page_Load** event handler of the partial **List** class, check whether the **ListViewPagerSize** key exists in the **appSettings** element of the web.config file. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **If** construct.

```
' Check if key exists in web.config
If Not
System.Web.Configuration.WebConfigurationManager.AppSettings("List
ViewPagerSize") Is Nothing Then
End If
```

- Retrieve the default page size value from the web.config file in the **Page_Load** event handler, in the **If** construct, by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in a variable named **pagerSize**, of type **Integer**.

```
' Get pager size from configuration file
Dim pagerSize As Integer = Integer.Parse(

System.Web.Configuration.WebConfigurationManager.AppSettings("List
ViewPagerSize"))
```

- Set the page size of the **GridView** control to the value of the **pagerSize** variable.

```
' Set page size
GridView1.PageSize = pagerSize
```

- Save the changes, and close the List.aspx.vb file.

▶ Task 5: View the new page size in the List view

- Run the application.
- View all countries.

Note: Notice that only 5 items display in the table. This is because the page size is set to 5.

- Close Internet Explorer.

▶ Task 6: View the Country Import Page

- Run the application.
- View the **Import Countries** page.

Note: Notice that it is possible to save the countries by using the **Save Countries** button.

- Close Internet Explorer.

▶ Task 7: Set the default value for enabling saving of the filtered countries

- Open the **web.config** file.
- In the web.config file, add a new self-closing **add** element within the **appSettings** element, with a **key** attribute value of **EnableSaveImportedCountries**, and a **value** attribute value of **true**.

```
<add key="EnableSaveImportedCountries" value="true" />
```

- Save and close the web.config file.
- Open the **ImportCountries** Web Form in Code view.
- Check whether the **EnableSaveImportedCountries** key exists in the **appSettings** element of the web.config file in the **Page_Load** event handler of the partial **ImportCountries** class. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **If** construct.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Check if key exists in web.config
    If Not
System.Web.Configuration.WebConfigurationManager.AppSettings("Enab
leSaveImportedCountries") Is Nothing Then
    End If
End Sub
```

- Retrieve the default value for enabling the user to save the imported countries from the **web.config** file in the **Page_Load** event handler, in the **If** construct, by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class. Save the retrieved value in a variable named **enableSaveCountries**, of type **Boolean**.
- Enable the **SaveButton** control, depending on the value of the **enableSaveCountries** local variable.

```
' Enable/disable SaveButton
SaveButton.Enabled = enableSaveCountries
```

- Save the changes, and close the ImportCountries.aspx.vb file.

▶ Task 8: View the Country Import page

- Run the application.
- View the **Import Countries** page.

Note: Notice that the **Save Countries** button is enabled.

- Close Internet Explorer.

▶ Task 9: Set the default value for enabling saving of the filtered countries

- Open the **web.config** file.
- In the web.config file, modify the self-closing add element within the **appSettings** element, with a **key** attribute value of **EnableSaveImportedCountries**, to have a **value** attribute value of **false**.

```
<add key="EnableSaveImportedCountries" value="false" />
```

- Save and close the web.config file.

▶ Task 10: View the Country Import page

- Run the application.
- View all countries.

Note: Notice that the **Save Countries** button is disabled.

- Close Internet Explorer.

Results: After completing this exercise, you will have saved a page size value to the web.config file, written the code to retrieve this value upon page load, and assigned the retrieved value to the page size of the **GridView** control in List view for Dynamic Data. In addition, you will have saved a value to the web.config file for enabling the **Save Countries** Button control on the ImportCountries Web Form, written the code to retrieve this value upon page load, and assigned the retrieved value to the **Enabled** property of the **SaveButton** control in the ImportCountries Web Form.

Exercise 2: Configuring the Visitor Counter

The main tasks for this exercise are as follows:

1. Set the default visitor counter value.
2. Set the visitor counter value in the **Application_Start** method.
3. Save the visitor counter value in the **Application_End** method.

▶ Task 1: Set the default visitor counter value

- Open the **web.config** file.
- In the web.config file, add a new self-closing **add** element within the **appSettings** element with a **key** attribute value of **VisitorCounter**, and a **value** attribute with a value of **0**.

```
<add key="VisitorCounter" value="0" />
```

- Save and close the web.config file.

▶ Task 2: Set the visitor counter value in the Application_Start method

- Open the **Global.asax** file.
- In the **Application_Start** method, declare a local variable named **numVisitors**, of type **Long**, and initialize the variable with a value of **0**.

```
Dim numVisitors As Long = 0
```

- Check whether the **VisitorCounter** key exists in the **appSettings** element of the web.config file in the **Application_Start** method. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **If** construct.

```
' Check if key exists in web.config
If Not
System.Web.Configuration.WebConfigurationManager.AppSettings("Visi
torCounter") Is Nothing Then
End If
```

- Retrieve the visitor counter value from the web.config file in the **Application_Start** method by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in the **numVisitors** variable.

```
' Get visitor counter from configuration file
numVisitors = Long.Parse(

System.Web.Configuration.WebConfigurationManager.AppSettings("Visi
torCounter"))
```

- Save the visitor counter value to the Application state in the **Visitors** application variable, by modifying the existing assignment of the value to **numVisitors**.

```
' Save application variable
Application("Visitors") = numVisitors
```

- Save the Global.asax file.

▶ Task 3: Save the visitor counter value in the Application_End method

- In the **Application_End** method, declare a local variable named **mainConfiguration**, of type **System.Configuration.Configuration**, and initialize the variable with a value returned by the **System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~/")** method.

```
' Get configuration instance
Dim mainConfiguration As System.Configuration.Configuration = _

System.Web.Configuration.WebConfigurationManager.OpenWebConfigurat
ion("~/")
```

- Check whether the **VisitorCounter** key exists in the **appSettings** element of the web.config file in the **Application_End** method by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **If Then...Else** construct.

```
' Check if key exists in web.config
If
System.Web.Configuration.WebConfigurationManager.AppSettings("Visi
torCounter") Is Nothing Then
Else
End If
```

- In the **Application_End** method, in the **If** construct, save a new **VisitorCounter** key and value in the web.config file. To do this, use the **Add** method of the **System.Web.Configuration.WebConfigurationManager.AppSettings** property. Set the value to the value saved in the **Visitors** application variable, if it exists. If the application variable does not exist, save the value of **0**.

```
' Save new key and value
mainConfiguration.AppSettings.Settings.Add(New
KeyValueConfigurationElement("VisitorCounter", _
    IIf(Application("Visitors") Is Nothing, "0",
Application("Visitors").ToString())))
```

- In the **Application_End** method, in the **Else** construct, save a new **VisitorCounter** value in the web.config file. To do this, use the **System.Web.Configuration.WebConfigurationManager.AppSettings** property. If the **Visitors** application variable exists, set the **VisitorCounter** value to the value saved in the **Visitors** application variable. If the **Visitors** application variable does not exist, assign the value **0**, to the **VisitorCounter** value.

```
' Save new value
mainConfiguration.AppSettings.Settings("VisitorCounter").Value = _
    IIf(Application("Visitors") Is Nothing, "0",
Application("Visitors").ToString())
```

- In the **Application_End** method, save the changes to the web.config file by using the **System.Configuration.Configuration.Save** method on the **mainConfiguration** object.

```
' Save to file
mainConfiguration.Save()
```

- Save and close the Global.asax file.

Results: After completing this exercise, you will have saved a visitor counter value to the web.config file, written code to retrieve this value upon application start, and assigned the retrieved value to the Visitors application variable. You will also have created code to save the current value of the Visitors application variable to the web.config file on application end.

Exercise 3: Deploying the Web Application

The main tasks for this exercise are as follows:

1. Build the application.
2. Copy the Web site.
3. Test the deployed Web site.

▶ Task 1: Build the application

- Build the application, and fix any compile-time errors.

▶ Task 2: Copy the Web site

- Open the Copy Web Site tool.
- Connect to the **CM** application on the **Default** Web Site on the local Internet Information Services (IIS).
- Copy source files to the destination site. Overwrite the existing files.
- Close the Copy Web Site tool.

▶ Task 3: Test the deployed Web site

- Open Internet Explorer, and enter the URL, **http://localhost:1112/CM**.
- On the default page of the deployed Contoso Customer Management Web site, notice that the **Number of visitors** is set to **1**.
- Close Internet Explorer.
- Open Internet Explorer again, and browse to **http://localhost:1112/CM**.
- On the default page of the deployed Customer Management Web site, notice that the **Number of visitors** is now set to **2**.
- Close Internet Explorer.
- Open IIS Manager as an Administrator.
- Stop IIS.
- Open the **web.config** file from the folder, **C:\inetpub\wwwroot\CM**, by using Windows Explorer.

- In Visual Studio 2010, notice the **add** element in the **appSettings** element, with a **key** attribute value of **VisitorCounter** and a **value** attribute value of **2**.
- Close the deployed web.config file.
- Open the **web.config** file from the project Web site that is currently open in Visual Studio 2010.
- In Visual Studio 2010, modify the **value** attribute value to **100** for the **add** element in the **appSettings** element, with a **key** attribute value of **VisitorCounter**.
- Save and close the web.config file.
- Open the Copy Web Site tool.
- Connect to the local IIS, and copy the web.config source file to the destination site.

Note: If a confirmation message box appears, click **Yes**.

- Close the Copy Web Site tool.
- Close Visual Studio 2010.
- Close Windows Explorer.
- Start IIS.
- Close IIS Manager.
- Open Internet Explorer, and browse to the address, **http://localhost:1112/CM**.

  On the default page of the deployed Contoso Customer Management Web site, notice that the **Number of visitors** is set to **101**.
- Close Internet Explorer.

▶ Task 4: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

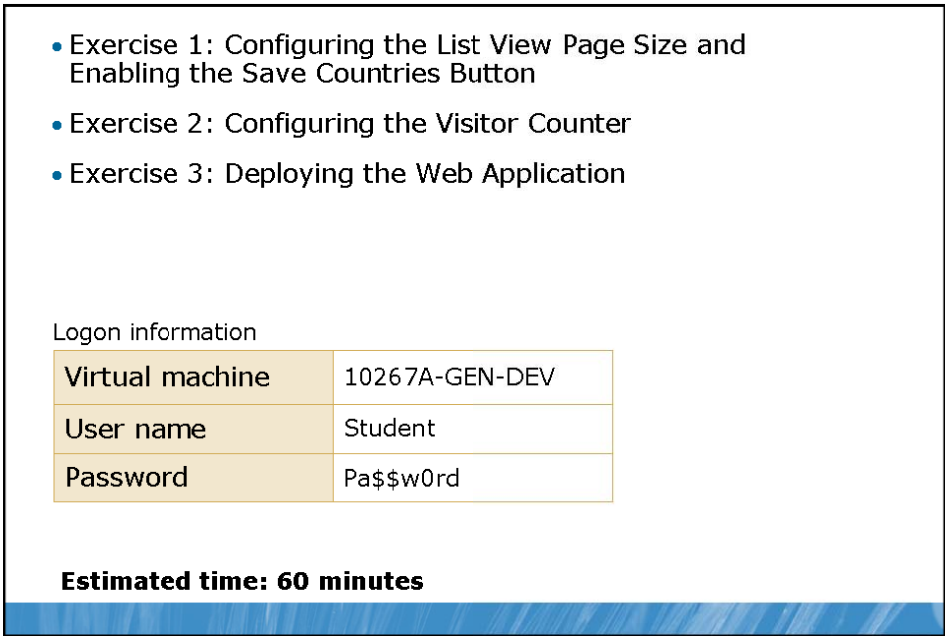
Results: After completing this exercise, you will have built the Web application, copied it to a virtual directory on the Default Web site on the local IIS, and tested the deployed Web site.

Module 14

Lab Instructions: Configuring and Deploying a Microsoft® ASP.NET Web Application (Visual C#)

Contents:

Lab: Configuring and Deploying a Microsoft ASP.NET Web Application



Note: You can perform tasks in this lab by using either Microsoft Visual Basic® or Microsoft Visual C#®. If you are using Visual Basic, refer to the steps provided in Section 1 of the lab document. If you are using Visual C#, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will configure an ASP.NET Web application by using the web.config file. In addition, you will deploy and publish the Web application to a Web site.

Objectives

After completing this lab, you will be able to:

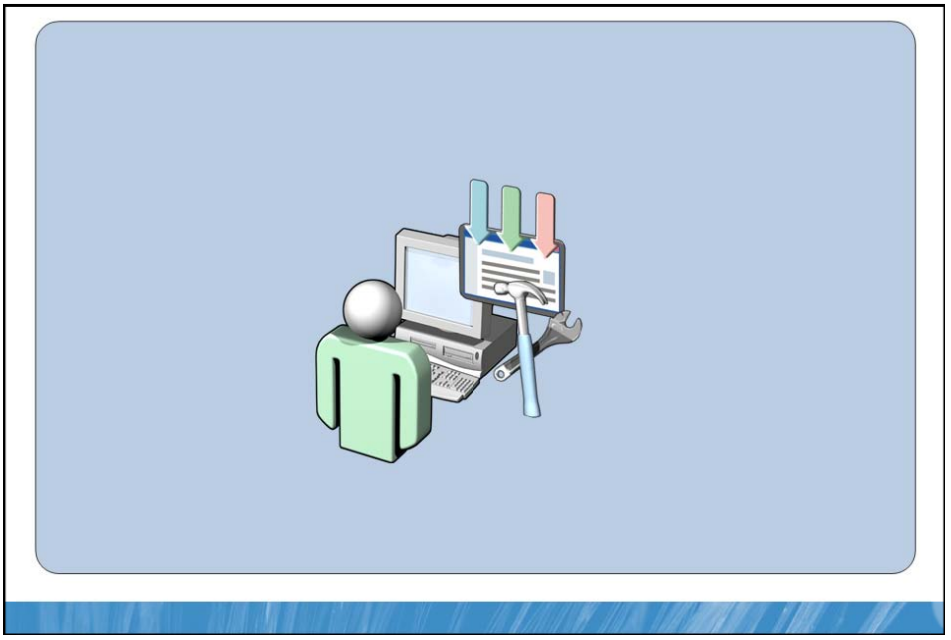
- Configure an ASP.NET Web application by using the web.config file.
- Deploy and publish an ASP.NET Web application to a Web site.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses a Customer Management application to create, customize, and manage its customer information. After configuring the Customer Management application, you need to deploy the Web site to the local IIS to help the sales team in different branches of Contoso, Ltd access the application. Before deploying the application to provide access for the branch offices, you need to test the application on the server. To do this, you need to save both static and dynamic values in the web.config file, because the web.config file accompanies the Customer Management application during deployment. In addition, you need to deploy the Web application to IIS.

Section 2: Visual C#

Exercise 1: Configuring the List View Page Size and Enabling the Save Countries Button

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Set the start page.
3. View the default page size in the List view.
4. Set the default List view page size.
5. View the new page size in the List view.
6. View the Country Import Page.
7. Set the default value for enabling saving of the filtered countries.
8. View the Country Import Page.
9. Set the default value for enabling saving of the filtered countries.
10. View the Country Import Page.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010 as an Administrator.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M14\CS folder.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.

▶ Task 3: View the default page size in the List view

- Run the application.
- View all the countries.

Note: Notice that by default, 10 items display in the table. This is because the page size is set to 10.

- Close Windows® Internet Explorer®.

▶ Task 4: Set the default List view page size

- Open the **web.config** file, and add the **appSettings** element after the opening **configuration** tag.

```
<appSettings>
</appSettings>
```

- Add a self-closing **add** element with a **key** attribute value of **ListViewPagerSize**, and a **value** attribute value of **5**, to the **appSettings** element.

```
<add key="ListViewPagerSize" value="5" />
```

- Save and close the web.config file.
- Navigate to **DynamicData/PageTemplates**, and open the **List** Web Form in the Code view.
- In the **Page_Load** event handler of the partial **List** class, check whether the **ListViewPagerSize** key exists in the **appSettings** element of the web.config file. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class by using an **if** construct.

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings["Lis
tViewPagerSize"] != null)
{
}
```

- Retrieve the default page size value from the web.config file in the **Page_Load** event handler, in the **if** construct, by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in a variable named **pagerSize**, of type **int**.

```
// Get pager size from configuration file
int pagerSize = int.Parse(

System.Web.Configuration.WebConfigurationManager.AppSettings["List
ViewPagerSize"]);
```

- Set the page size of the **GridView** control to the value of the **pagerSize** variable.

```
// Set page size
GridView1.PageSize = pagerSize;
```

- Save the changes, and close the List.aspx.cs file.

▶ Task 5: View the new page size in the List view

- Run the application.
- View all countries.

Note: Notice that only 5 items display in the table. This is because the page size is set to 5.

- Close Internet Explorer.

▶ Task 6: View the Country Import page

- Run the application.
- View the **Import Countries** page.

Note: Notice that it is possible to save the countries by using the **Save Countries** button.

- Close Internet Explorer.

▶ Task 7: Set the default value for enabling saving of the filtered countries

- Open the **web.config** file.
- In the web.config file, add a new self-closing **add** element within the **appSettings** element, with a **key** attribute value of **EnableSaveImportedCountries**, and a **value** attribute value of **true**.

```
<add key="EnableSaveImportedCountries" value="true" />
```

- Save and close the web.config file.
- Open the **ImportCountries** Web Form in Code view.
- Check whether the **EnableSaveImportedCountries** key exists in the **appSettings** element of the web.config file in the **Page_Load** event handler of the partial **ImportCountries** class. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **if** construct.

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings["Ena
bleSaveImportedCountries"] != null)
{
}
```

- Retrieve the default value for enabling the user to save the imported countries from the **web.config** file in the **Page_Load** event handler, in the **if** construct, by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in a variable named **enableSaveCountries**, of type **bool**.
- Enable the **SaveButton** control, depending on the value of the **enableSaveCountries** local variable.

```
// Enable/disable SaveButton
SaveButton.Enabled = enableSaveCountries;
```

- Save the changes, and close the ImportCountries.aspx.cs file.

▶ Task 8: View the Country Import page

- Run the application.
- View the **Import Countries** page.

Note: Notice that the Save Countries button is enabled.

- Close Internet Explorer.

▶ Task 9: Set the default value for enabling saving of the filtered countries

- Open the **web.config** file.
- In the web.config file, modify the self-closing **add** element within the **appSettings** element, with a **key** attribute value of **EnableSaveImportedCountries**, to have a **value** attribute value of **false**.

```
<add key="EnableSaveImportedCountries" value="false" />
```

- Save and close the web.config file.

▶ Task 10: View the Country Import page

- Run the application.
- View all countries.

Note: Notice that the Save Countries button is disabled.

- Close Internet Explorer.

Results: After completing this exercise, you will have saved a page size value to the web.config file, written the code to retrieve this value upon page load, and assigned the retrieved value to the page size of the **GridView** control in List view for Dynamic Data. In addition, you will have saved a value to the web.config file for enabling the **Save Countries** Button control on the ImportCountries Web Form, written the code to retrieve this value upon page load, and assigned the retrieved value to the **Enabled** property of the **SaveButton** control in the ImportCountries Web Form.

Exercise 2: Configuring the Visitor Counter

The main tasks for this exercise are as follows:

1. Set the default visitor counter value.
2. Set the visitor counter value in the **Application_Start** method.
3. Save the visitor counter value in the **Application_End** method.

▶ Task 1: Set the default visitor counter value

- Open the **web.config** file.
- In the web.config file, modify the value attribute of the self-closing **add** element within the **appSettings** element with a **key** attribute value of **VisitorCounter**, to and a **value** attribute with a value of **0**.

```
<add key="VisitorCounter" value="0" />
```

- Save and close the web.config file.

▶ Task 2: Set the visitor counter value in the Application_Start method

- Open the **Global.asax** file.
- In the **Application_Start** method, declare a local variable named **numVisitors**, of type **long**, and initialize the variable with a value of **0**.

```
Dim numVisitors As Long = 0
```

- Check whether the **VisitorCounter** key exists in the **appSettings** element of the web.config file in the **Application_Start** method. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **if** construct.

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings["Vis
itorCounter"] != null)
{
}
```

- Retrieve the visitor counter value from the web.config file in the **Application_Start** method by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in the **numVisitors** variable.

```
// Get visitor counter from configuration file
numVisitors = long.Parse(

System.Web.Configuration.WebConfigurationManager.AppSettings["Visi
torCounter"]);
```

- Save the visitor counter value to the Application state, in the **Visitors** application variable, by modifying the existing assignment of the value to **numVisitors**.

```
// Save application variable
Application["Visitors"] = numVisitors;
```

- Save the Global.asax file.

▶ Task 3: Save the visitor counter value in the Application_End method

- In the **Application_End** method, declare a local variable named **mainConfiguration**, of type **System.Configuration.Configuration**, and initialize the variable with a value returned by the **System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~/")** method.

```
// Get configuration instance
System.Configuration.Configuration mainConfiguration =

System.Web.Configuration.WebConfigurationManager.OpenWebConfigurat
ion("~/");
```

- Check whether the **VisitorCounter** key exists in the **appSettings** element of the web.config file in the **Application_End** method by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **if...else** construct.

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings["Vis
itorCounter"] == null)
{
}
else
{
}
```

- In the **Application_End** method, in the **If** construct, save a new **VisitorCounter** key and value in the web.config file. To do this, use the **Add** method of the **System.Web.Configuration.WebConfigurationManager.AppSettings** property. Set the value to the value saved in the **Visitors** application variable, if it exists. If the application variable does not exist, save the value of **0**.

```
// Save new key and value
mainConfiguration.AppSettings.Settings.Add(new
KeyValueConfigurationElement("VisitorCounter",
    Application["Visitors"] == null ? "0" :
Application["Visitors"].ToString()));
```

- In the **Application_End** method, in the **else** construct, save a new **VisitorCounter** value in the web.config file. To do this, use the **System.Web.Configuration.WebConfigurationManager.AppSettings** property. If the **Visitors** application variable exists, set the **VisitorCounter** value to the value saved in the **Visitors** application variable. If the **Visitors** application does not exist, assign the value, **0**, to the **VisitorCounter** value.

```
// Save new value
mainConfiguration.AppSettings.Settings["VisitorCounter"].Value =
    Application["Visitors"] == null ? "0" :
Application["Visitors"].ToString();
```

- In the **Application_End** method, save the changes to the web.config file by using the **System.Configuration.Configuration.Save** method on the **mainConfiguration** object.

```
// Save to file
mainConfiguration.Save();
```

- Save and close the **Global.asax** file.

Results: After completing this exercise, you will have saved a visitor counter value to the web.config file, written the code to retrieve this value upon application start, and assigned the retrieved value to the **Visitors** application variable. You will also have created the code to save the current value of the **Visitors** application variable to the web.config file on application end.

Exercise 3: Deploying the Web Application

The main tasks for this exercise are as follows:

1. Build the application.
2. Copy the Web site.
3. Test the deployed Web site.

▶ Task 1: Build the application

- Build the application, and fix any compile-time errors.

▶ Task 2: Copy the Web site

- Open the Copy Web Site tool.
- Connect to the **CM** application on the **Default Web Site** on the local IIS.
- Copy source files to the destination site. Overwrite the existing files.
- Close the Copy Web Site tool.

▶ Task 3: Test the deployed Web site

- Open Internet Explorer, and enter the URL, **http://localhost:1112/CM**.
- On the default page of the deployed Contoso Customer Management Web site, notice that the **Number of visitors** is set to **1**.
- Close Internet Explorer.
- Open Internet Explorer again, and browse to **http://localhost:1112/CM**.
- On the default page of the deployed Customer Management Web site, notice that the **Number of visitors** is now set to **2**.
- Close Internet Explorer.
- Open IIS Manager as an Administrator.
- Stop IIS.
- Open the **web.config** file from the folder **C:\inetpub\wwwroot\CM**, by using Windows Explorer.

- In Visual Studio 2010, notice the **add** element in the **appSettings** element, with a **key** attribute value of **VisitorCounter**, and a **value** attribute value of **2**.
- Close the deployed **web.config** file.
- Open the **web.config** file from the project Web site that is currently open in Visual Studio 2010.
- In Visual Studio 2010, modify the **value** attribute value to **100** for the **add** element in the **appSettings** element, with a **key** attribute value of **VisitorCounter**.
- Save and close the **web.config** file.
- Open the Copy Web Site tool.
- Connect to the local IIS, and copy the web.config source file to the destination site.

Note: If a confirmation message box opens, click **Yes** in the message box.

- Close the Copy Web Site tool.
- Close Visual Studio 2010.
- Close Windows Explorer.
- Start IIS.
- Close IIS Manager.
- Open Internet Explorer, and browse to **http://localhost:1112/CM**.

  On the default page of the deployed Contoso Customer Management Web site, notice that the **Number of visitors** is set to **101**.
- Close Internet Explorer.

▶ Task 4: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

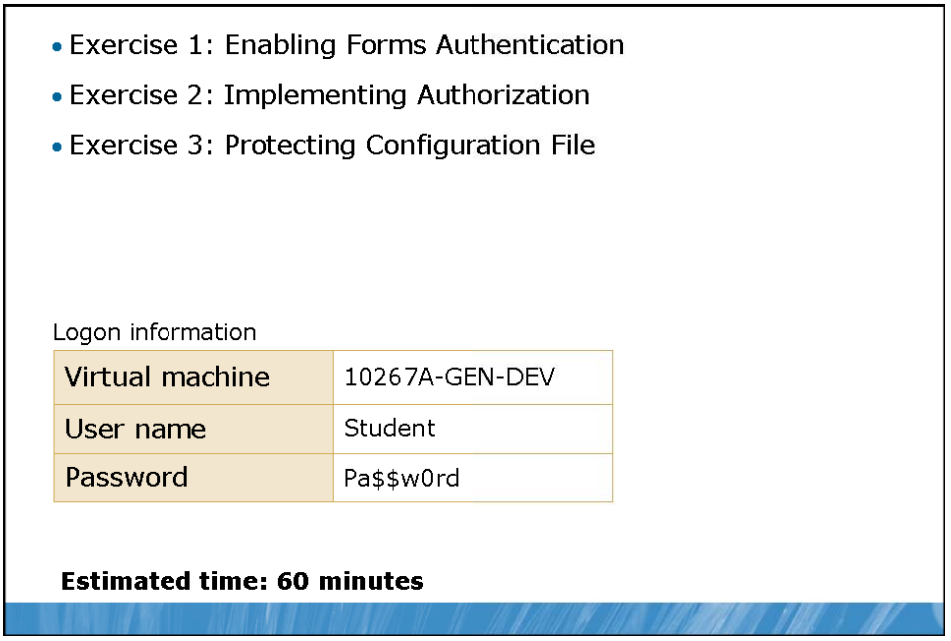
Results: After completing this exercise, you will have built the Web application, copied it to a virtual directory on the Default Web site on the local IIS, and tested the deployed Web site.

Module 15

Lab Instructions: Securing a Microsoft® ASP.NET Web Application (Visual Basic)

Contents:

Lab: Securing a Microsoft ASP.NET Web Application



Note: You can perform tasks in this lab by using either Microsoft Visual Basic® or Microsoft Visual C#®. If you are using Visual Basic, refer to the steps provided in Section 1 of the lab document. If you are using Visual C#, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will enable Forms authentication, implement role-based authorization for an ASP.NET Web application, and protect parts of the configuration file. In addition, you will use ASP.NET Login controls to implement authentication.

Objectives

After completing this lab, you will be able to:

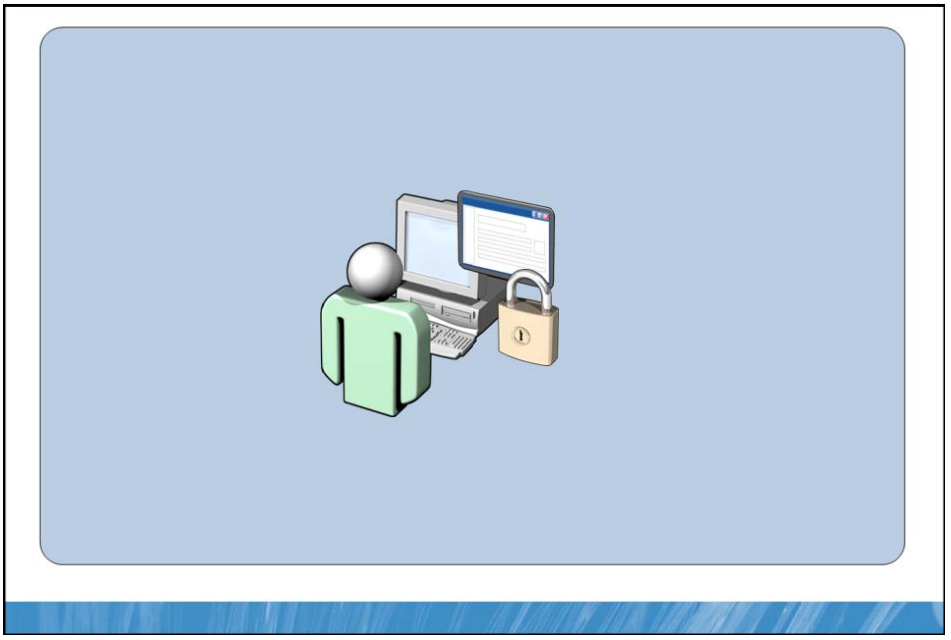
- Enable Forms authentication.
- Implement authorization.
- Protect sections of a configuration file.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage their customer information. Your organization has created a Web site to manage customer data and services in ASP.NET.

The senior developer has requested that you add some Login controls and a login page to the application. As part of the requirements of the Sales team, you need to add security features to the Customer Management application. You need to ensure that anonymous access is not allowed to the Customer Management Web application. In addition, you need to implement authentication based on the role for the users who do not belong to the AD DS of the company. To do this, you need to implement Forms authentication and role-based security for the Customer Management Web application. Finally, you must make the Import Countries page, which is accessible only to members of the Admins role, and protect parts of the configuration file.

Section 1: Visual Basic

Exercise 1: Enabling Forms Authentication

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Set the start page.
3. Add the Login controls.
4. Test the Login controls.
5. Create a sample ASP.NET Web site.
6. Add the Account files and folders.
7. Update Account Web Forms to use an existing master page.
8. Modify the Login Web Form.
9. Enable Forms authentication.
10. Test the Login controls.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M15\VB folder.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.

▶ Task 3: Add the Login controls

- Open the master page, **Site.master**, in Source view.

- Add a **div** element with a **class** attribute with the value of **login** to the **body** element of the master page, **Site.master**, after the **div** element with a **class** attribute set to the value of **appTitle**.

```
<div class="login">
</div>
```

- Add a **LoginStatus** control named **MainLoginStatus** to the **div** element with a **class** attribute value of **login**.

```
<asp:LoginStatus ID="MainLoginStatus" runat="server"
LogoutAction="RedirectToLoginPage"
    LogoutPageUrl="~/Account/Login.aspx" />
```

Note: The Login.aspx Web Form does not yet exist. You will add one later in this exercise.

Note: The **LoginStatus** control must have the **LogoutAction** attribute set to the value of **RedirectToLoginPage** to send the user to the login page. Optionally, the attribute **LogoutPageUrl**, can be set to the value of **~/Account/Login.aspx**, to redirect to this specific page, instead of the one specified in the web.config file.

- Add a **LoginName** control named **MainLoginName**, to the **div** element, after the **MainLoginStatus** control.

```
<asp:LoginName ID="MainLoginName" runat="server" />
```

- Save and close the master page.
- Add the following style to the Styles/Site.css stylesheet.

```
div.login
{
    position: fixed;
    top: 49px;
    right: 10px;
    padding-left: 10px;
    padding-bottom: 10px;
    background-color: #ffffff;
}
```

- Save and close the Site.css file.

▶ Task 4: Test the Login controls

- Run the application.
- View the **Login** controls.

Note: Notice the **Login** controls in the upper-right corner of the window, a **Logout** link, and the currently signed-in user, **10267A-GEN-DEV\student**, because of the default Windows authentication.

- Close Internet Explorer.

▶ Task 5: Create a sample ASP.NET Web site

- Open a second instance of Visual Studio 2010.
- Create a new file system-based ASP.NET Web site in **D:\Labfiles\Starter\M15\VB\SampleWebSite**, by using the ASP.NET Web Site template.
- Close second instance of Visual Studio 2010.

▶ Task 6: Add the Account files and folders

- Add the **Account** folder and default account content from the SampleWebSite Web site to the **CustomerManagement** Web site, by copying the Account folder from the path, **D:\Labfiles\Starter\M15\VB\SampleWebSite\Account**. Use Windows Explorer to copy the Account folder.
- In Solution Explorer, refresh the Web site content to ensure the addition of the **Account** folder.

▶ Task 7: Update Account Web Forms to use an existing master page

- Expand the **Account** folder.
- Open the **ChangePassword.aspx** Web Form in Source view.
- Replace the name of the ContentPlaceHolder named **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.

- Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

```
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
```

- Save and close the **ChangePassword.aspx** Web Form.
- Open the **ChangePasswordSuccess.aspx** Web Form in Source view.
- Replace the name of the ContentPlaceHolder named **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.
- Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

```
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
```

- Save and close **ChangePasswordSuccess.aspx** Web Form.
- Open the **Login.aspx** Web Form in Source view.
- Replace the name of the ContentPlaceHolder referenced, **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.
- Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

```
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
```

- Save and close **Login.aspx** Web Form.
- Open the **Register.aspx** Web Form in Source view.
- Replace the name of the ContentPlaceHolder named **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.

- Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

```
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
```

- Save and close **Register.aspx** Web Form.

▶ Task 8: Modify the Login Web Form

- Open the **Login** Web Form.
- Set the page title as **Contoso Customer Management – User Login**.
- Save and close the **Login** Web Form.

▶ Task 9: Enable Forms authentication

- Open the **web.config** file in the project root folder.
- Add to the web.config file an **authentication** element that specifies **Forms** authentication. Place the authentication element in the **system.web** element.

```
<authentication mode="Forms">
</authentication>
```

- Add to the **authentication** element a **forms** element that specifies the value of the **loginUrl** attribute as **~/Account/Login.aspx**.

```
<forms loginUrl="~/Account/Login.aspx" />
```

- Save the **web.config** file.

▶ Task 10: Test the Login controls

- Stop the ASP.NET Development Server by using the icon in the System tray.
- Run the application.
- View the **Login** controls.

Note: Notice the **Login** controls at the upper-right corner of the window, which now display a login link. No user is currently signed in, because of the Forms authentication.

- Open the **Login** Web Form by clicking the **Login** link.
- Close Internet Explorer.

Results: After completing this exercise, you will have added Login controls to the master page, added Account files and folders from a sample Web site, updated the Account Web Forms to match the current master page, updated the Login Web Form, and enabled Forms authentication.

Exercise 2: Implementing Authorization

The main tasks for this exercise are as follows:

1. Set up the connection to the membership database.
2. Disallow anonymous access.
3. Allow access to the Styles folder.
4. Create the default database for application services.
5. Test the anonymous access.
6. Enable the security trimming of the site map.
7. Disallow access to the Import Countries page.
8. Allow access to the Customers, Countries, and Help menu items.
9. Disallow access to the Import Countries menu item.
10. Test the access to the Import Countries menu item.
11. Add student to the Admins role.
12. Test the Import Country access.

▶ Task 1: Set up the connection to the membership database

- Add a new connection string named **ApplicationServices** to the web.config file by adding an **add** element in the **connectionStrings** element.

```
<add name="ApplicationServices" connectionString="Data
Source=.\SQLEXPRESS;Integrated
Security=True;AttachDBFilename=|DataDirectory|\ASPNETDB.MDF;User
Instance=true" providerName="System.Data.SqlClient" />
```

- Save the **web.config** file.

▶ Task 2: Disallow anonymous access

- Add an **authorization** element to the web.config file, after the **authentication** element.

```
<authorization>
</authorization>
```

- Add a self-closing **deny** element to the authorization element that disallows all anonymous users.

```
<deny users="?" />
```

- Save the **web.config** file.

▶ Task 3: Allow access to the Styles folder

- Add a **location** element to the web.config file with a **path** attribute value of **Styles**, within the **configuration** element.

```
<location path="Styles">
</location>
```

- Add a **system.web** element to the web.config file, in the **location** element.

```
<system.web>
</system.web>
```

- Add an **authorization** element to the web.config file, in the **system.web** element.

```
<authorization>
</authorization>
```

- Add an **allow** element to the web.config file with a **users** attribute value of *, in the **authorization** element.

```
<allow users="*"/>
```

- Save the **web.config** file.

▶ Task 4: Create the default database for application services

- In Solution Explorer, click the **ASP.NET Configuration** button.

Note: This step will take a while.

- In the ASP.Net Web Administration- Windows Internet Explorer window, in the Home tab, click **Security**.

- In the **Security** tab, click **Create user**.
- In the **Security** tab, in the **Create user** section, complete the following settings, and then click **Create User**:
  - User Name: **student**
  - Password: **Pa$$w0rd**
  - Confirm Password: **Pa$$w0rd**
  - E-mail: **student@contoso.com**
  - Security Question: **Contoso Founder**
  - Security Answer: **John Contoso**
- In the ASP.Net Web Administration- Windows Internet Explorer window, click the **Close** button.
- In Solution Explorer, click the **Refresh** button, and then expand **App_Data**.

Note: Notice that the App_Data folder now contains the default ASPNETDB.MDF database file that is used for the application services.

▶ Task 5: Test the anonymous access

- Run the application.
- View the **Login** Web Form.

Note: Notice that the **Login** Web Form displays instead of the **Default** Web Form. This is because you have not yet logged in to the Web site.

- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- View the **Default** Web Form.

Note: Notice that the **Default** Web Form displays after redirecting you from the **Login** Web Form.

- Close Internet Explorer.

▶ Task 6: Enable the security trimming of the site map

- Add a **siteMap** element to the **system.web** element in the web.config file, with a **defaultProvider** attribute value of **AspNetXmlSiteMapProvider**, and an **enabled** attribute value of **true**.

```
<siteMap defaultProvider="AspNetXmlSiteMapProvider"
enabled="true">
</siteMap>
```

- Add a **providers** element to the web.config file, in the **siteMap** element.

```
<providers>
</providers>
```

- Remove the default **AspNetXmlSiteMapProvider** provider element from the web.config file by using a self-closing **remove** element.

```
<remove name="AspNetXmlSiteMapProvider"/>
```

- Add a new **AspNetXmlSiteMapProvider** element to the web.config file by using a self-closing **add** element with a **type** attribute value of **System.Web.XmlSiteMapProvider, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a**, a **siteMap** attribute value of **web.sitemap**, and a **securityTrimmingEnabled** attribute value of **true**.

```
<add name="AspNetXmlSiteMapProvider"
    type="System.Web.XmlSiteMapProvider, System.Web,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
    siteMapFile="web.sitemap"
    securityTrimmingEnabled="true" />
```

- Save the **web.config** file.

▶ Task 7: Disallow access to the Import Countries page

- Add a **location** element to the web.config file **configuration** element, with a **path** attribute value of **ImportCountries.aspx**.

```
<location path="ImportCountries.aspx">
</location>
```

- Add a **system.web** element to the web.config file, in the **location** element.

```
<system.web>
</system.web>
```

- Add an **authorization** element to the web.config file, in the **system.web** element.

```
<authorization>
</authorization>
```

- Add an **allow** element to the web.config file **authorization** element, with a **roles** attribute value of **Admins**.

```
<allow roles="Admins"/>
```

- Add a **deny** element to the web.config file with a **users** attribute value of *, in the **authorization** element.

```
<deny users="*"/>
```

- Save and close the **web.config** file.

▶ Task 8: Allow access to the Customers, Countries, and Help menu items

- Open the web.sitemap file.
- Set the **roles** attribute of the **Customers siteMapNode** element to a value of *.

```
roles="*"
```

- Set the **roles** attribute of the **Countries siteMapNode** element to a value of *.

```
roles="*"
```

- Set the **roles** attribute of the **Help siteMapNode** element to a value of *.

```
roles="*"
```

▶ Task 9: Disallow access to the Import Countries menu item.

- Set the **roles** attribute of the **Import siteMapNode** element to a value of **Admins**.

```
roles="Admins"
```

- Save and close the **web.sitemap** file.

▶ Task 10: Test the access to the Import Countries menu item

- Stop the ASP.NET Development Server by using the icon in the System tray.
- Run the application.
- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- Notice that the **Import** menu item is missing.

Note: Notice that the **Import** menu item has been hidden, because the user, Student, is not a member of the Admins role.

- Browse to the **ImportCountries.aspx** Web Form by using the URL **http://localhost:1111/CustomerManagement/ImportCountries.aspx**.

Note: Notice that you are redirected to the login page, because you do not have access to the **ImportCountries** page.

- Close Internet Explorer.

▶ Task 11: Add student to the Admins role

- Open the ASP.NET Web Site Administration tool.
- Enable roles by using the **Security** tab.
- Create an Admins role.
- Add a student user to the Admins role.
- Close Internet Explorer.

▶ Task 12: Test the Import Country access

- Stop the ASP.NET Development Server by using the ASP.NET Development Server icon in the System tray.
- Run the application.
- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- View the **Import** menu item.

Note: Notice the Import menu item now displays, because the user, Student, is a member of the Admins role.

- Open the **Import Countries** page.

Note: Notice that the Import **Countries** page displays.

- Close Internet Explorer.
- Close Microsoft Visual Studio 2010.

Results: After completing this exercise, you will have set up the connection to the membership database, disallowed anonymous access to the Web site, tested the anonymous access, enabled security trimming of the site map, and tested the access to the **Import Country** menu item and page.

Exercise 3: Protecting Configuration File

The main tasks for this exercise are as follows:

1. Grant the IIS Application Pool Identity access to the RSA key container.
2. Copy the Web site.
3. Test the deployed Web site.
4. Encrypt the connectionStrings section in the configuration file.
5. Test the deployed Web site.

▶ Task 1: Grant the IIS Application Pool Identity access to the RSA key container

- Open the Visual Studio 2010 command prompt as an Administrator.
- Grant the NETWORK SERVICE account access to the default machine-level **NetFrameworkConfigurationKey** RSA key container by running the following command from the Visual Studio 2010 Command Prompt window, which you should type in on a single line.

```
aspnet_regiis -pa "NetFrameworkConfigurationKey"
 "NT AUTHORITY\NETWORK SERVICE"
```

▶ Task 2: Copy the Web site

- Open Microsoft® Visual Studio® 2010 as an Administrator.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M15\VB folder.
- Open the Copy Web Site tool.
- Connect to the CM application on the **Default Web Site** on the local IIS.
- Copy the source files to the destination site.
- Close the Copy Web Site tool.

▶ Task 3: Test the deployed Web site

- Open Internet Explorer, and enter the URL **http://localhost:1112/CM**.
- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- Close Internet Explorer.

▶ Task 4: Encrypt the connectionStrings section in the configuration file

- Encrypt the **connectionStrings** section of the web.config file for the deployed Web site, which has an application name of CM, by running the following command from the Visual Studio Command Prompt (2010).

```
aspnet_regiis -pef "connectionStrings" "C:\inetpub\wwwroot\CM"
```

- Close the Visual Studio 2010 Command Prompt.
- In Windows Explorer, open the **web.config** file from the folder **C:\inetpub\wwwroot\CM**.

- In Visual Studio 2010, notice the connectionStrings element, which is now encrypted, and looks similar to the following markup.

```
<connectionStrings
configProtectionProvider="RsaProtectedConfigurationProvider">
  <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
    xmlns="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <KeyName>Rsa Key</KeyName>
        </KeyInfo>
        <CipherData>

<CipherValue>KmUnZDOmmWvygLNCaQbDl6Aj3/RX8peI9O/WAJH8A91RYL3N7+caF
u1fnD9gcxAYO4Jl3ERKBceXRATRXdDp72uqAm9TQvxAx3OGV79hYqKD44zY3SrjcGw
1Wp+iDYB7K+GlzMsOwn4SV/C5P6/DNEp9EFhpouV75u/OPzAOsag=</CipherValue
>
        </CipherData>
      </EncryptedKey>
    </KeyInfo>
    <CipherData>

<CipherValue>YU4c3gvwctn4dZAf/9aTqX2Zl6SOuUIWVnoYZO4aCVGM7mLpnt4uJ
OddwzdxOfSWaPLLLivXmE5RJ5ea/8nCM54UYniYiKPkVqptTphkLW5vcQK1typHs75
DHMuvzr4UO321IS8fNK9WUAJL5dJCrOQzuOBYgxHVRdLwJ5JbwgpRIjxAJSqtQfh7W
pASi6iRXHdfTXKJpQUtE9hGcRz/mIvj4QQ9I4sno//aFYcnEbCZtYOZMHtoOj1ukUW
qxQP3f5Cl4WxDq6MHDTwTigeHKJ9sYKxkRlGQKJZW131xUF+BHdy43K5/eDtOtCaCQ
vUjIOOI8eka7yauiHlp9hAsuUHwcyQnF4OXqhOjBAqLgHiJBOtlOEg4pLV/k2NYk7V
BQ3sqQSxNeQJDiD87TApFE6+FddnOFjxvttuRVmjjPdG5cdsva4PgyEv7+I4OySTru
xdDT6zV+lvgfza2Sxvfkwle W6KnxGRCU4DaiIBrcefNvbGo+gvVeiZN2qsaH9J4nA/
lfkpn2mTUUF8I3YZWMThmiWYnAyL3xy/6MMJJ+oeGUffouvMOpu7H/JNEOdEghiXj/
ybQr4g=</CipherValue>
    </CipherData>
  </EncryptedData>
</connectionStrings>
```

- Close the **web.config** file.

▶ Task 5: Test the deployed Web site

- Open Internet Explorer, and enter the URL **http://localhost:1112/CM**.
- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- Show all customers.
- Close Internet Explorer.
- Close Microsoft Visual Studio 2010.

Task 6: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

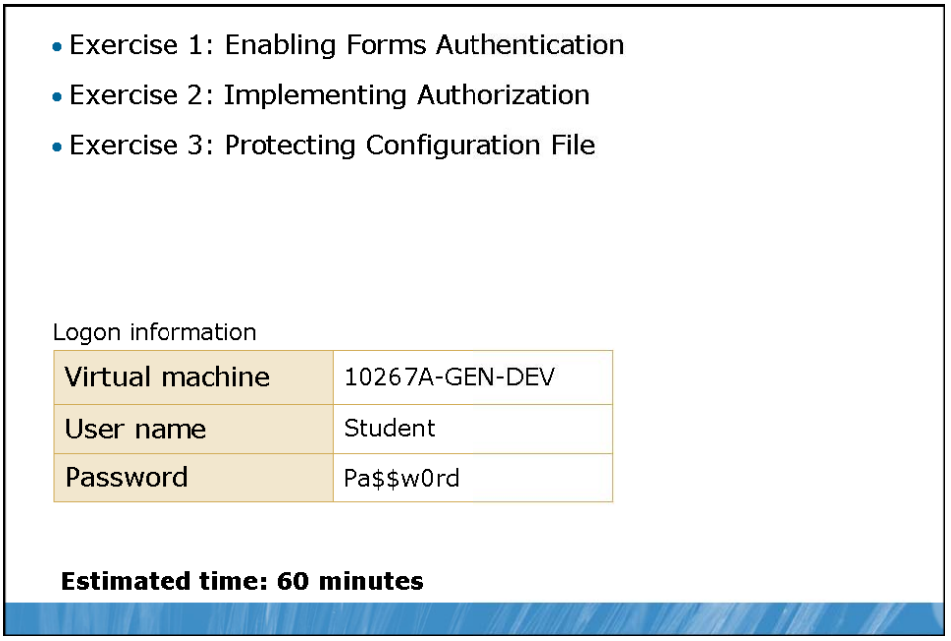
Result: After completing this exercise, you will have encrypted sections of the web.config file, and deployed the Web site to IIS.

Module 15

Lab Instructions: Securing a Microsoft® ASP.NET Web Application (Visual C#)

Contents:

Lab: Securing a Microsoft ASP.NET Web Application



Note: You can perform tasks in this lab by using either Microsoft Visual Basic® or Microsoft Visual C#®. If you are using Visual Basic, refer to the steps provided in Section 1 of the lab document. If you are using Visual C#, refer to the steps provided in Section 2 of the lab document.

Introduction

In this lab, you will enable Forms authentication, implement role-based authorization for an ASP.NET Web application, and protect parts of the configuration file. In addition, you will use ASP.NET Login controls to implement authentication.

Objectives

After completing this lab, you will be able to:

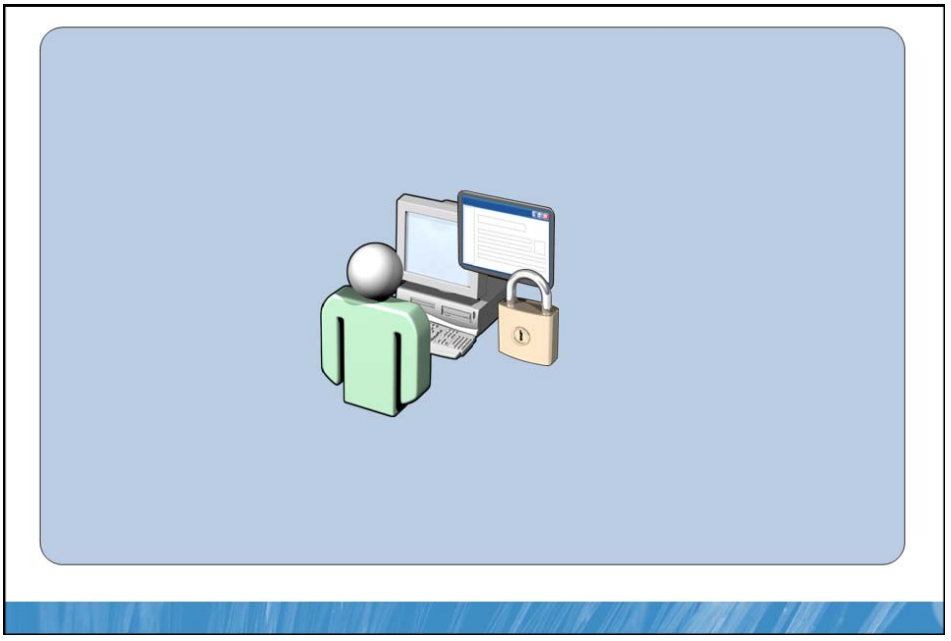
- Enable Forms authentication.
- Implement authorization.
- Protect sections of a configuration file.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses Microsoft .NET applications to create, customize, and manage their customer information. Your organization has created a Web site to manage customer data and services in ASP.NET.

The senior developer has requested that you add some Login controls and a login page to the application. As part of the requirements of the Sales team, you need to add security features to the Customer Management application. You need to ensure that anonymous access is not allowed to the Customer Management Web application. In addition, you need to implement authentication based on the role for the users who do not belong to the AD DS of the company. To do this, you need to implement Forms authentication and role-based security for the Customer Management Web application. Finally, you must make the Import Countries page, which is accessible only to members of the Admins role, and protect parts of the configuration file.

Section 2: Visual C#

Exercise 1: Enabling Forms Authentication

The main tasks for this exercise are as follows:

1. Open an existing ASP.NET Web site.
2. Set the start page.
3. Add the Login controls.
4. Test the Login controls.
5. Create a sample ASP.NET Web site.
6. Add the Account files and folders.
7. Update Account Web Forms to use an existing master page.
8. Modify the Login Web Form.
9. Enable Forms authentication.
10. Test the Login controls.

▶ Task 1: Open an existing ASP.NET Web site

- Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M15\CS folder.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.

▶ Task 3: Add the Login controls

- Open the master page, **Site.master**, in Source view.

- Add a **div** element with a **class** attribute with the value of **login**, to the **body** element of the master page **Site.master**, after the **div** element with a **class** attribute set to the value of **appTitle**.

```
<div class="login">
</div>
```

- Add a **LoginStatus** control named **MainLoginStatus**, to the **div** element with a **class** attribute value of **login**.

```
<asp:LoginStatus ID="MainLoginStatus" runat="server"
LogoutAction="RedirectToLoginPage"
    LogoutPageUrl="~/Account/Login.aspx" />
```

Note: The **Login.aspx** Web Form does not yet exist. You will add one later in this exercise.

Note: The **LoginStatus** control must have the **LogoutAction** attribute set to the value of **RedirectToLoginPage** to send the user to the login page. Optionally, the attribute, **LogoutPageUrl**, can be set to the value of **~/Account/Login.aspx**, to redirect to this specific page, instead of the one specified in the web.config file.

- Add a **LoginName** control named **MainLoginName** to the **div** element, after the **MainLoginStatus** control.

```
<asp:LoginName ID="MainLoginName" runat="server" />
```

- Save and close the master page.
- Add the following style to the **Styles**/**Site.css** stylesheet.

```
div.login
{
    position: fixed;
    top: 49px;
    right: 10px;
    padding-left: 10px;
    padding-bottom: 10px;
    background-color: #ffffff;
}
```

- Save and close the **Site.css** file.

▶ Task 4: Test the Login controls

- Run the application.
- View the **Login** controls.

Note: Notice the Login controls in the upper-right corner of the window, a **Logout** link, and the currently signed-in user, 10267A-GEN-DEV\student, because of the default Windows authentication.

- Close Internet Explorer.

▶ Task 5: Create a sample ASP.NET Web site

- Open a second instance of Visual Studio 2010.
- Create a new file system based ASP.NET Web site in **D:\Labfiles\Starter\M15\CS\SampleWebSite**, by using the ASP.NET Web Site template.
- Close second instance of Visual Studio 2010.

▶ Task 6: Add the Account files and folders

- Add the **Account** folder and default account content from the SampleWebSite Web site to the **CustomerManagement** Web site by copying the Account folder from the path **D:\Labfiles\Starter\M15\CS\SampleWebSite\Account**. Use Windows Explorer to copy the Account folder.
- In Solution Explorer, refresh the Web site content to ensure the addition of the **Account** folder.

▶ Task 7: Update Account Web Forms to use an existing master page

- Expand the **Account** folder.
- Open the **ChangePassword.aspx** Web Form in Source view.
- Replace the name of the **ContentPlaceHolder** control named **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.

- Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

```
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
```

- Save and close the **ChangePassword.aspx** Web Form.
- Open the **ChangePasswordSuccess.aspx** Web Form in Source view.
- Replace the name of the **ContentPlaceHolder** control named **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.
- Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

```
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
```

- Save and close the **ChangePasswordSuccess.aspx** Web Form.
- Open the **Login.aspx** Web Form in Source view.
- Replace the name of the **ContentPlaceHolder** control named **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.
- Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

```
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
```

- Save and close the **Login.aspx** Web Form.
- Open the **Register.aspx** Web Form in Source view.
- Replace the name of the **ContentPlaceHolder** control named **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.

- Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

```
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
```

- Save and close the **Register.aspx** Web Form.

▶ Task 8: Modify the Login Web Form

- Open the **Login** Web Form.
- Set the page title as **Contoso Customer Management – User Login**.
- Save and close the **Login** Web Form.

▶ Task 9: Enable Forms authentication

- Open the **web.config** file in the project root folder.
- Add to the web.config file an **authentication** element that specifies **Forms** authentication. Place the authentication element in the **system.web** element.

```
<authentication mode="Forms">
</authentication>
```

- Add to the **authentication** element a **forms** element that specifies the value of the **loginUrl** attribute as **~/Account/Login.aspx**.

```
<forms loginUrl="~/Account/Login.aspx" />
```

- Save the **web.config** file.

▶ Task 10: Test the Login controls

- Stop the ASP.NET Development Server by using the ASP.NET Development Server icon in the System tray.
- Run the application.
- View the **Login** controls.

Note: Notice the Login controls at the upper-right corner of the window, which now display a login link. No user is currently signed in, because of the Forms authentication.

- Open the **Login** Web Form by clicking the **Login** link.
- Close Internet Explorer.

Results: After completing this exercise, you will have added Login controls to the master page, added Account files and folders from a sample Web site, updated the Account Web Forms to match the current master page, updated the Login Web Form, and enabled Forms authentication.

Exercise 2: Implementing Authorization

The main tasks for this exercise are as follows:

1. Set up the connection to the membership database.
2. Disallow anonymous access.
3. Allow access to the Styles folder.
4. Create the default database for application services.
5. Test the anonymous access.
6. Enable the security trimming of the site map.
7. Disallow access to the Import Countries page.
8. Allow access to the Customers, Countries, and Help menu items.
9. Disallow access to the Import Countries menu item.
10. Test the access to the Import Countries menu item.
11. Add student to the Admins role.
12. Test the Import Country access.

▶ Task 1: Set up the connection to the membership database

- Add a new connection string named **ApplicationServices** to the web.config file by adding an **add** element in the **connectionStrings** element.

```
<add name="ApplicationServices" connectionString="Data
Source=.\SQLEXPRESS;Integrated
Security=True;AttachDBFilename=|DataDirectory|\ASPNETDB.MDF;User
Instance=true" providerName="System.Data.SqlClient" />
```

- Save the **web.config** file.

▶ Task 2: Disallow anonymous access

- Add an **authorization** element to the web.config file, after the **authentication** element.

```
<authorization>
</authorization>
```

- Add a self-closing **deny** element to the authorization element that disallows all anonymous users.

```
<deny users="?" />
```

- Save the **web.config** file.

▶ Task 3: Allow access to the Styles folder

- Add to the **configuration** element a **location** element to the web.config file with a **path** attribute value of **Styles**.

```
<location path="Styles">
</location>
```

- Add a **system.web** element to the web.config file, in the **location** element.

```
<system.web>
</system.web>
```

- Add an **authorization** element to the web.config file, in the **system.web** element.

```
<authorization>
</authorization>
```

- Add an **allow** element to the web.config file with a **users** attribute value of *, in the **authorization** element.

```
<allow users="*"/>
```

- Save the **web.config** file.

▶ Task 4: Create the default database for application services

- In Solution Explorer, click the **ASP.NET Configuration** button.

Note: This step will take a while.

- In the ASP.Net Web Administration- Windows Internet Explorer window, in the Home tab, click **Security**.

- In the **Security** tab, click **Create user**.
- In the **Security** tab, in the **Create user** section, complete the following settings, and then click **Create User**:
  - User Name: **student**
  - Password: **Pa$$w0rd**
  - Confirm Password: **Pa$$w0rd**
  - E-mail: **student@contoso.com**
  - Security Question: **Contoso Founder**
  - Security Answer: **John Contoso**
- In the ASP.Net Web Administration- Windows Internet Explorer window, click the **Close** button.
- In Solution Explorer, click the **Refresh** button, and then expand **App_Data**.

Note: Notice that the App_Data folder now contains the default ASPNETDB.MDF database file that is used for the application services.

▶ Task 5: Test the anonymous access

- Run the application.
- View the **Login** Web Form.

Note: Notice that the Login Web Form displays instead of the Default Web Form. This is because you have not yet logged in to the Web site.

- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- View the **Default** Web Form.

Note: Notice that the Default Web Form displays after redirecting you from the Login Web Form.

- Close Internet Explorer.

▶ Task 6: Enable the security trimming of the site map

- Add a **siteMap** element to the **system.web** element in the web.config file, with a **defaultProvider** attribute value of **AspNetXmlSiteMapProvider** and an **enabled** attribute value of **true**.

```
<siteMap defaultProvider="AspNetXmlSiteMapProvider"
enabled="true">
</siteMap>
```

- Add a **providers** element to the web.config file, in the **siteMap** element.

```
<providers>
</providers>
```

- Remove the default **AspNetXmlSiteMapProvider** provider element from the web.config file by using a self-closing **remove** element.

```
<remove name="AspNetXmlSiteMapProvider"/>
```

- Add a new **AspNetXmlSiteMapProvider** element to the web.config file by using a self-closing **add** element with a **type** attribute value of **System.Web.XmlSiteMapProvider, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a**, a **siteMap** attribute value of **web.sitemap**, and a **securityTrimmingEnabled** attribute value of **true**.

```
<add name="AspNetXmlSiteMapProvider"
    type="System.Web.XmlSiteMapProvider, System.Web,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
    siteMapFile="web.sitemap"
    securityTrimmingEnabled="true" />
```

- Save the **web.config** file.

▶ Task 7: Disallow access to the Import Countries page

- Add a **location** element to the web.config file **configuration** element, with a **path** attribute value of **ImportCountries.aspx**.

```
<location path="ImportCountries.aspx">
</location>
```

- Add a **system.web** element to the web.config file, in the **location** element.

```
<system.web>
</system.web>
```

- Add an **authorization** element to the web.config file, in the **system.web** element.

```
<authorization>
</authorization>
```

- Add an **allow** element to the web.config file **authorization** element with a **roles** attribute value of **Admins**.

```
<allow roles="Admins"/>
```

- Add a **deny** element to the web.config file with a **users** attribute value of *, in the **authorization** element.

```
<deny users="*"/>
```

- Save and close the **web.config** file.

▶ Task 8: Allow access to the Customers, Countries, and Help menu items

- Open the **web.sitemap** file.
- Set the **roles** attribute of the **Customers siteMapNode** element to a value of *.

```
roles="*"
```

- Set the **roles** attribute of the **Countries siteMapNode** element to a value of *.

```
roles="*"
```

- Set the **roles** attribute of the **Help siteMapNode** element to a value of *.

```
roles="*"
```

▶ Task 9: Disallow access to the Import Countries menu item

- Set the **roles** attribute of the **Import siteMapNode** element to a value of **Admins**.

```
roles="Admins"
```

- Save and close the **web.sitemap** file.

▶ Task 10: Test the access to the Import Countries menu item

- Stop the ASP.NET Development Server by using the icon in the System tray.
- Run the application.
- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- Notice that the **Import** menu item is missing.

Note: Notice that the **Import** menu item is hidden, because the user student is not a member of the Admins role.

- Browse to the **ImportCountries.aspx** Web Form by using the URL **http://localhost:1110/CustomerManagement/ImportCountries.aspx**.

Note: Notice that you are redirected to the login page, because you do not have access to the **ImportCountries** page.

- Close Internet Explorer.

▶ Task 11: Add student to the Admins role

- Open the ASP.NET Web Site Administration tool.
- Enable roles by using the **Security** tab.
- Create an Admins role.
- Add a student user to the Admins role.
- Close Internet Explorer.

▶ Task 12: Test the Import Country access

- Stop the ASP.NET Development Server by using the icon in the System tray.
- Run the application.
- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- View the **Import** menu item.

Note: Notice the Import menu item now displays, because the user, Student, is a member of the **Admins** role.

- Open the **Import Countries** page.

Note: Notice that the Import Countries page now displays.

- Close Internet Explorer.
- Close Microsoft Visual Studio 2010.

Results: After completing this exercise, you will have set up the connection to the membership database, disallowed anonymous access to the Web site, tested the anonymous access, enabled security trimming of the site map, and tested the access to the Import Country menu item and page.

Exercise 3: Protecting Configuration File

The main tasks for this exercise are as follows:

1. Grant the IIS Application Pool Identity access to the RSA key container.
2. Copy the Web site.
3. Test the deployed Web site.
4. Encrypt the connectionStrings section in the configuration file.
5. Test the deployed Web site.

▶ Task 1: Grant the IIS Application Pool Identity access to the RSA key container

- Open the Visual Studio 2010 command prompt as an Administrator.
- Grant the NETWORK SERVICE account access to the default machine-level **NetFrameworkConfigurationKey** RSA key container by running the following command from the Visual Studio 2010 Command Prompt.

```
aspnet_regiis -pa "NetFrameworkConfigurationKey"
 "NT AUTHORITY\NETWORK SERVICE"
```

▶ Task 2: Copy the Web site

- Open Microsoft® Visual Studio® 2010 as an Administrator.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M15\CS folder.
- Open the Copy Web Site tool.
- Connect to the **CM** application on the **Default Web Site** on the local Internet Information Services (IIS).
- Copy the source files to the destination site.
- Close the Copy Web Site tool.

▶ Task 3: Test the deployed Web site

- Open Internet Explorer and enter the URL **http://localhost:1112/CM**.
- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- Close Internet Explorer.

▶ Task 4: Encrypt the connectionStrings section in the configuration file

- Encrypt the **connectionStrings** section of the web.config file for the deployed Web site, which has an application name of **CM**, by running the following command from the Visual Studio Command Prompt (2010).

```
aspnet_regiis -pef "connectionStrings" "C:\inetpub\wwwroot\CM"
```

- Close the Visual Studio 2010 Command Prompt.
- In Windows Explorer, open the **web.config** file from the folder **C:\inetpub\wwwroot\CM**.

- In Visual Studio 2010, notice the **connectionStrings** element, which is now encrypted, and looks similar to the following markup.

```
<connectionStrings
configProtectionProvider="RsaProtectedConfigurationProvider">
  <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
    xmlns="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <KeyName>Rsa Key</KeyName>
        </KeyInfo>
        <CipherData>

<CipherValue>KmUnZDOmmWvygLNCaQbDl6Aj3/RX8peI9O/WAJH8A91RYL3N7+caF
u1fnD9gcxAY04Jl3ERKBceXRATRXdDp72uqAm9TQvxAx3OGV79hYqKD44zY3SrjcGw
1Wp+iDYB7K+GlzMsOwn4SV/C5P6/DNEp9EFhpouV75u/OPzAOsag=</CipherValue
>
        </CipherData>
      </EncryptedKey>
    </KeyInfo>
    <CipherData>

<CipherValue>YU4c3gvwctn4dZAf/9aTqX2Zl6SOuUIWVnoYZO4aCVGM7mLpnt4uJ
OddwzdxOfSWaPLLLivXmE5RJ5ea/8nCM54UYniYiKPkVqptTphkLW5vcQK1typHs75
DHMuvzr4UO321IS8fNK9WUAJL5dJCrOQzuOBYgxHVRdLwJ5JbwgpRIjxAJSqtQfh7W
pASi6iRXHdfTXKJpQUtE9hGcRz/mIvj4QQ9I4sno//aFYcnEbCZtYOZMHtoOj1ukUW
qxQP3f5Cl4WxDq6MHDTwTigeHKJ9sYKxkRlGQKJZW131xUF+BHdy43K5/eDtOtCaCQ
vUjIOOI8eka7yauiHlp9hAsuUHwcyQnF4OXqhOjBAqLgHiJBOtlOEg4pLV/k2NYk7V
BQ3sqQSxNeQJDiD87TApFE6+FddnOFjxvttuRVmjjPdG5cdsva4PgyEv7+I4OySTru
xdDT6zV+lvgfza2SxvfkwleW6KnxGRCU4DaiIBrcefNvbGo+gvVeiZN2qsaH9J4nA/
lfkpn2mTUUF8I3YZWMThmiWYnAyL3xy/6MMJJ+oeGUffouvMOpu7H/JNEOdEghiXj/
ybQr4g=</CipherValue>
    </CipherData>
  </EncryptedData>
</connectionStrings>
```

- Close the web.config file.

▶ Task 5: Test the deployed Web site

- Open Internet Explorer, and enter the URL **http://localhost:1112/CM**.
- Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
- Show all customers.
- Close Internet Explorer.
- Close Microsoft Visual Studio 2010.

Task 6: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

Result: After completing this exercise, you will have encrypted sections of a web.config file, and deployed the Web site to IIS.

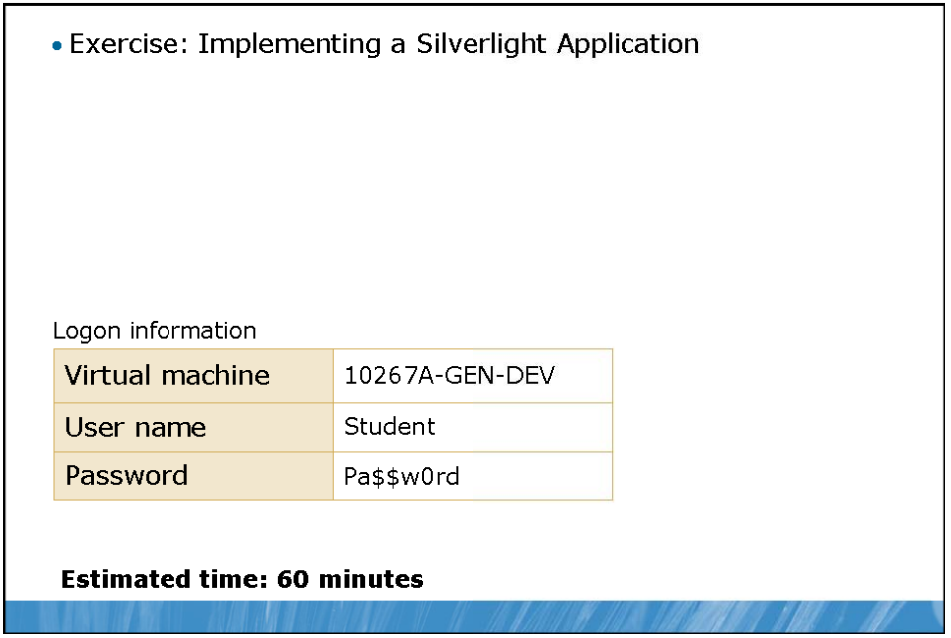
Note: The answers to the exercises are on the Course Companion CD.

Module 16

Lab Instructions: Implementing Advanced Technologies Supported by Microsoft® Visual Studio® 2010 for Web Development (Visual Basic)

Contents:

Lab: Implementing Advanced Technologies Supported by Microsoft Visual Studio 2010 for Web Development



Note: You can perform tasks in this lab by using either Visual Basic or Visual C#. If you are using Visual Basic, refer to the steps provided in Section 1 of the lab page. If you are using Visual C#, refer to the steps provided in Section 2 of the lab page.

Introduction

In this lab, you will create and test a new Microsoft ADO.NET Data Service. In addition, you will create a Silverlight application and host the application by using the existing CustomerManagement ASP.NET Web Site.

Objectives

After completing this lab, you will be able to:

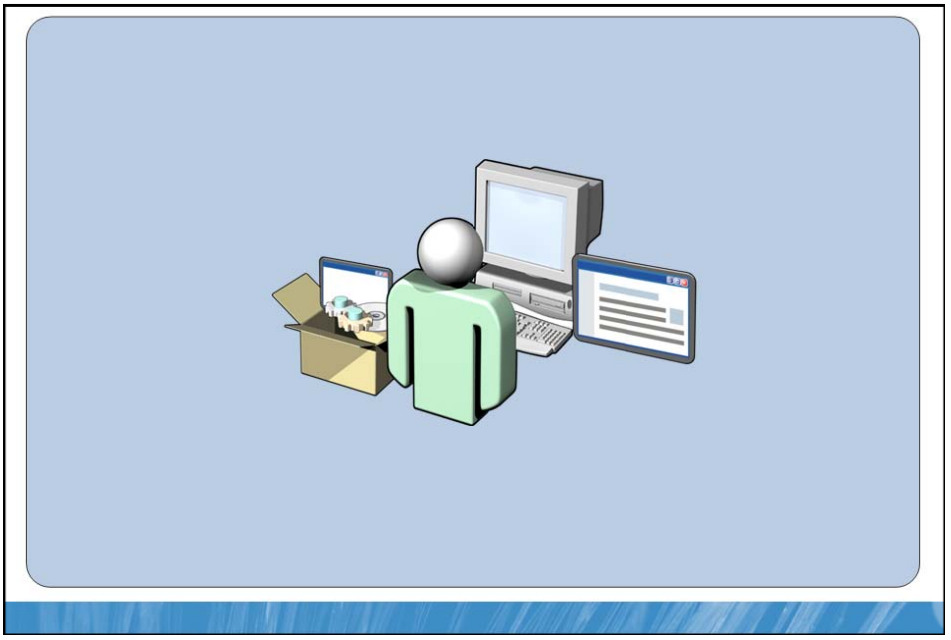
- Implement a Silverlight application.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses .NET applications to create, customize, and manage its customer information. Your organization has created a Web site to manage customer data and services in ASP.NET.

The senior developer has explained the new technologies that have been released, such as Silverlight 4.0, and wants you to use this technology to complement the existing ASP.NET Web site of your organization. To do this, you need to understand how to use Silverlight with ASP.NET to add media elements to a Web application with little coding, and how to use the Silverlight application as a new front-end with media.

Section 1: Visual Basic

Exercise: Implementing a Silverlight Application

The main tasks for this exercise are as follows:

1. Open an existing Web site.
2. Set the start page.
3. Add a Silverlight project.
4. Add a media file to the Silverlight application.
5. Add grid row definitions to the **Grid** control.
6. Add grid column definitions to the **Grid** control.
7. Add the **TextBlock** control to the **Grid** control.
8. Add a **Button** control to the **Grid** control.
9. Add a **MediaElement** control to the **Grid** control.
10. Add resources definition to the **Grid** control.
11. Apply static styles to the **TextBlock** control.
12. Add an Event Handler for the **Button** click event.
13. Test the Silverlight application.

▶ Task 1: Open an existing Web site

- Log on to **10267A-GEN-DEV** as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M16\VB folder.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.

▶ Task 3: Add a Silverlight project

- Add a Silverlight project to the **CustomerManagement** solution that is hosted by the existing **CustomerManagement** Web site. Use the default settings for creating the Silverlight project.

▶ Task 4: Add a media file to the Silverlight application

- Add the file **D:\Labfiles\Starter\M16\Robotica_720.wmv**, to the Silverlight application.
- Add the media file to the application package as a resource.

▶ Task 5: Add grid row definitions to the Grid control

- In the MainPage.xaml document, add a **Grid.RowDefinitions** element to the existing **Grid** control.

```
<Grid.RowDefinitions>
</Grid.RowDefinitions>
```

- Add a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **50**.

```
<RowDefinition Height="50"/>
```

- Append a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **32**.

```
<RowDefinition Height="32"/>
```

- Append a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **Auto**.

```
<RowDefinition Height="Auto"/>
```

- Save the **MainPage.xaml** file.

▶ Task 6: Add grid column definitions to the Grid control

- Add a **Grid.ColumnDefinitions** element to the existing **Grid** control.

```
<Grid.ColumnDefinitions>
</Grid.ColumnDefinitions>
```

- Add a **ColumnDefinition** element to the **Grid.ColumnDefinitions** element with a **Width** attribute value of **Auto**.

```
<ColumnDefinition Width="Auto" />
```

- Save the **MainPage.xaml** file.

▶ Task 7: Add the TextBlock control to the Grid control

- In the MainPage.xaml document, append a **TextBlock** control to the **Grid** control.

Note: The **TextBlock** control must have a **Text** attribute value of **Contoso Customer Management**, and it must be displayed in the first row.

```
<TextBlock Text="Contoso Customer Management" Grid.Row="0" />
```

- Save the **MainPage.xaml** file.

▶ Task 8: Add a Button control to the Grid control

- In the MainPage.xaml document, append a **Button** control to the **Grid** control.

Note: The **Button** control must have a **Content** attribute value of **Play Media**, a **Margin** attribute value of **10,3,10,3**; it must be 75 pixels wide and 25 pixels high, aligned to the left, and it must be displayed in the second row.

```
<Button Width="75" Height="25" HorizontalAlignment="Left"
Content="Play Media" Grid.Row="1" Margin="10,3,10,3" />
```

- Name the **Button** control as **PlayButton** by adding an **x:Name** attribute to the opening **Button** tag.

```
x:Name="PlayButton"
```

- Save the MainPage.xaml file.

▶ Task 9: Add a MediaElement control to the Grid control

- In the MainPage.xaml document, append a **MediaElement** control to the **Grid** control.

Note: The **MediaElement** control must have a **Source** attribute value of **Robotica_720.wmv**, an **AutoPlay** attribute value of **False**, and a **Margin** attribute value of **10,3,10,3**; it must be vertically aligned to the top of the **Grid** control and must be displayed in the third row.

```
<MediaElement AutoPlay="False" Source="Robotica_720.wmv"
Grid.Row="2" Margin="10,3,10,3" VerticalAlignment="Top" />
```

- Name the **MediaElement** control **MainMediaElement**, by adding an **x:Name** attribute to the opening **MediaElement** tag.

```
x:Name="MainMediaElement"
```

- Save the **MainPage.xaml** file.

▶ Task 10: Add a resources definition to the Grid control

- Add a **Grid.Resources** element at the top of the existing **Grid** control in the MainPage.xaml document.

```
<Grid.Resources>
</Grid.Resources>
```

- Add a **Style** element to the **Grid.Resources** element with a **TargetType** attribute value of **TextBlock**.

```
<Style TargetType="TextBlock">
</Style>
```

- Name the **Style** element **AppTitleStyle**, by adding an **x:Key** attribute to the opening **Style** tag.

```
x:Key="AppTitleStyle"
```

- Add a **Setter** element to the **Style** element with a **Property** attribute value of **FontFamily**, and a **Value** attribute value of **Trebuchet MS**, **Arial**, **sans-serif**.

```
<Setter Property="FontFamily" Value="Trebuchet MS, Arial, sans-
serif" />
```

- Append a **Setter** element to the **Style** element with a **Property** attribute value of **FontSize**, and a **Value** attribute value of **30**.

```
<Setter Property="FontSize" Value="30" />
```

- Append a **Setter** element to the **Style** element with a **Property** attribute value of **Foreground**, and a **Value** attribute value of **#888888**.

```
<Setter Property="Foreground" Value="#888888" />
```

- Append a **Setter** element to the **Style** element with a **Property** attribute value of **Margin**, and a **Value** attribute value of **10,3,10,3**.

```
<Setter Property="Margin" Value="10,3,10,3" />
```

- Format the markup by pressing the CTRL+K keys, and then pressing the CTRL+D keys.
- Save the **MainPage.xaml** file.

▶ Task 11: Apply static styles to the TextBlock control

- In the MainPage.xaml document, add a **Style** attribute with a value of **{StaticResource AppTitleStyle}** to the **TextBlock** control.
- Save the **MainPage.xaml** file.

▶ Task 12: Add an Event Handler for the Button click event

- Add a **Click** attribute to the **Button** element with the value of **PlayButton_Click**.
- Save and close the **MainPage.xaml** file.

- Open the code-behind file for the MainPage.xaml file.
- Add code to play the media file when the button is clicked.

```
MainMediaElement.Play()
```

- Save and close the code-behind file.

▶ Task 13: Test the Silverlight application

- Run the application.

Note: Notice the progress indicator showing the download of the Silverlight application package to the Web site.

- Play the media file.
- Close Internet Explorer.

Task 14: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

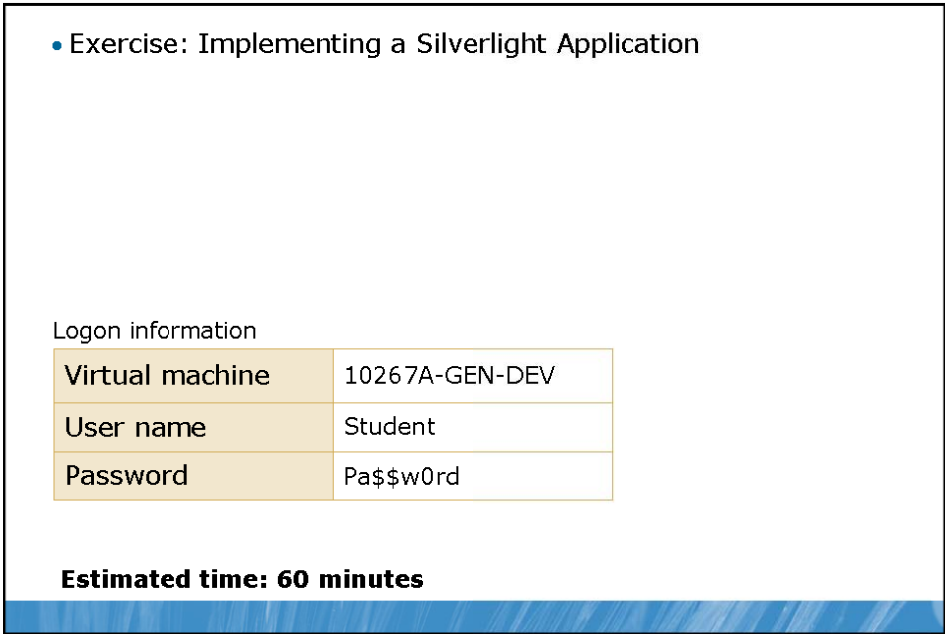
Results: After completing this exercise, you will have added a new Silverlight application to the solution that is hosted by the existing Web site. In addition, you will have added a grid definition, some of the Silverlight controls, and a media file to the existing user control in the Silverlight application. Finally, you will have tested the Silverlight application.

Module 16

Lab Instructions: Implementing Advanced Technologies Supported by Microsoft® Visual Studio® 2010 for Web Development (Visual C#)

Contents:

Lab: Implementing Advanced Technologies Supported by Microsoft Visual Studio 2010 for Web Development



Note: You can perform tasks in this lab by using either Visual Basic or Visual C#. If you are using Visual Basic, refer to the steps provided in Section 1 of the lab page. If you are using Visual C#, refer to the steps provided in Section 2 of the lab page.

Introduction

In this lab, you will create and test a new Microsoft ADO.NET Data Service. In addition, you will create a Silverlight application and host the application by using the existing CustomerManagement ASP.NET Web Site.

Objectives

After completing this lab, you will be able to:

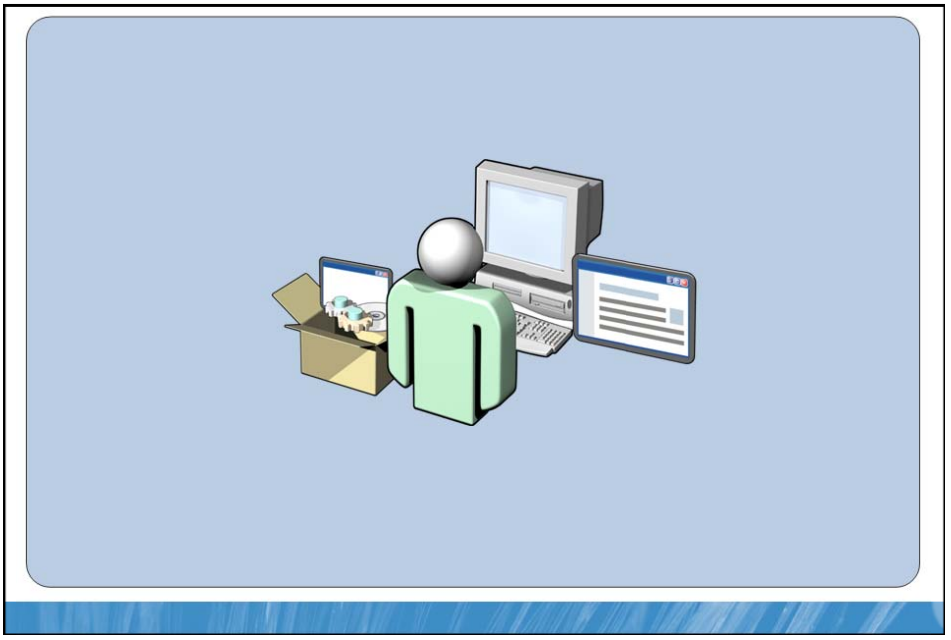
- Implement a Silverlight application.

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must:

- Start the **10267A-GEN-DEV** virtual machine, and then log on by using the following credentials:
  - User name: **Student**
  - Password: **Pa$$w0rd**

Lab Scenario



You are a developer at Contoso, Ltd, which is a large organization with a global customer base. Your organization uses .NET applications to create, customize, and manage its customer information. Your organization has created a Web site to manage customer data and services in ASP.NET.

The senior developer has explained the new technologies that have been released, such as Silverlight 4.0, and wants you to use this technology to complement the existing ASP.NET Web site of your organization. To do this, you need to understand how to use Silverlight with ASP.NET to add media elements to a Web application with little coding, and how to use the Silverlight application as a new front-end with media.

Section 2: Visual C#

Exercise: Implementing a Silverlight Application

The main tasks for this exercise are as follows:

1. Open an existing Web site.
2. Set the start page.
3. Add a Silverlight project.
4. Add a media file to the Silverlight application.
5. Add grid row definitions to the **Grid** control.
6. Add grid column definitions to the **Grid** control.
7. Add the **TextBlock** control to the **Grid** control.
8. Add a **Button** control to the **Grid** control.
9. Add a **MediaElement** control to the **Grid** control.
10. Add a resources definition to the **Grid** control.
11. Apply static styles to the **TextBlock** control.
12. Add an Event Handler for the **Button** click event.
13. Test the Silverlight application.

▶ Task 1: Open an existing Web site

- Log on to 10267A-GEN-DEV as **Student**, with the password, **Pa$$w0rd**.
- Open Microsoft Visual Studio 2010.
- Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M16\CS folder.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.

▶ Task 3: Add a Silverlight project

- Add a Silverlight project to the **CustomerManagement** solution that is hosted by the existing **CustomerManagement** Web site. Use the default settings for creating the Silverlight project.

▶ Task 4: Add a media file to the Silverlight application

- Add the file **D:\Labfiles\Starter\M16\Robotica_720.wmv**, to the Silverlight application.
- Add the media file to the application package as a resource.

▶ Task 5: Add grid row definitions to the Grid control

- In the MainPage.xaml document, add a **Grid.RowDefinitions** element to the existing **Grid** control.

```
<Grid.RowDefinitions>
</Grid.RowDefinitions>
```

- Add a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **50**.

```
<RowDefinition Height="50"/>
```

- Append a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **32**.

```
<RowDefinition Height="32"/>
```

- Append a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **Auto**.

```
<RowDefinition Height="Auto"/>
```

- Save the **MainPage.xaml** file.

▶ Task 6: Add grid column definitions to the Grid control

- Add a **Grid.ColumnDefinitions** element to the existing **Grid** control.

```
<Grid.ColumnDefinitions>
</Grid.ColumnDefinitions>
```

- Add a **ColumnDefinition** element to the **Grid.ColumnDefinitions** element with a **Width** attribute value of **Auto**.

```
<ColumnDefinition Width="Auto" />
```

- Save the **MainPage.xaml** file.

▶ Task 7: Add the TextBlock control to the Grid control

- In the MainPage.xaml document, append a **TextBlock** control to the **Grid** control.

Note: The **TextBlock** control must have a **Text** attribute value of **Contoso Customer Management**, and it must be displayed in the first row.

```
<TextBlock Text="Contoso Customer Management" Grid.Row="0" />
```

- Save the **MainPage.xaml** file.

▶ Task 8: Add a Button control to the Grid control

- In the MainPage.xaml document, append a **Button** control to the **Grid** control.

Note: The **Button** control must have a **Content** attribute value of **Play Media**, a **Margin** attribute value of **10,3,10,3**; it must be 75 pixels wide and 25 pixels high, aligned to the left, and it must be displayed in the second row.

```
<Button Width="75" Height="25" HorizontalAlignment="Left"
Content="Play Media" Grid.Row="1" Margin="10,3,10,3" />
```

- Name the **Button** control as **PlayButton** by adding an **x:Name** attribute to the opening **Button** tag.

```
x:Name="PlayButton"
```

- Save the **MainPage.xaml** file.

▶ Task 9: Add a MediaElement control to the Grid control

- In the MainPage.xaml document, append a **MediaElement** control to the **Grid** control.

Note: The **MediaElement** control must have a **Source** attribute value of **Robotica_720.wmv**, an **AutoPlay** attribute value of **False**, and a **Margin** attribute value of **10,3,10,3**; it must be vertically aligned to the top of the **Grid** control and must be displayed in the third row.

```
<MediaElement AutoPlay="False" Source="Robotica_720.wmv"
    Grid.Row="2" Margin="10,3,10,3" VerticalAlignment="Top" />
```

- Name the **MediaElement** control, **MainMediaElement**, by adding an **x:Name** attribute to the opening **MediaElement** tag.

```
x:Name="MainMediaElement"
```

- Save the **MainPage.xaml** file.

▶ Task 10: Add a resources definition to the Grid control

- Add a **Grid.Resources** element at the top of the existing **Grid** control in the MainPage.xaml document.

```
<Grid.Resources>
</Grid.Resources>
```

- Add a **Style** element to the **Grid.Resources** element with a **TargetType** attribute value of **TextBlock**.

```
<Style TargetType="TextBlock">
</Style>
```

- Name the **Style** element, **AppTitleStyle**, by adding an **x:Key** attribute to the opening **Style** tag.

```
x:Key="AppTitleStyle"
```

- Add a **Setter** element to the **Style** element with a **Property** attribute value of **FontFamily**, and a **Value** attribute value of **Trebuchet MS**, **Arial**, **sans-serif**.

```
<Setter Property="FontFamily" Value="Trebuchet MS, Arial, sans-
serif" />
```

- Append a **Setter** element to the **Style** element with a **Property** attribute value of **FontSize**, and a **Value** attribute value of **30**.

```
<Setter Property="FontSize" Value="30" />
```

- Append a **Setter** element to the **Style** element with a **Property** attribute value of **Foreground**, and a **Value** attribute value of **#888888**.

```
<Setter Property="Foreground" Value="#888888" />
```

- Append a **Setter** element to the **Style** element with a **Property** attribute value of **Margin**, and a **Value** attribute value of **10,3,10,3**.

```
<Setter Property="Margin" Value="10,3,10,3" />
```

- Format the markup by pressing the CTRL+K keys, and then pressing the CTRL+D keys.
- Save the **MainPage.xaml** file.

▶ Task 11: Apply static styles to the TextBlock control

- In the MainPage.xaml document, add a **Style** attribute with a value of **{StaticResource AppTitleStyle}** to the **TextBlock** control.

```
Style="{StaticResource AppTitleStyle}"
```

- Save the **MainPage.xaml** file.

▶ Task 12: Add an Event Handler for the Button click event

- Add a **Click** attribute to the **Button** element with the value of **PlayButton_Click**.
- Save and close the **MainPage.xaml** file.
- Open the code-behind file for the MainPage.xaml file.
- Add code to play the media file when the button is clicked.

```
MainMediaElement.Play()
```

- Save and close the code-behind file.

▶ Task 13: Test the Silverlight application

- Run the application.
- Play the media file.
- Close Internet Explorer.

Task 14: Turn off the virtual machine and revert the changes

- Turn off the **10267A-GEN-DEV** virtual machine.
- Revert the changes made to the **10267A-GEN-DEV** virtual machine.

Results: After completing this exercise, you will have added a new Silverlight application to the solution that is hosted by the existing Web site. In addition, you will have added a grid definition, some of the Silverlight controls, and a media file to the existing user control in the Silverlight application. Finally, you will have tested the Silverlight application.

Module 2

Lab Answer Key: Creating Web Applications by Using Microsoft® Visual Studio® 2010 and Microsoft .NET–Based Languages (Visual Basic)

Contents:

Lab: Creating Web Applications by Using Microsoft® Visual Studio 2010 and Microsoft .NET–Based Languages

(Visual Basic)

Exercise 1: Creating an ASP.NET Web Site

▶ Task 1: Create an empty ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Create the empty CustomerManagement Web site with the following settings, by using the **New Web Site** dialog box:
   - Template: **Empty ASP.NET Web Site**
   - Name: **CustomerManagement**
   - Location: **File system**
   - Path: **D:\Labfiles\Starter\M2\VB\CustomerManagement**
   - Language: **Visual Basic**

   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **New Web Site**.

   b. In the **New Web Site** dialog box, in the left pane, under **Installed Templates**, click **Visual Basic**.

   c. Ensure that **.NET Framework 4** is selected in the target framework list in the middle pane.

d. In the **New Web Site** dialog box, in the middle pane, click **ASP.NET Empty Web Site**.

e. In the **Web location** list, ensure that **File System** is selected, in the **Web location** text box, type **D:\Labfiles\Starter\M2\VB\CustomerManagement**, and then click **OK**.

▶ Task 2: Use a static port with the ASP.NET Development server

1. In Solution Explorer, click **D:\Labfiles\Starter\M2\VB\CustomerManagement**.
2. In the Properties window, in the **Use dynamic ports** list, click **False**.
3. In the Properties window, in the **Port number** box, type **1111**, and then press ENTER.

> **Note:** It may take a few seconds before the **Port number** property is ready for editing after setting the **Use dynamic ports** property.

▶ Task 3: Save the Solution file

- In Solution Explorer, click Solution '**CustomerManagement' (1 project)**, and then save the solution file as **D:\Labfiles\Starter\M2\VB\CustomerManagement.sln**, by using the **Save As** command on the **File** menu.

  a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save CustomerManagement.sln As**.

  b. In the **Save File As** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M2\VB\CustomerManagement.sln**, and then click **Save**.

▶ Task 4: Add an existing CSS file to the Web site

1. Create a folder named **Styles** in the CustomerManagement Web site.
   a. Right-click the Web site in Solution Explorer, and then click **New Folder**.
   b. In Solution Explorer, right-click **D:\Labfiles\Starter\M2\VB \CustomerManagement**, click **New Folder**, type **Styles**, and then press ENTER.
2. Add the D:\Labfiles\Starter\M2\Styles\ Site.css file to the Styles folder, by using the **Add Existing Item** dialog box.
   a. In Solution Explorer, right-click the **Styles** folder, and then click **Add Existing Item**.
   b. In the **Add Existing Item – D:\Labfiles\Starter\M2\VB \CustomerManagement** dialog box, navigate to **D:\Labfiles \Starter\M2**, select the **Site.css** file, and then click **Add**.

Note: You have now added the existing Site.css file to the CustomerManagement Web site.

Exercise 2: Adding and Configuring Server Controls in Web Forms

▶ Task 1: Add a default Web Form to the Web site

- Add the Default.aspx Web Form to the CustomerManagement Web site by using the **Add New Item** dialog box.

  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M2\VB \CustomerManagement**, and then click **Add New Item**.

  b. In the **Add New Item – D:\Labfiles\Starter\M2\VB \CustomerManagement** dialog box, in the Templates pane in the middle of the dialog box, click **Web Form**, in the **Name** box, type **Default.aspx**, and then click **Add**.

> **Note:** The Default Web Form displays in Source view, where you can notice the elements such as form, div, and body, are empty.

▶ Task 2: Close Visual Studio 2010

1. Save modified files.
   - On the **File** menu, click **Save All**.
2. Close Visual Studio 2010.
   - In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 3: Add the application title to the default Web Form

1. Open Microsoft Visual Studio 2010 as an administrator, by using the **Run as administrator** command on the context menu.

   a. On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, right-click **Microsoft Visual Studio 2010**, and then click **Run as administrator**.

   b. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.

2. Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M2\VB folder, by using the **Open Project** dialog box.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M2\VB\CustomerManagement.sln**, and then click **Open**.
3. Add a **Literal** control from the Toolbox to the **div** element in the Default.aspx window, and set its **Text** attribute to **Customer Management**, and **ID** property to **AppTitleLiteral**, by using the Properties window.
   a. In the Default.aspx window, place the cursor within the **div** element.
   b. Point to **Toolbox**, expand **Standard**, and then double-click the **Literal** control.
   c. In the Default.aspx window, ensure that the **Literal** control is selected.
   d. In the Properties window, in the **Text** property, type **Customer Management**.
   e. In the Properties window, change the **(ID)** property type to **AppTitleLiteral**.
4. Save the changes, and view the default Web Form in a Web browser by using the **View in Browser** context menu command.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, right-click anywhere, and then click **View in Browser**.

> **Note:** You can now view the text that you have entered in the **Literal** control.

   c. In the http://localhost:1111/CustomerManagement/Default.aspx - Internet Explorer window, click the **Close** button.

▶ Task 4: Set the control properties of the default Web Form

1. Open the default Web Form in Design view.
   - In the Default.aspx window, click **Design**.
2. Set the **Visible** property of the **Literal** control to **False** by using the Properties window, and then save the changes.
   a. In the Default.aspx window, click the **Literal** control.
   b. In the Properties window, change the **Visible** property to **False**.
   c. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Default.aspx**.
3. View the Web Form in the browser, and then view the source rendered to the browser, by using the **Source** command on the **View** menu of Internet Explorer.
   a. In the Default.aspx window, right-click anywhere, and then click **View in Browser**.

> **Note:** The **Visible** property of a control helps hide or show a control. In this instance, notice that the application title is not visible.

   b. In the http://localhost:1111/CustomerManagement/Default.aspx - Internet Explorer window, on the **View** menu, click **Source**.

> **Note:** Notice that **Literal** control is not rendered to the browser, because the **Visible** property was set to **False**.

4. Close the open Internet Explorer windows.
   a. In the http://localhost:1111/CustomerManagement/Default.aspx - Original Source window, click the **Close** button.
   b. In the http://localhost:1111/CustomerManagement/Default.aspx - Internet Explorer window, click the **Close** button.

5. Set the **Visible** property of the **Literal** control to **True**.
   a. In the Default.aspx window, click the **Literal** control.
   b. In the Properties window, change the **Visible** property to **True**.
6. Set the **Styles** property of the **div** element by using the following properties in the **Modify Style** dialog box, which are accessible from the **Style** property in the Property window:
   - Font-family: **Trebuchet MS**
   - Font-size: **22**
   - Color: **Gray**

   a. In the Default.aspx window, click the **div** element.
   b. In the Properties window, click the **Style** box, and then click the ellipsis button.
   c. In the **Modify Style** dialog box, in the **font-family** list, click **Trebuchet MS**.
   d. In the **font-size** box, type **22**, in the **color** list, click the **Gray** button, and then click **OK**.
7. Save the changes, and view the default Web Form in a Web browser.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, right-click anywhere, and then click **View in Browser**.

Note: You can now view the changes that you have made to the **Literal** control.

   c. In the http://localhost:1111/CustomerManagement/Default.aspx - Internet Explorer window, click the **Close** button.

▶ Task 5: Apply the predefined style to the default Web Form

1. View the changed page in Source view.
   - In the Default.aspx window, click **Source**.
2. Add a reference to the **Site.css** file in the **Styles** folder from within the head element by dragging the **Styles/Site.css** file to the Default.aspx window, next to the **title** element.
   - In Solution Explorer, under **D:\Labfiles\Starter\M2\VB \CustomerManagement**, expand **Styles**, and then drag **Site.css** next to the closing tag of the **title** element in the Default.aspx markup.

> **Note:** Notice that the predefined styles are automatically added to the Web Form.

3. Set the **Class** property of the **div** element to **appTitle**, and then remove the specific styles applied for the **Styles** property, by using the Properties window.
   a. In the Default.aspx window, click **Design**, and then click the **div** element.
   b. In the Properties window, change the **Class** property to **appTitle**.
   c. In the Properties window, click the **Style** property, and clear the current text.
4. Save the changes and view the default Web Form in a Web browser.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, right-click anywhere, and then click **View in Browser**.

> **Note:** You can now view the changes that you have made to the **div** element.

   c. In the http://localhost:1111/CustomerManagement/Default.aspx - Internet Explorer window, click the **Close** button.

Exercise 3 Building and Deploying an ASP.NET Web Application

▶ Task 1: Build and deploy the CustomerManagement Web site

1. Build the CustomerManagement Web site, and then verify that the Web site has no errors.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Build** menu, click **Build Web Site**, and then in the Output window, ensure that the **Validation Complete** message appears.
2. Open the Copy Web Site tool by selecting the Web site in Solution Explorer, and then using the Web site menu.
   - In Solution Explorer, right-click **D:\Labfiles\Starter\M2\VB\CustomerManagement**, and then on the **Website** menu, click **Copy Web Site**.
3. Connect to the local Internet Information Services (IIS) in the Copy Web Site tool.
   a. In the Copy Web D:\Labfiles\Starter\M2\VB\CustomerManagement\ window, click **Connect**.
   b. In the **Open Web Site** dialog box, click **Local IIS**.
4. Create a new virtual directory below the Default Web Site, by selecting **Default Web Site**, and then clicking the **Create New Virtual Directory** button. Use the following settings:
   - Alias name: **CustomerManagement**
   - Folder: **C:\inetpub\wwwroot\CustomerManagement**

   a. In the **Open Web Site** dialog box, in the **Select the Web site you want to open** box, click **Default Web Site**, and then in the upper-right corner, click the **Create New Virtual Directory** button.
   b. In the **New Virtual Directory** dialog box, in the **Alias name** box, type **CustomerManagement**, in the **Folder** box, type **C:\inetpub\wwwroot\CustomerManagement**, and then click **OK**.
   c. In the Microsoft Visual Studio message box, click **Yes**.

5. Select and open the new virtual directory in the **Open Web Site** dialog box.
   - In the **Open Web Site** dialog box, in the **Select the Web site you want to open** box, ensure that the **CustomerManagement** is selected, and then click **Open**.
6. Copy the CustomerManagement Web site to the new virtual directory on the local IIS, by selecting all files and folder in the left pane of the Copy Web Site tool, and then clicking the **Copy selected files from source to remote web site** button.
   - In the Copy Web D:\Labfiles\Starter\M2\VB\CustomerManagement\ window, under **Source Web site**, in the **D:\Labfiles\Starter\M2\VB\CustomerManagement** box, select all items, and then click the **Copy selected files from source to remote web site** button.
7. View the deployed Web site in Internet Explorer at the address, **http://localhost:1112/CustomerManagement**.
   a. Open Internet Explorer.
   b. In the Address Bar, type the following address, **http://localhost:1112/CustomerManagement**, and then press ENTER.

> **Note:** You can now view the changes that you have made to the Web Form.

   c. In the http://localhost:1112/CustomerManagement/ –Internet Explorer window, click the **Close** button.
   d. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, click the **Close** button.

▶ Task 2: Turn off the virtual machine and revert the changes.

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 2

Lab Answer Key: Creating Web Applications by Using Microsoft® Visual Studio® 2010 and Microsoft .NET–Based Languages (Visual C#)

Contents:

Lab: Creating Web Applications by Using Microsoft® Visual Studio 2010 and Microsoft .NET–Based Languages

(C#)

Exercise 1: Creating an ASP.NET Web Site

▶ Task 1: Create an empty ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Create the empty CustomerManagement Web site with the following settings, by using the **New Web Site** dialog box:
   - Template: **Empty ASP.NET Web Site**
   - Name: **CustomerManagement**
   - Location: **File system**
   - Path: **D:\Labfiles\Starter\M2\CS\CustomerManagement**
   - Language: **Visual C#**

   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **New Web Site**.

   b. In the **New Web Site** dialog box, in the left pane, under **Installed Templates**, click **Visual C#**.

   c. Ensure that **.NET Framework 4** is selected in the target framework list in the middle pane.

d. In the **New Web Site** dialog box, in the middle pane, click **ASP.NET Empty Web Site**.

e. In the **Web location** list, ensure that **File System** is selected, and then in the **Web location** text box, type **D:\Labfiles\Starter\M2\CS\CustomerManagement**, and then click **OK**.

▶ Task 2: Use a static port with ASP.NET Development server

1. In Solution Explorer, click **D:\Labfiles\Starter\M2\CS\CustomerManagement**.
2. In the Properties window, in the **Use dynamic ports** list, click **False**.
3. In the Properties window, in the **Port number** box, type **1110**, and then press ENTER.

> **Note:** It may take a few seconds before the **Port number** property is ready for editing after setting the **Use dynamic ports** property.

▶ Task 3: Save the Solution file

- In Solution Explorer, click Solution '**CustomerManagement' (1 project)**, and then save the solution file as **D:\Labfiles\Starter\M2\CS\CustomerManagement.sln**, by using the **Save As** command on the **File** menu.

  a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save CustomerManagement.sln As**.

  b. In the **Save File As** dialog box, in **the File name** box, type **D:\Labfiles\Starter\M2\CS\CustomerManagement.sln**, and then click **Save**.

▶ Task 4: Add an existing CSS file to the Web site

1. Create a folder named **Styles** in the CustomerManagement Web site. Right-click the Web site in Solution Explorer, and then click **New Folder**.

   - In Solution Explorer, right-click **D:\Labfiles\Starter\M2\CS\CustomerManagement**, click **New Folder**, type **Styles**, and then press ENTER.

2. Add the **D:\Labfiles\Starter\M2\Styles\ Site.css** file to the **Styles** folder, by using the **Add Existing Item** dialog box.

   a. In Solution Explorer, right-click the **Styles** folder, and then click **Add Existing Item**.

   b. In the **Add Existing Item – D:\Labfiles\Starter\M2\CS\CustomerManagement** dialog box, navigate to **D:\Labfiles\Starter\M2**, select the **Site.css** file, and then click **Add**.

Note: You have now added the existing Site.css file to the CustomerManagement Web site.

Exercise 2: Adding and Configuring Server Controls in Web Forms

▶ Task 1: Add a default Web Form to the Web site

- Add the Default.aspx Web Form to the CustomerManagement Web site by using the **Add New Item** dialog box.

  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M2\CS\CustomerManagement**, and then click **Add New Item**.

  b. In the **Add New Item – D:\Labfiles\Starter\M2\CS\CustomerManagement** dialog box, in the Templates pane in the middle of the dialog box, click **Web Form**. In the **Name** box, type **Default.aspx**, and then click **Add**.

> **Note:** The Default Web Form displays in Source view, where you can see that the elements such as form, div, and body, are empty.

▶ Task 2: Close Visual Studio 2010

1. Save modified files.
   - On the **File** menu, click **Save All**.
2. Close Visual Studio 2010.
   - In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 3: Add the application title to the default Web Form

1. Open Microsoft Visual Studio 2010 as an administrator, by using the **Run as administrator** command on the context menu.

   a. On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, right-click **Microsoft Visual Studio 2010**, and then click **Run as administrator**.

   b. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.

2. Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M2\CS folder, by using the **Open Project** dialog box.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M2\CS\CustomerManagement.sln**, and then click **Open**.
3. Add a **Literal** control from the Toolbox to the **div** element in the Default.aspx window, and set its **Text** attribute to **Customer Management** and **ID** property to **AppTitleLiteral**, by using the Properties window.
   a. In the Default.aspx window, place the cursor within the **div** element.
   b. Point to **Toolbox**, expand **Standard**, and then double-click the **Literal** control.
   c. In the Default.aspx window, ensure that the **Literal** control is selected.
   d. In the Properties window, in the **Text** property, type **Customer Management**.
   e. In the Properties window, change the **(ID)** property type to **AppTitleLiteral**.
4. Save the changes, and view the default Web Form in a Web browser, by using the **View in Browser** context menu command.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, right-click anywhere, and then click **View in Browser**.

> **Note:** You can now view the text that you have entered in the **Literal** control.

   c. In the http://localhost:1110/CustomerManagement/Default.aspx - Internet Explorer window, click the **Close** button.

▶ Task 4: Set the control properties of the default Web Form

1. Open the default Web Form in Design view.
   - In the Default.aspx window, click **Design**.
2. Set the **Visible** property of the **Literal** control to **False** by using the Properties window, and then save the changes.
   a. In the Default.aspx window, click the **Literal** control.
   b. In the Properties window, change the **Visible** property to **False**.
   c. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Default.aspx**.
3. View the Web Form in the browser, and then view the source rendered to the browser, by using the **Source** command on the **View** menu of Internet Explorer.
   a. In the Default.aspx window, right-click anywhere, and then click **View in Browser**.

> **Note:** The **Visible** property of a control helps hide or show a control. In this instance, notice that the application title is not visible.

   b. In the http://localhost:1110/CustomerManagement/Default.aspx - Internet Explorer window, on the **View** menu, click **Source**.

> **Note:** Notice that Literal control is not rendered to the browser, because the **Visible** property was set to **False**.

4. Close the open Internet Explorer windows.
   a. In the localhost:1110/CustomerManagement/Default.aspx - Original Source window, click the **Close** button.
   b. In the http://localhost:1110/CustomerManagement/Default.aspx - Internet Explorer window, click the **Close** button.

5. Set the **Visible** property of the **Literal** control to **True**.
   a. In the Default.aspx window, click the **Literal** control.
   b. In the Properties window, change the **Visible** property to **True**.
6. Set the **Styles** property of the **div** element by using the following properties in the **Modify Style** dialog box, which are accessible from the **Style** property in the Property window:
   - Font-family: **Trebuchet MS**
   - Font-size: **22**
   - Color: **Gray**

   a. In the Default.aspx window, click the **div** element.
   b. In the Properties window, click the **Style** box, and then click the ellipsis button.
   c. In the **Modify Style** dialog box, in the font-family list, click **Trebuchet MS**.
   d. In the font-size box, type **22**, in the color list, click the **Gray** button, and then click **OK**.
7. Save the changes, and view the default Web Form in a Web browser.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, right-click anywhere, and then click **View in Browser**.

Note: You can now view the changes that you have made to the **Literal** control.

   c. In the http://localhost:1110/CustomerManagement/Default.aspx - Internet Explorer window, click the **Close** button.

▶ Task 5: Apply the predefined style to the default Web Form

1. View the changed page in Source view.
   - In the Default.aspx window, click **Source**.
2. Add a reference to the **Site.css** file in the **Styles** folder from within the head element by dragging the **Styles/Site.css** file to the Default.aspx window, next to the **title** element.
   - In Solution Explorer, under **D:\Labfiles\Starter\M2\CS \CustomerManagement**, expand **Styles**, and then drag **Site.css** next to the closing tag of the **title** element in the Default.aspx markup.

> **Note:** Notice that the predefined styles are automatically added to the Web Form.

3. Set the **Class** property of the **div** element to **appTitle**, and then remove the specific styles applied for the **Styles** property, by using the Properties window.
   a. In the Default.aspx window, click **Design**, and then click the **div** element.
   b. In the Properties window, change the **Class** property to **appTitle**.
   c. In the Properties window, click the **Style** property, and clear the current text.
4. Save the changes, and view the default Web Form in a Web browser.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, right-click anywhere, and then click **View in Browser**.

> **Note:** You can now view the changes that you have made to the **div** element.

   c. In the http://localhost:1110/CustomerManagement/Default.aspx - Internet Explorer window, click the **Close** button.

Exercise 3 Building and Deploying an ASP.NET Web Application

▶ Task 1: Build and deploy the CustomerManagement Web site

1. Build the CustomerManagement Web site, and then verify that the Web site has no errors.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Build** menu, click **Build Web Site**, and then in the Output window, ensure that the Validation Complete message appears.
2. Open the Copy Web Site tool, by selecting the Web site in Solution Explorer and then using the Web site menu.
   - In Solution Explorer, right-click **D:\Labfiles\Starter\M2\CS\CustomerManagement**, and then on the **Web site** menu, click **Copy Web Site**.
3. Connect to the local Internet Information Services (IIS) in the Copy Web Site tool.
   a. In the Copy Web D:\Labfiles\Starter\M2\CS\CustomerManagement\ window, click **Connect**.
   b. In the **Open Web Site** dialog box, click **Local IIS**.
4. Create a new virtual directory below the Default Web Site, by selecting **Default Web Site**, and then clicking the **Create New Virtual Directory** button. Use the following settings:
   - Alias name: **CustomerManagement**
   - Folder: **C:\inetpub\wwwroot\CustomerManagement**

   a. In the **Open Web Site** dialog box, in the **Select the Web site you want to open**. box, click **Default Web Site**, and then in the upper-right corner, click the **Create New Virtual Directory** button.
   b. In the **New Virtual Directory** dialog box, in the **Alias name** box, type **CustomerManagement**, in the **Folder** box, type **C:\inetpub\wwwroot\CustomerManagement**, and then click **OK**.
   c. In the Microsoft Visual Studio message box, click **Yes**.

5. Select and open the new virtual directory in the **Open Web Site** dialog box.
   - In the **Open Web Site** dialog box, in the **Select the Web site you want to open** box, ensure that the **CustomerManagement** is selected, and then click **Open**.
6. Copy the CustomerManagement Web site to the new virtual directory on the local IIS, by selecting all files and folder in the left pane of the Copy Web Site tool, and then clicking the **Copy selected files from source to remote web site** button.
   - In the Copy Web D:\Labfiles\Starter\M2\CS\CustomerManagement\ window, under **Source Web site**, in the **D:\Labfiles\Starter\M2\CS\CustomerManagement** box, select all items, and then click the **Copy selected files from source to remote web site** button.
7. View the deployed Web site in Internet Explorer at the address **http://localhost:1112/CustomerManagement**.
   a. Open Internet Explorer.
   b. In the Address Bar, type the address **http://localhost:1112/CustomerManagement**, and then press ENTER.

> **Note:** You can now view the changes that you have made to the Web Form.

   c. In the http://localhost:1112/CustomerManagement/ −Internet Explorer window, click the **Close** button.
   d. In the CustomerManagement − Microsoft Visual Studio (Administrator) window, click the **Close** button.

▶ Task 2: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V Manager™, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 3

Lab Answer Key: Creating a Microsoft® ASP.NET Web Form (Visual Basic)

Contents:

Lab: Creating a Microsoft® ASP.NET Web Form

Exercise 1: Creating a Web Form

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M3\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M3\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Add a Web Form to the Web site

- Add the InsertCustomer.aspx Web Form to the CustomerManagement Web site, by using the **Add New Item** dialog box.
  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M3\VB \CustomerManagement**, and then click **Add New Item**.
  b. In the **Add New Item – D:\Labfiles\Starter\M3\VB \CustomerManagement** dialog box, in the Templates pane in the middle of the dialog box, click **Web Form**, in the **Name** box, type **InsertCustomer.aspx**, and then click **Add**.

> **Note:** The InsertCustomer Web Form displays in Source view.

Exercise 2: Adding and Configuring Server Controls in a Web Form

▶ Task 1: Add an existing style sheet to the Web Form

1. Reference the **Styles/Site.css** file in the InsertCustomer Web Form, relative to the root folder, by using the Manage Styles window.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **View** menu, click **Manage Styles**.
   b. In the Manage Styles window, click the **Attach Style Sheet** button.
   c. In the **Select Style Sheet** dialog box, in the **Project Folders** list, click **Styles**, in the **Contents of folder** list, click **Site.css**, and then click **OK**.

> **Note:** Notice that a self-closing **link** element is added to the **head** element, just below the **title** element.

   d. In the InsertCustomer.aspx window, modify the **href** attribute of the self-closing link element to include a tilde (~) and a forward slash (/) character at the beginning of the path.

```
<link href="~/Styles/Site.css" rel="stylesheet"
type="text/css" />
```

2. Press CTRL+S to save InsertCustomer.aspx.
3. View the styles added in the style sheet by using the Manage Styles window.
   a. In the Manage Styles window, under cascading style sheet (CSS) styles, view all the styles added in the Site.css list.
   b. In the Manage Styles window, click the **Close** button.

▶ Task 2: Create a table-style layout in the Web Form

1. In the InsertCustomer.aspx window, place the cursor between the opening and closing **div** tags.

2. Add a new **div** element from the Toolbox to the existing div element.

```
<div>
    <div>
    </div>
</div>
```

- In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
- In Toolbox, expand **Html**, and then double-click the **div** element.

```
<div>
    <div>
    </div>
</div>
```

3. Add two new div elements from the Toolbox to the newly added **div** element.

```
<div>
    <div>
        <div>
        </div>
        <div>
        </div>
    </div>
</div>
```

a. In the InsertCustomer.aspx window, place the cursor between the opening and closing div tags of the newly added div element.

b. In Toolbox, expand **Html**, and then double-click the **div** element twice.

```
<div>
    <div>
        <div>
        </div>
        <div>
        </div>
    </div>
</div>
```

4. Save the changes to the InsertCustomer Web Form.

- In the CustomerManagement – Microsoft Visual Studio window, click CTRL+S.

5. Use the Apply Styles window to apply the **customerTable** style to the outermost div element. Apply **div.customerTable** from the Contextual Selectors section of the Apply Styles window.

    a. In the InsertCustomer.aspx window, click the opening tag of the outermost **div** element.

    b. In the CustomerManagement – Microsoft Visual Studio window, on the **View** menu, click **Apply Styles**.

    c. In the Apply Styles window, in the **Contextual Selectors** section, under **Site.css**, click **div.customerTable**.

6. Click the opening tag of the **div** element, which is a child element of the **div** element with a **class** attribute value of **customerTable**.

    - In the InsertCustomer.aspx window, click the opening tag of the **div** element, which is a child element of the **div** element with a **class** attribute value of **customerTable**.

7. Create new style in the Apply Styles window, by using the **New Style** button.

    - In the Apply Styles window, click the **New Style** button.

8. Name the new style **div.customerTableRow**, by using the **Selector** box of the **New Style** dialog box.

    - In the **New Style** dialog box, in the **Selector** box, type **div.customerTableRow**.

9. Apply the style to the current document selection, by using the **Apply new style to document selection** check box.

    - In the **New Style** dialog box, select the **Apply new style to document selection** check box.

10. Define the new style in the existing **Styles/Site.css** CSS file.

    - In the **New Style** dialog box, in the **Define in** list, click **Existing style sheet**, and then in the URL list, click **Styles/Site.css**.

11. Set the layout of the new style to **table-row**, by using the display list of the **Layout** category, and then close the **New Style** dialog box.

    - In the **New Style** dialog box, in the **Category** list, click **Layout**, in the **display** list, click **table-row**, and then click **OK**.

12. Use the Apply Styles window to apply the **customerTableLeftCol** style to the first child **div** element of the **div** element with a class attribute value of **customerTableRow**.

    a. In the InsertCustomer.aspx window, click the opening tag of the **div** element, which is the first child element of the **div** element with a **class** attribute value of **customerTableRow**.

    b. In the CustomerManagement – Microsoft Visual Studio window, on the **View** menu, click **Apply Styles**.

    c. In the Apply Styles window, in the **Contextual Selectors** section, under **Site.css**, click **div.customerTableLeftCol**.

13. Use the Apply Styles window to apply the **customerTableRightCol** style to the second child **div** element of the **div** element with a class attribute value of **customerTableRow**.

    a. In the InsertCustomer.aspx window, click the opening tag of the **div** element, which is the second child element of the **div** element with a **class** attribute value of **customerTableRow**.

    b. In the CustomerManagement – Microsoft Visual Studio window, on the **View** menu, click **Apply Styles**.

    c. In the Apply Styles window, in the **Contextual Selectors** section, under **Site.css**, click **div.customerTableRightCol**.

14. Add 11 more table rows, by copying the existing **div** element with a **class** attribute value of **customerTableRow**.

    a. In the InsertCustomer.aspx window, select the **div** element with a **class** attribute value of **customerTableRow**, right-click the selection, and then click **Copy**.

```
<div class="customerTableRow">
    <div class="customerTableLeftCol">
    </div>
    <div class="customerTableRightCol">
    </div>
</div>
```

    b. In the InsertCustomer.aspx window, place the cursor after the closing div tag of the **div** element with a **class** attribute value of **customerTableRow** and then press ENTER.

    c. Press CTRL+V 11 times in succession.

15. Append a new **div** element from the Toolbox to the **div** element with a **class** attribute value of **customerTable**, placing the new **div** element immediately before the closing div tag of the **customerTable** div element.

```
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
        </div>
        <div class="customerTableRightCol">
        </div>
    </div>
    ...
    <div>
    </div>
</div>
```

    a. In the InsertCustomer.aspx window, place the cursor after the last closing div tag of the **div** element with a **class** attribute value of **customerTableRow**, and then press ENTER.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, expand **Html**, and then double-click the **div** element.

```
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
        </div>
        <div class="customerTableRightCol">
        </div>
    </div>
    ...
    <div>
    </div>
</div>
```

16. View the InsertCustomer Web Form in Design view.

    - In the InsertCustomer.aspx window, click **Design**.

Note: Notice that two equal-sized columns are added to the **div** element.

17. View the InsertCustomer Web Form in Source view.
    - In the InsertCustomer.aspx window, click **Source**.
18. Use the Apply Styles window to apply the **customerTableFooter** style to the last child **div** element of the div element with **class** attribute value of **customerTable**.
    a. In the InsertCustomer.aspx window, click the opening tag of the **div** element, which is the last child element of the **div** element with a class attribute value of **customerTable**.
    b. In the Apply Styles window, in the **Contextual Selectors** section, under **Site.css**, click **div.customerTableFooter**.
    c. In the Apply Styles window, click the **Close** button.
19. Save the changes to the InsertCustomer Web Form and the Site.css CSS file.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**.
20. View the InsertCustomer Web Form in a Web browser.
    a. In Solution Explorer, under D:\Labfiles\Starter\M3\VB \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

> **Note:** Although you have styled the Web form by using the CSS file, notice that nothing displays. This is because the div elements are empty.

    b. In the http://localhost:1111/CustomerManagement /InsertCustomer.aspx - Windows® Internet Explorer® window, click the **Close** button.

▶ Task 3: Add the server controls to the Web Form and configure their basic properties

1. View the InsertCustomer Web Form in Design view.
   - In the InsertCustomer.aspx window, click **Design**.
2. Add the **Label** control from the Toolbox to the first cell of the left column in the **div** element, change the **(ID)** property to **CustomerFirstNameLabel**, and then set the **Text** property to **First Name:**.
   a. In the InsertCustomer.aspx window, in the first **div** element, click the first cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.
   b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
   c. In Toolbox, expand **Standard**, and then double-click the **Label** control.
   d. In the InsertCustomer.aspx window, click the **Label** control.
   e. In the Properties window, change the **(ID)** property to **CustomerFirstNameLabel**.
   f. In the Properties window, in the **Text** property, type **First Name:**.
3. Add the **TextBox** control from the Toolbox to the first cell of the right column in the **div** element, and change the **(ID)** property to **CustomerFirstNameTextBox**.
   a. In the InsertCustomer.aspx window, in the **div** element, click the first cell of the right column that has **customerTableRightCol** as the value for the **Class** property.
   b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
   c. In Toolbox, under **Standard**, double-click the **TextBox** control.
   d. In the InsertCustomer.aspx window, click the **TextBox** control.
   e. In the Properties window, change the **(ID)** property to **CustomerFirstNameTextBox**.

4. Add the **Label** control from the Toolbox to the second cell of the left column in the **div** element, change the **(ID)** property to **CustomerLastNameLabel**, and then set the **Text** property to **Last Name:**.

    a. In the InsertCustomer.aspx window, in the second **div** element, click the first cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, expand **Standard**, and then double-click the **Label** control.

    d. In the InsertCustomer.aspx window, click the **Label** control.

    e. In the Properties window, change the **(ID)** property to **CustomerLastNameLabel**.

    f. In the Properties window, in the **Text** property, type **Last Name:**.

5. Add the **TextBox** control from the Toolbox to the second cell of the right column in the **div** element, and change the **(ID)** property to **CustomerLastNameTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the second cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

    d. In the InsertCustomer.aspx window, click the **TextBox** control.

    e. In the Properties window, change the **(ID)** property to **CustomerLastNameTextBox**.

6. Add the **Label** control from the Toolbox to the third cell of the left column in the **div** element, change the **(ID)** property to **CustomerAddressLabel**, and then set the **Text** property to **Address:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the third cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, change the **(ID)** property to **CustomerAddressLabel**.

f. In the Properties window, in the **Text** property, type **Address:**.

7. Add the **TextBox** control from the Toolbox to the third cell of the right column in the **div** element, and change the **(ID)** property to **CustomerAddressTextBox**.

a. In the InsertCustomer.aspx window, in the **div** element, click the third cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerAddressTextBox**.

8. Add the **Label** control from the Toolbox to the fourth cell of the left column in the **div** element, change the **(ID)** property to **CustomerZipCodeLabel**, and then set the **Text** property to **Zip Code:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the fourth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, in the **(ID)** property, type **CustomerZipCodeLabel**.

f. In the Properties window, in the **Text** property, type **Zip Code:**

9. Add the **TextBox** control from the Toolbox to the fourth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerZipCodeTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the fourth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

    d. In the InsertCustomer.aspx window, click the **TextBox** control.

    e. In the Properties window, change the **(ID)** property to **CustomerZipCodeTextBox**.

10. Add the **Label** control from the Toolbox to the fifth cell of the left column in the **div** element, change the **(ID)** property to **CustomerCityLabel**, and then set the **Text** property to **City:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the fifth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Label** control.

    d. In the InsertCustomer.aspx window, click the **Label** control.

    e. In the Properties window, change the **(ID)** property to **CustomerCityLabel**.

    f. In the Properties window, in the **Text** property, type **City:**

11. Add the **TextBox** control from the Toolbox to the fifth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCityTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the fifth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerCityTextBox**.

12. Add the **Label** control from the Toolbox to the sixth cell of the left column in the **div** element, change the **(ID)** property to **CustomerStateLabel**, and then set the **Text** property to **State:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the sixth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, change the **(ID)** property to **CustomerStateLabel**.

f. In the Properties window, in the **Text** property, type **State:**

13. Add the **TextBox** control from the Toolbox to the sixth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerStateTextBox**.

a. In the InsertCustomer.aspx window, in the **div** element, click the sixth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerStateTextBox**.

14. Add the **Label** control from the Toolbox to the seventh cell of the left column in the **div** element, change the **(ID)** property to **CustomerCountryLabel**, and then set the **Text** property to **Country:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the seventh cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, for the **(ID)** property, type **CustomerCountryLabel**.

f. In the Properties window, for the **Text** property, type **Country:**

15. Add the **DropDownList** control from the Toolbox to the seventh cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCountryDropDownList**.

a. In the InsertCustomer.aspx window, in the **div** element, click the seventh cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **DropDownList** control.

d. In the Properties window, change the **(ID)** property to **CustomerCountryDropDownList**.

16. Add the **Label** control from the Toolbox to the eighth cell of the left column in the **div** element, change the **(ID)** property to **CustomerPhoneLabel**, and then set the **Text** property to **Phone:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the eighth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, for the **(ID)** property, type **CustomerPhoneLabel**.

f. In the Properties window, for the **Text** property, type **Phone:**

17. Add the **TextBox** control from the Toolbox to the eighth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerPhoneTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the eighth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

    d. In the Properties window, change the **(ID)** property to **CustomerPhoneTextBox**.

18. Add the **Label** control from the Toolbox to the ninth cell of the left column in the **div** element, change the **(ID)** property to **CustomerEmailAddressLabel**, and then set the **Text** property to **Email Address:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the ninth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Label** control.

    d. In the InsertCustomer.aspx window, click the **Label** control.

    e. In the Properties window, change the **(ID)** property to **CustomerEmailAddressLabel**.

    f. In the Properties window, in the **Text** property, type **Email Address:**

19. Add the **TextBox** control from the Toolbox to the ninth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerEmailAddressTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the ninth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerEmailAddressTextBox**.

20. Add the **Label** control from the Toolbox to the tenth cell of the left column in the **div** element, change the **(ID)** property to **CustomerWebAddressLabel**, and then set the **Text** property to **Web Address:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the tenth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Label** control.

    d. In the InsertCustomer.aspx window, click the **Label** control.

    e. In the Properties window, change the **(ID)** property to **CustomerWebAddressLabel**.

    f. In the Properties window, in the **Text** property, type **Web Address:**.

21. Add the **TextBox** control from the Toolbox to the tenth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerWebAddressTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the tenth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

    d. In the InsertCustomer.aspx window, click the **TextBox** control.

    e. In the Properties window, change the **(ID)** property to **CustomerWebAddressTextBox**.

22. Add the **Label** control from the Toolbox to the eleventh cell of the left column in the **div** element, change the **(ID)** property to **CustomerCreditLimitLabel**, and then set the **Text** property to **Credit Limit:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the eleventh cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, change the **(ID)** property to **CustomerCreditLimitLabel**.

f. In the Properties window, in the **Text** property, type **Credit Limit:**.

23. Add the **TextBox** control from the Toolbox to the eleventh cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCreditLimitTextBox**.

a. In the InsertCustomer.aspx window, in the **div** element, click the eleventh cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerCreditLimitTextBox**.

24. Add the **Label** control from the Toolbox to the twelfth cell of the left column in the **div** element, change the **(ID)** property to **CustomerNewsSubscriberLabel**, and then set the **Text** property to **News Subscriber:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the twelfth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, change the **(ID)** property to **CustomerNewsSubscriberLabel**.

f. In the Properties window, in the **Text** property, type **News Subscriber:**.

25. Add the **CheckBox** control from the Toolbox to the twelfth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerNewsSubscriberCheckBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the twelfth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **CheckBox** control.

    d. In the InsertCustomer.aspx window, click the **CheckBox** control.

    e. In the Properties window, change the **(ID)** property to **CustomerNewsSubscriberCheckBox**.

26. Save the changes, and view the InsertCustomer Web Form in a Web browser.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.

    b. In Solution Explorer, under D:\Labfiles\Starter\M3\VB \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

> **Note:** Notice that the controls have been placed in two equal-sized columns in the center of the browser, because of the CSS styling.

    c. In the http://localhost:1111/CustomerManagement/InsertCustomer.aspx - Windows Internet Explorer window, click the **Close** button.

27. Add the **Button** control from the Toolbox to the footer of the table in the **div** element, change the **(ID)** property to **CustomerInsertButton**, and then set the **Text** property to **Insert**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the table footer that has **customerTableFooter** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Button** control.

    d. In the InsertCustomer.aspx window, click the **Button** control.

e. In the Properties window, change the **(ID)** property to **CustomerInsertButton**.

f. In the Properties window, in the **Text** property, type **Insert**.

28. Add the **Button** control from the Toolbox to the footer of the table in the **div** element, change the **(ID)** property to **CustomerCancelButton**, and then set the **Text** property to **Cancel**.

a. In the InsertCustomer.aspx window, in the **div** element, place the cursor to the right of the **Insert** control, and then press the SPACEBAR.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Button** control.

d. In the InsertCustomer.aspx window, click the **Button** control.

e. In the Properties window, change the **(ID)** property to **CustomerCancelButton**.

f. In the Properties window, in the **Text** property, type **Cancel**.

29. Save the changes, and view the InsertCustomer Web Form in a Web browser.

a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.

b. In Solution Explorer, under D:\Labfiles\Starter\M3\VB \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

Note: Notice that the two Button controls have been placed in the table footer, but are centered like the other controls, because of the CSS styling.

c. In the http://localhost:1111/CustomerManagement/InsertCustomer.aspx - Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Set the non-trivial control properties

1. Set the **MaxLength** property of the **CustomerFirstNameTextBox**, **CustomerAddressTextBox**, and **CustomerEmailAddressTextBox** controls to **50**.

   a. In the InsertCustomer.aspx window, click the **CustomerFirstNameTextBox** control, press the CTRL key, and then click the **CustomerAddressTextBox**, and **CustomerEmailAddressTextBox** controls.

   b. In the Properties window, in the **MaxLength** property, type **50**.

2. Set the **MaxLength** property of the **CustomerLastNameTextBox**, **CustomerCityTextBox**, **CustomerStateTextBox**, and **CustomerPhoneTextBox** controls to **30**.

   a. In the InsertCustomer.aspx window, click the **CustomerLastNameTextBox** control, press CTRL, and then click the **CustomerCityTextBox**, **CustomerStateTextBox** and **CustomerPhoneTextBox** controls.

   b. In the Properties window, in the **MaxLength** property, type **30**.

3. Set the **MaxLength** property of the **CustomerZipCodeTextBox** and **CustomerCreditLimitTextBox** controls to **10**.

   a. In the InsertCustomer.aspx window, click the **CustomerZipCodeTextBox** control, press CTRL, and then click the **CustomerCreditLimitTextBox** control.

   b. In the Properties window, in the **MaxLength** property, type **10**.

4. Open the InsertCustomer Web Form in the Source view.

   - In the InsertCustomer.aspx window, click **Source**.

5. Set the **MaxLength** property of the **CustomerWebAddressTextBox** control to **80**.

   a. In the InsertCustomer.aspx window, place the cursor in the markup for the **CustomerWebAddressTextBox** control, after the **runat="server"** attribute and value, and then press SPACEBAR.

   b. In the drop-down list, double-click **MaxLength**, type **=**, and then type **80**.

6. Save the changes, and view the InsertCustomer Web Form in a Web browser.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.

b. In Solution Explorer, under D:\Labfiles\Starter\M3\VB \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

c. In the **Zip Code** text box, type **0123456789**.

> **Note:** Notice that you cannot enter more than 10 characters in the **Zip Code** text box.

d. In the http://localhost:1111/CustomerManagement/InsertCustomer.aspx - Windows Internet Explorer window, click the **Close** button.

▶ Task 5: Apply a predefined style to the Web Form

1. Apply the predefined template style to the **body** element of the InsertCustomer Web form.

   a. In the InsertCustomer.aspx window, place the cursor in the opening tag of the **body** element, next to the text **body**, and press SPACEBAR.

   b. In the drop-down list, double-click **class**, type **=**, and then double-click **template**.

2. Save the changes, and view the InsertCustomer Web Form in a Web browser.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.

   b. In Solution Explorer, under D:\Labfiles\Starter\M3\VB \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

   c. In the http://localhost:1111/CustomerManagement/InsertCustomer.aspx - Windows Internet Explorer window, click the **Close** button.

   d. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 6: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 3

Lab Answer Key: Creating a Microsoft® ASP.NET Web Form (Visual C#)

Contents:

Lab: Creating a Microsoft ASP.NET Web Form

(C#)

Exercise 1: Creating a Web Form

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M3\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click Ope**n Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M3\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Add a Web Form to the Web site

1. Add the InsertCustomer.aspx Web Form to the CustomerManagement Web site, by using the **Add New Item** dialog box.
2. In Solution Explorer, right-click **D:\Labfiles\Starter\M3\CS \CustomerManagement**, and then click **Add New Item**.
3. In the **Add New Item – D:\Labfiles\Starter\M3\CS \CustomerManagement** dialog box, in the Templates pane in the middle of the dialog box, click **Web Form**, in the **Name** box, type **InsertCustomer.aspx**, and then click **Add**.

> **Note:** The InsertCustomer Web Form displays in Source view.

Exercise 2: Adding and Configuring Server Controls in a Web Form

▶ Task 1: Add an existing style sheet to the Web Form

1. Reference the **Styles/Site.css** file in the InsertCustomer Web Form, relative to the root folder, by using the Manage Styles window.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **View** menu, click **Manage Styles**.
   b. In the Manage Styles window, click the **Attach Style Sheet** button.
   c. In the **Select Style Sheet** dialog box, in the **Project Folders** list, click **Styles**, in the **Contents of folder** list, click **Site.css**, and then click **OK**.

> **Note:** Notice that a self-closing **link** element is added to the **head** element, just below the **title** element.

2. In the InsertCustomer.aspx window, modify the **href** attribute of the self-closing link element to include a tilde (~) and a forward slash (/) character at the beginning of the path.

```
<link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />
```

3. Press CTRL+S to save InsertCustomer.aspx.
4. View the styles added in the style sheet by using the Manage Styles window.
   a. In the Manage Styles window, under cascading style sheet (CSS) styles, view all the styles added in the **Site.css** list.
   b. In the Manage Styles window, click the **Close** button.

▶ Task 2: Create a table-style layout in the Web Form

1. In the InsertCustomer.aspx window, place the cursor between the opening and closing div tags.
2. Add a new **div** element from the Toolbox to the existing div element.

```
<div>
    <div>
    </div>
</div>
```

   a. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

   b. In Toolbox, expand **HTML**, and then double-click the **div** element.

```
<div>
    <div>
    </div>
</div>
```

3. Add two new **div** elements from the Toolbox to the newly added **div** element.

```
<div>
    <div>
        <div>
        </div>
        <div>
        </div>
    </div>
</div>
```

   a. In the InsertCustomer.aspx window, place the cursor between the opening and closing div tags of the newly added div element.

   b. In the Toolbox, expand **HTML**, and then double-click the **div** element twice.

```
<div>
    <div>
        <div>
        </div>
        <div>
        </div>
    </div>
</div>
```

4. Save the changes to the InsertCustomer Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, click CTRL+S.
5. Use the Apply Styles window to apply the **customerTable** style to the outermost **div** element. Apply **div.customerTable** from the **Contextual Selectors** section of the Apply Styles window.
   a. In the InsertCustomer.aspx window, click the opening tag of the outermost **div** element.
   b. In the CustomerManagement – Microsoft Visual Studio window, on the **View** menu, click **Apply Styles**.
   c. In the Apply Styles window, in the **Contextual Selectors** section, under **Site.css**, click **div.customerTable**.
6. Click the opening tag of the **div** element, which is a child element of the **div** element with a **class** attribute value of **customerTable**.
   - In the InsertCustomer.aspx window, click the opening tag of the **div** element, which is a child element of the **div** element with a **class** attribute value of **customerTable**.
7. Create new style in the Apply Styles window, by using the **New Style** button.
   - In the Apply Styles window, click the **New Style** button.
8. Name the new style **div.customerTableRow** by using the **Selector** box of the **New Style** dialog box.
   - In the **New Style** dialog box, in the **Selector** box, type **div.customerTableRow**
9. Apply the style to the current document selection by using the **Apply new style to document selection** check box.
   - In the **New Style** dialog box, select the **Apply new style to document selection** check box.
10. Define the new style in the existing **Styles/Site.css** CSS file.
    - In the **New Style** dialog box, in the **Define in** list, click **Existing style sheet**, and then in the URL list, click **Styles/Site.css**.

11. Set the layout of the new style to **table-row** by using the display list of the **Layout** category, and then close the **New Style** dialog box.

    - In the **New Style** dialog box, in the **Category** list, click **Layout**, in the **display** list, click **table-row**, and then click **OK**.

12. Use the Apply Styles window to apply the **customerTableLeftCol** style to the first child div element of the div element with a class attribute value of **customerTableRow**.

    a. In the InsertCustomer.aspx window, click the opening tag of the div element, which is the first child element of the div element with a class attribute value of **customerTableRow**.

    b. In the CustomerManagement – Microsoft Visual Studio window, on the **View** menu, click **Apply Styles**.

    c. In the Apply Styles window, in the **Contextual Selectors** section, under **Site.css**, click **div.customerTableLeftCol**.

13. Use the Apply Styles window to apply the **customerTableRightCol** style to the second child div element of the div element with a class attribute value of **customerTableRow**.

    a. In the InsertCustomer.aspx window, click the opening tag of the div element, which is the second child element of the div element with a class attribute value of **customerTableRow**.

    b. In the CustomerManagement – Microsoft Visual Studio window, on the **View** menu, click **Apply Styles**.

    c. In the Apply Styles window, in the **Contextual Selectors** section, under **Site.css**, click **div.customerTableRightCol**.

14. Add 11 more table rows, by copying the existing **div** element with a class attribute value of **customerTableRow**.

    a. In the InsertCustomer.aspx window, select the div element with a class attribute value of **customerTableRow**, right-click the selection, and then click **Copy**.

```
<div class="customerTableRow">
    <div class="customerTableLeftCol">
    </div>
    <div class="customerTableRightCol">
    </div>
</div>
```

b. In the InsertCustomer.aspx window, place the cursor after the closing **div** tag of the div element with a class attribute value of **customerTableRow** and then press ENTER.

c. Press CTRL+V 11 times in succession.

15. Append a new div element from the Toolbox to the div element with a **class** attribute value of **customerTable**, placing the new **div** element immediately before the closing div tag of the **customerTable** div element.

```
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
        </div>
        <div class="customerTableRightCol">
        </div>
    </div>
    ...
    <div>
    </div>
</div>
```

a. In the InsertCustomer.aspx window, place the cursor after the last closing div tag of the div element with a class attribute value of **customerTableRow** and then press ENTER.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, expand **HTML**, and then double-click the **div** element.

```
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
        </div>
        <div class="customerTableRightCol">
        </div>
    </div>
    ...
    <div>
    </div>
</div>
```

16. View the InsertCustomer Web Form in Design view.
    - In the InsertCustomer.aspx window, click **Design**.

> **Note:** Notice that two equal-sized columns are added to the **div** element.

17. View the InsertCustomer Web Form in Source view.
    - In the InsertCustomer.aspx window, click **Source**.
18. Use the Apply Styles window to apply the **customerTableFooter** style to the last child div element of the div element with class attribute value of **customerTable**.
    a. In the InsertCustomer.aspx window, click the opening tag of the div element, which is the last child element of the div element with a class attribute value of **customerTable**.
    b. In the Apply Styles window, in the **Contextual Selectors** section, under **Site.css**, click **div.customerTableFooter**.
    c. In the Apply Styles window, click the **Close** button.
19. Save the changes to the InsertCustomer Web Form and the Site.css CSS stylesheet file.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**.
20. View the InsertCustomer Web Form in a Web browser.
    a. In Solution Explorer, under D:\Labfiles\Starter\M3\CS \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

> **Note:** Although you have styled the Web form by using the CSS file, notice that nothing displays. This is because the div elements are empty.

    b. In the http://localhost:1110/CustomerManagement /InsertCustomer.aspx - Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Add the server controls to the Web Form and configure their basic properties

1. View the InsertCustomer Web Form in Design view.
   - In the InsertCustomer.aspx window, click **Design**.
2. Add the **Label** control from the Toolbox to the first cell of the left column in the **div** element, change the **(ID)** property to **CustomerFirstNameLabel**, and then set the **Text** property to **First Name:**.
   a. In the InsertCustomer.aspx window, in the first div element, click the first cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.
   b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
   c. In Toolbox, expand **Standard**, and then double-click the **Label** control.
   d. In the InsertCustomer.aspx window, click the **Label** control.
   e. In the Properties window, change the **(ID)** property to **CustomerFirstNameLabel**.
   f. In the Properties window, in the **Text** property, type **First Name:**.
3. Add the **TextBox** control from the Toolbox to the first cell of the right column in the **div** element, and change the **(ID)** property to **CustomerFirstNameTextBox**.
   a. In the InsertCustomer.aspx window, in the div element, click the first cell of the right column that has **customerTableRightCol** as the value for the **Class** property.
   b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
   c. In Toolbox, under **Standard**, double-click the **TextBox** control.
   d. In the InsertCustomer.aspx window, click the **TextBox** control.
   e. In the Properties window, change the **(ID)** property to **CustomerFirstNameTextBox**.

4. Add the Label control from the Toolbox to the second cell of the left column in the **div** element, change the **(ID)** property to **CustomerLastNameLabel**, and then set the **Text** property to **Last Name:**.

    a. In the InsertCustomer.aspx window, in the second **div** element, click the first cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, expand **Standard**, and then double-click the **Label** control.

    d. In the InsertCustomer.aspx window, click the **Label** control.

    e. In the Properties window, change the **(ID)** property to **CustomerLastNameLabel**.

    f. In the Properties window, in the **Text** property, type **Last Name:**.

5. Add the **TextBox** control from the Toolbox to the second cell of the right column in the **div** element, and change the **(ID)** property to **CustomerLastNameTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the second cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

    d. In the InsertCustomer.aspx window, click the **TextBox** control.

    e. In the Properties window, change the **(ID)** property to **CustomerLastNameTextBox**.

6. Add the **Label** control from the Toolbox to the third cell of the left column in the **div** element, change the **(ID)** property to **CustomerAddressLabel**, and then set the **Text** property to **Address:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the third cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, change the **(ID)** property to **CustomerAddressLabel**.

f. In the Properties window, in the **Text** property, type **Address:**.

7. Add the **TextBox** control from the Toolbox to the third cell of the right column in the **div** element, and change the **(ID)** property to **CustomerAddressTextBox**.

a. In the InsertCustomer.aspx window, in the **div** element, click the third cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerAddressTextBox**.

8. Add the **Label** control from the Toolbox to the fourth cell of the left column in the **div** element, change the **(ID)** property to **CustomerZipCodeLabel**, and then set the **Text** property to **Zip Code:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the fourth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, in the **(ID)** property, type **CustomerZipCodeLabel**.

f. In the Properties window, in the **Text** property, type **Zip Code:**.

9. Add the **TextBox** control from the Toolbox to the fourth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerZipCodeTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the fourth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

    d. In the InsertCustomer.aspx window, click the **TextBox** control.

    e. In the Properties window, change the **(ID)** property to **CustomerZipCodeTextBox**.

10. Add the Label control from the Toolbox to the fifth cell of the left column in the **div** element, change the **(ID)** property to **CustomerCityLabel**, and then set the **Text** property to **City:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the fifth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Label** control.

    d. In the InsertCustomer.aspx window, click the **Label** control.

    e. In the Properties window, change the **(ID)** property to **CustomerCityLabel**.

    f. In the Properties window, in the **Text** property, type **City:**.

11. Add the **TextBox** control from the Toolbox to the fifth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCityTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the fifth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerCityTextBox**.

12. Add the **Label** control from the Toolbox to the sixth cell of the left column in the **div** element, change the **(ID)** property to **CustomerStateLabel**, and then set the **Text** property to **State:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the sixth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, change the **(ID)** property to **CustomerStateLabel**.

f. In the Properties window, in the **Text** property, type **State:**.

13. Add the **TextBox** control from the Toolbox to the sixth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerStateTextBox**.

a. In the InsertCustomer.aspx window, in the **div** element, click the sixth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerStateTextBox**.

14. Add the **Label** control from the Toolbox to the seventh cell of the left column in the **div** element, change the **(ID)** property to **CustomerCountryLabel**, and then set the **Text** property to **Country:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the seventh cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, for the **(ID)** property, type **CustomerCountryLabel**.

f. In the Properties window, for the **Text** property, type **Country:**.

15. Add the **DropDownList** control from the Toolbox to the seventh cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCountryDropDownList**.

a. In the InsertCustomer.aspx window, in the **div** element, click the seventh cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **DropDownList** control.

d. In the Properties window, change the **(ID)** property to **CustomerCountryDropDownList**.

16. Add the **Label** control from the Toolbox to the eighth cell of the left column in the **div** element, change the **(ID)** property to **CustomerPhoneLabel**, and then set the **Text** property to **Phone:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the eighth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, for the **(ID)** property, type **CustomerPhoneLabel**.

f. In the Properties window, for the **Text** property, type **Phone:**.

17. Add the **TextBox** control from the Toolbox to the eighth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerPhoneTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the eighth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

    d. In the Properties window, change the **(ID)** property to **CustomerPhoneTextBox**.

18. Add the **Label** control from the Toolbox to the ninth cell of the left column in the **div** element, change the **(ID)** property to **CustomerEmailAddressLabel**, and then set the **Text** property to **Email Address:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the ninth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Label** control.

    d. In the InsertCustomer.aspx window, click the **Label** control.

    e. In the Properties window, change the **(ID)** property to **CustomerEmailAddressLabel**.

    f. In the Properties window, in the **Text** property, type **Email Address:**.

19. Add the **TextBox** control from the Toolbox to the ninth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerEmailAddressTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the ninth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerEmailAddressTextBox**.

20. Add the **Label** control from the Toolbox to the tenth cell of the left column in the **div** element, change the **(ID)** property to **CustomerWebAddressLabel**, and then set the **Text** property to **Web Address:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the tenth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Label** control.

    d. In the InsertCustomer.aspx window, click the **Label** control.

    e. In the Properties window, change the **(ID)** property to **CustomerWebAddressLabel**.

    f. In the Properties window, in the **Text** property, type **Web Address:**.

21. Add the **TextBox** control from the Toolbox to the tenth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerWebAddressTextBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the tenth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **TextBox** control.

    d. In the InsertCustomer.aspx window, click the **TextBox** control.

    e. In the Properties window, change the **(ID)** property to **CustomerWebAddressTextBox**.

22. Add the **Label** control from the Toolbox to the eleventh cell of the left column in the **div** element, change the **(ID)** property to **CustomerCreditLimitLabel**, and then set the **Text** property to **Credit Limit:**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the eleventh cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, change the **(ID)** property to **CustomerCreditLimitLabel**.

f. In the Properties window, in the **Text** property, type **Credit Limit:**.

23. Add the **TextBox** control from the Toolbox to the eleventh cell of the right column in the **div** element, and change the **(ID)** property to **CustomerCreditLimitTextBox**.

a. In the InsertCustomer.aspx window, in the **div** element, click the eleventh cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **TextBox** control.

d. In the InsertCustomer.aspx window, click the **TextBox** control.

e. In the Properties window, change the **(ID)** property to **CustomerCreditLimitTextBox**.

24. Add the **Label** control from the Toolbox to the twelfth cell of the left column in the **div** element, change the **(ID)** property to **CustomerNewsSubscriberLabel**, and then set the **Text** property to **News Subscriber:**.

a. In the InsertCustomer.aspx window, in the **div** element, click the twelfth cell of the left column that has **customerTableLeftCol** as the value for the **Class** property.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Label** control.

d. In the InsertCustomer.aspx window, click the **Label** control.

e. In the Properties window, change the **(ID)** property to **CustomerNewsSubscriberLabel**.

f. In the Properties window, in the **Text** property, type **News Subscriber:**.

25. Add the **CheckBox** control from the Toolbox to the twelfth cell of the right column in the **div** element, and change the **(ID)** property to **CustomerNewsSubscriberCheckBox**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the twelfth cell of the right column that has **customerTableRightCol** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **CheckBox** control.

    d. In the InsertCustomer.aspx window, click the **CheckBox** control.

    e. In the Properties window, change the **(ID)** property to **CustomerNewsSubscriberCheckBox**.

26. Save the changes, and view the InsertCustomer Web Form in a Web browser.

    a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.

    b. In Solution Explorer, under D:\Labfiles\Starter\M3\CS \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

> **Note:** Notice that the controls have been placed in two equal-sized columns in the center of the browser, because of the CSS styling.

    c. In the http://localhost:1110/CustomerManagement /InsertCustomer.aspx - Windows Internet Explorer window, click the **Close** button.

27. Add the **Button** control from the Toolbox to the footer of the table in the **div** element, change the **(ID)** property to **CustomerInsertButton**, and then set the **Text** property to **Insert**.

    a. In the InsertCustomer.aspx window, in the **div** element, click the table footer that has **customerTableFooter** as the value for the **Class** property.

    b. In the CustomerManagement − Microsoft Visual Studio window, point to **Toolbox**.

    c. In Toolbox, under **Standard**, double-click the **Button** control.

d. In the InsertCustomer.aspx window, click the **Button** control.

e. In the Properties window, change the **(ID)** property to **CustomerInsertButton**.

f. In the Properties window, in the **Text** property, type **Insert**.

28. Add the **Button** control from the Toolbox to the footer of the table in the **div** element, change the **(ID)** property to **CustomerCancelButton**, and then set the **Text** property to **Cancel**.

a. In the InsertCustomer.aspx window, in the **div** element, place the cursor to the right of the **Insert** control, and then press the SPACEBAR.

b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

c. In Toolbox, under **Standard**, double-click the **Button** control.

d. In the InsertCustomer.aspx window, click the **Button** control.

e. In the Properties window, change the **(ID)** property to **CustomerCancelButton**.

f. In the Properties window, in the **Text** property, type **Cancel**.

29. Save the changes, and view the InsertCustomer Web Form in a Web browser.

a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.

b. In Solution Explorer, under D:\Labfiles\Starter\M3\CS \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

Note: Notice that the two Button controls have been placed in the table footer, but are centered like the other controls, because of the CSS styling.

c. In the http://localhost:1110/CustomerManagement /InsertCustomer.aspx - Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Set the non-trivial control properties

1. Set the **MaxLength** property of the **CustomerFirstNameTextBox**, **CustomerAddressTextBox**, and **CustomerEmailAddressTextBox** controls to **50**.

   a. In the InsertCustomer.aspx window, click the **CustomerFirstNameTextBox** control, press the CTRL key, and then click the **CustomerAddressTextBox**, and **CustomerEmailAddressTextBox** controls.

   b. In the Properties window, in the **MaxLength** property, type **50**.

2. Set the **MaxLength** property of the **CustomerLastNameTextBox**, **CustomerCityTextBox**, **CustomerStateTextBox**, and **CustomerPhoneTextBox** controls to **30**.

   a. In the InsertCustomer.aspx window, click the **CustomerLastNameTextBox** control, press the CTRL key, and then click the **CustomerCityTextBox**, **CustomerStateTextBox** and **CustomerPhoneTextBox** controls.

   b. In the Properties window, in the **MaxLength** property, type **30**.

3. Set the **MaxLength** property of the **CustomerZipCodeTextBox** and **CustomerCreditLimitTextBox** controls to **10**.

   a. In the InsertCustomer.aspx window, click the **CustomerZipCodeTextBox** control, press the CTRL key, and then click the **CustomerCreditLimitTextBox** control.

   b. In the Properties window, in the **MaxLength** property, type **10**.

4. Open the InsertCustomer Web Form in the Source view.

   - In the InsertCustomer.aspx window, click **Source**.

5. Set the **MaxLength** property of the **CustomerWebAddressTextBox** control to **80**.

   a. In the InsertCustomer.aspx window, place the cursor in the markup for the **CustomerWebAddressTextBox** control, after the **runat="server"** attribute and value, and then press SPACEBAR.

   b. In the drop-down list, double-click **MaxLength**, type **=**, and then type **80**.

6. Save the changes, and view the InsertCustomer Web Form in a Web browser.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.

b. In Solution Explorer, under D:\Labfiles\Starter\M3\CS \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

c. In the **Zip Code** text box, type **0123456789**.

> **Note:** Notice that you cannot enter more than 10 characters in the Zip Code text box.

d. In the http://localhost:1110/CustomerManagement /InsertCustomer.aspx - Windows Internet Explorer window, click the **Close** button.

▶ Task 5: Apply a predefined style to the Web Form

1. Apply the predefined template style to the **body** element of the InsertCustomer Web form.

   a. In the InsertCustomer.aspx window, place the cursor in the opening tag of the **body** element, next to the text **body**, and press SPACEBAR.

   b. In the drop-down list, double-click **class**, type **=**, and then double-click **template**.

2. Save the changes, and view the InsertCustomer Web Form in a Web browser.

   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.

   b. In Solution Explorer, under D:\Labfiles\Starter\M3\CS \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

   c. In the http://localhost:1110/CustomerManagement /InsertCustomer.aspx - Windows Internet Explorer window, click the **Close** button.

   d. In the CustomerManagement − Microsoft Visual Studio window, click the **Close** button.

▶ Task 6: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert** Virtual **Machine** dialog box, click **Revert**.

Module 4

Lab Answer Key: Adding Functionality to a Microsoft® ASP.NET Web Form (Visual Basic)

Contents:

Lab: Adding Functionality to a Microsoft® ASP.NET Web Form

Exercise 1: Implementing Code in a Web Application

▶ Task 1: Open an existing Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M4\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M4\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Open the code-behind file for an existing Web Form

- Open the code-behind file of the InsertCustomer Web Form, by using the **View Code** context menu command.
  - In Solution Explorer, right-click the **InsertCustomer** Web Form, and then click **View Code**.

Exercise 2: Creating Event Procedures

▶ Task 1: Create an event procedure for the Click event of the Insert button

- Open the **InsertCustomer** Web Form in Design view, and create an event procedure for the **Click** event of the **Insert** button, by double-clicking the **Button** control.
  - In the InsertCustomer.aspx.vb code window, right-click anywhere, and then click **View Designer**.
  - In the InsertCustomer.aspx window, double-click the **Insert** button.

Note: Notice that the initial event procedure for the **Click** event of the **CustomerInsertButton** control is added in the code window.

▶ Task 2: Create an event procedure for the Click event of the Cancel button

- Open the **InsertCustomer** Web Form in the Design view, and create an event procedure for the **Click** event of the **Cancel** button, by double-clicking the box next to the **Click** event in the Properties window, with the **Button** control selected in the Designer.
  - In the InsertCustomer.aspx.vb code window, right-click anywhere, and then click **View Designer**.
  - In the InsertCustomer.aspx window, click the **Cancel** button.
  - In the Properties window, click the **Events** icon, and then double-click the box next to the **Click** event.

Note: Notice that the initial event procedure for the **Click** event of the **CustomerCancelButton** is added to the class in the code window.

▶ Task 3: Create an event procedure for the Page_Load event

- Create an event procedure for the **Page_Load** event.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load

End Sub
```

  - In the InsertCustomer.aspx.vb code window, in the **Object** list, click **(Page Events)**, and then in the **Declarations** list, click **Load**.

▶ Task 4: Create an event procedure for the Page_LoadComplete event

- Create an event procedure for the **Page_LoadComplete** event.

```
Protected Sub Page_LoadComplete(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.LoadComplete

End Sub
```

  - In the In the InsertCustomer.aspx.vb code window, in the **Object** list, click **(Page Events)**, and then in the **Declarations** list, click **LoadComplete**.

▶ Task 5: Create an event procedure for the Page_Unload event

- Create an event procedure for the **Page_Unload** event.

```
Protected Sub Page_Unload(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Unload

End Sub
```

  - In the In the InsertCustomer.aspx.vb code window, in the **Object** list, click **(Page Events)**, and then in the **Declarations** list, click **Unload**.

Exercise 3: Creating an Entity Component

▶ Task 1: Create an entity component

1. Create the **CustomerManagementEntities** class library project by using the **Class Library project** item in the **Add New Project** dialog box. Place the new project in the D:\Labfiles\Starter\M4\VB folder.
2. In Solution Explorer, right-click **Solution 'CustomerManagement' (1 project)**, point to **Add**, and then click **New Project**.
   a. In the **Add New Project** dialog box, in the left pane, click **Visual Basic**.
   b. In the middle pane, click **Class Library**.
   c. In the **Name** box, type **CustomerManagementEntities**.
   d. In the **Location** box, ensure that **D:\Labfiles\Starter\M4\VB** is entered, and then click **OK**.

> **Note:** Notice that the **CustomerManagementEntities** class library project is added in Solution Explorer.

3. Rename the default class file **Class1.vb**, as **Customer.vb**.
   a. In Solution Explorer, right-click **Class1.vb**, click **Rename**, modify the name as **Customer.vb**, and then press ENTER.
   b. In the Microsoft Visual Studio message box, click **Yes**.

▶ Task 2: Add the class member fields

1. In the **Customer** class, add a region named **Class member fields**, just below the class declaration.
   - In the Customer.vb code window, type the following code in the **Customer** class, just below the class declaration.

```
#Region "Class member fields"
#End Region
```

2. In the **Customer** class, within the **Class member fields** region, add a private member field for the first name of the customer named **customerFirstName**, of type **string**, and initialize to **Nothing**.

```
Private customerFirstName As String = Nothing
```

3. In the **Customer** class, within the **Class member fields** region, add a private member field named **customerLastName**, of type **String**, for the last name of the customer, and initialize to **Nothing**.

```
Private customerLastName As String = Nothing
```

4. In the **Customer** class, within the **Class member fields** region, add a private member field named **customerAddress**, of type **String**, for the address of the customer, and initialize to **Nothing**.

```
Private customerAddress As String = Nothing
```

5. Append the remaining backing fields by using a code snippet named **Customer class backing fields**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the declaration of the **customerAddress** backing field, and insert the snippet by clicking **Insert Snippet**.

   a. In the CustomerManagement − Microsoft Visual Studio window, right-click the line following the declaration of the **customerAddress** backing field, and then click **Insert Snippet**.

   b. In the context menu, double-click **My Code Snippets**, and then double-click **Customer class backing fields**.

```
Private customerZipCode As String = Nothing
Private customerCity As String = Nothing
Private customerState As String = Nothing
Private customerCountryID As Guid? = Nothing
Private customerPhone As String = Nothing
Private customerEmailAddress As String = Nothing
Private customerWebAddress As String = Nothing
Private customerCreditLimit As Integer = 0
Private customerNewsSubscriber As Boolean = False
Private customerCreatedDate As DateTime? = Nothing
Private customerCreatedBy As String = Nothing
Private customerModifiedDate As DateTime? = Nothing
Private customerModifiedBy As String = Nothing
```

▶ Task 3: Add the properties

1. In the **Customer** class, add a region named **Properties**, below the **Class member fields** region.

   - In the Customer.vb code window, type the following code in the **Customer** class, just below the **Class member fields** region.

   ```
   #Region "Properties"
   #End Region
   ```

2. In the **Customer** class, within the **Properties** region, append an auto-implemented public property named **ID**, of nullable type **Guid**.

   ```
   ''' <summary>
   ''' The unique customer ID
   ''' </summary>
   ''' <value></value>
   ''' <returns></returns>
   ''' <remarks></remarks>
   Public Property ID() As Guid?
   ```

   - In the Customer.vb code window, within the **Properties** region, append the following code to the **Customer** class.

   ```
   ''' <summary>
   ''' The unique customer ID
   ''' </summary>
   ''' <value></value>
   ''' <returns></returns>
   ''' <remarks></remarks>
   Public Property ID() As Guid?
   ```

3. In the **Customer** class, within the **Properties** region, add a public property named **FirstName** of type **String**, that sets and gets the private member backing field **customerFirstName**, and ensure that the length is no longer than 50 characters by using the following lines of code.

```
''' <summary>
''' The customer first name
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property FirstName As String
    Get
        Return Me.customerFirstName
    End Get

    Set(ByVal value As String)
        ' Null value?
        If Value Is Nothing Then
            Me.customerFirstName = String.Empty
        Else
            ' Only get the first 50 characters
            If (Value.Length > 50) Then
                Me.customerFirstName = value.Substring(0, 50)
            Else
                Me.customerFirstName = value
            End If
        End If
    End Set
End Property
```

- In the Customer.vb code window, within the **Properties** region, append the following code to the **Customer** class.

```
''' <summary>
''' The customer first name
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property FirstName As String
    Get
        Return Me.customerFirstName
    End Get

    Set(ByVal value As String)
        ' Null value?
        If Value Is Nothing Then
            Me.customerFirstName = String.Empty
        Else
            ' Only get the first 50 characters
            If (Value.Length > 50) Then
                Me.customerFirstName = value.Substring(0, 50)
            Else
                Me.customerFirstName = value
            End If
        End If
    End Set
End Property
```

4. In the **Customer** class, within the **Properties** region, add a public property named **LastName** of type **String**, that sets and gets the private member backing field **customerLastName**, and ensure that the length is no longer than 30 characters by using the following lines of code.

```
''' <summary>
''' The customer last name
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property LastName As String
    Get
        Return Me.customerLastName
    End Get

    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerLastName = String.Empty
        Else
            ' Only get the first 30 characters
            If (value.Length > 30) Then
                Me.customerLastName = value.Substring(0, 30)
            Else
                Me.customerLastName = value
            End If
        End If
    End Set
End Property
```

- In the Customer.vb code window, within the **Properties** region, append the following code to the **Customer** class.

```
''' <summary>
''' The customer last name
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property LastName As String
    Get
        Return Me.customerLastName
    End Get

    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerLastName = String.Empty
        Else
            ' Only get the first 30 characters
            If (value.Length > 30) Then
                Me.customerLastName = value.Substring(0, 30)
            Else
                Me.customerLastName = value
            End If
        End If
    End Set
End Property
```

5. In the **Customer** class, within the **Properties** region, add a public property named **Address** of type **string**, that sets and gets the private member backing field **customerAddress**, and ensure that the length is no longer than 50 characters by using the following lines of code.

```
''' <summary>
''' The customer address, including street name, house number and
floor
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property Address As String
    Get
        Return Me.customerAddress
    End Get

    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerAddress = String.Empty
        Else
            ' Only get the first 50 characters
            If (value.Length > 50) Then
                Me.customerAddress = value.Substring(0, 50)
            Else
                Me.customerAddress = value
            End If
        End If
    End Set
End Property
```

- In the Customer.vb code window, within the **Properties** region, append the following code to the **Customer** class.

```
''' <summary>
''' The customer address, including street name, house number
and floor
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property Address As String
    Get
        Return Me.customerAddress
    End Get

    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerAddress = String.Empty
        Else
            ' Only get the first 50 characters
            If (value.Length > 50) Then
                Me.customerAddress = value.Substring(0, 50)
            Else
                Me.customerAddress = value
            End If
        End If
    End Set
End Property
```

6. Append the remaining properties within the **Properties** region by using a code snippet named **Customer class properties**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the declaration of the **Address** property, and insert the snippet by clicking **Insert Snippet**.

    a. In the CustomerManagement – Microsoft Visual Studio window, right-click the line following the declaration of the **Address** property, and then click **Insert Snippet**.

b. In the context menu, double-click **My Code Snippets**, and then double-click **Customer class properties**.

```
''' <summary>
''' The customer zip code or postal code
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property ZipCode() As String
    Get
        Return Me.customerZipCode
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerZipCode = ""
        Else
            ' Only get the first 10 characters
            If (value.Length > 10) Then
                Me.customerZipCode = value.Substring(0, 10)
            Else
                Me.customerZipCode = value
            End If
        End If
     End Set
End Property

''' <summary>
''' The name of the city in which the customer lives
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property City() As String
    Get
        Return Me.customerCity
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerCity = ""
        Else
            ' Only get the first 30 characters
            If (value.Length > 30) Then
                Me.customerCity = value.Substring(0, 30)
```

(*Code continued on the following page.*)

```
            Else
                Me.customerCity = value
            End If
        End If
    End Set
End Property

''' <summary>
''' The name of the state or region in which the customer lives
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property State() As String
    Get
        Return Me.customerState
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerState = ""
        Else
            ' Only get the first 30 characters
            If (value.Length > 30) Then
                Me.customerState = value.Substring(0, 30)
            Else
                Me.customerState = value
            End If
        End If
    End Set
End Property

''' <summary>
''' The ID of the country in which the customer lives
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property CountryID() As Guid?
    Get
        Return Me.customerCountryID
    End Get
    Set(ByVal value As Guid?)
        Me.customerCountryID = value
    End Set
End Property
```

(*Code continued on the following page.*)

```
''' <summary>
''' The customer phone number
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property Phone() As String
    Get
        Return Me.customerPhone
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerPhone = ""
        Else
            ' Only get the first 30 characters
            If (value.Length > 30) Then
                Me.customerPhone = value.Substring(0, 30)
            Else
                Me.customerPhone = value
            End If
        End If
    End Set
End Property

''' <summary>
''' The customer e-mail address
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property EmailAddress() As String
    Get
        Return Me.customerEmailAddress
    End Get
    Set(ByVal value As String)
        If (value Is Nothing) Then
            Me.customerEmailAddress = ""
        Else
            ' Only get the first 50 characters
            If (value.Length > 50) Then
                Me.customerEmailAddress = value.Substring(0, 50)
            Else
                Me.customerEmailAddress = value
```

(*Code continued on the following page.*)

```
            End If
        End If
    End Set
End Property

''' <summary>
''' The customer web address
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property WebAddress() As String
    Get
        Return Me.customerWebAddress
    End Get
    Set(ByVal value As String)
        If (value Is Nothing) Then
            Me.customerWebAddress = ""
        Else
           ' Only get the first 80 characters
           If (value.Length > 80) Then
               Me.customerWebAddress = value.Substring(0, 80)
           Else
               Me.customerWebAddress = value
            End If
        End If
    End Set
End Property

''' <summary>
''' The current credit limit of the customer
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property CreditLimit() As Integer
    Get
        Return Me.customerCreditLimit
    End Get
    Set(ByVal value As Integer)
        ' Negative value?
        If value < 0 Then
            Me.customerCreditLimit = 0
        Else
```

(*Code continued on the following page.*)

```
            Me.customerCreditLimit = value
        End If
    End Set
End Property

''' <summary>
''' Does the customer subscriber to news?
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property NewsSubsriber() As Boolean
    Get
        Return Me.customerNewsSubscriber
    End Get
    Set(ByVal value As Boolean)
        Me.customerNewsSubscriber = value
    End Set
End Property

''' <summary>
''' The date the customer was created in the system
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property CreatedDate() As DateTime?
    Get
        Return Me.customerCreatedDate
    End Get
    Private Set(ByVal value As DateTime?)
        ' Date in the past?
        If (value < DateTime.Now) Then
            Me.customerCreatedDate = DateTime.Now
        Else
            Me.customerCreatedDate = value
        End If
    End Set
End Property

''' <summary>
''' The name of the user creating the customer
''' </summary>
''' <value></value>
```

(*Code continued on the following page.*)

```
''' <returns></returns>
''' <remarks></remarks>
Public Property CreatedBy() As String
    Get
        Return Me.customerCreatedBy
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerCreatedBy = ""
        Else
            ' Only get the first 15 characters
            If (value.Length > 15) Then
                Me.customerCreatedBy = value.Substring(0, 15)
            Else
                Me.customerCreatedBy = value
            End If
        End If
    End Set
End Property

''' <summary>
''' The date the customer was last modified in the system
''' </summary>
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property ModifiedDate() As DateTime?
    Get
        Return Me.customerModifiedDate
    End Get
    Set(ByVal value As DateTime?)
        ' Date in the past?
        If value < DateTime.Now Then
            Me.customerModifiedDate = DateTime.Now
        Else
            Me.customerModifiedDate = value
        End If
    End Set
End Property

''' <summary>
''' The name of the user who last modified the customer
''' </summary>
```

(*Code continued on the following page.*)

```
''' <value></value>
''' <returns></returns>
''' <remarks></remarks>
Public Property ModifiedBy() As String
    Get
        Return Me.customerModifiedBy
    End Get
    Set(ByVal value As String)
        ' Null value?
        If value Is Nothing Then
            Me.customerModifiedBy = ""
        Else
            ' Only get the first 15 characters
            If (value.Length > 15) Then
                Me.customerModifiedBy = value.Substring(0, 15)
            Else
                Me.customerModifiedBy = value
            End If
        End If
    End Set
End Property
```

▶ Task 4: Add the constructors

1. In the **Customer** class, add a region named **Constructors**, below the **Properties** region.

```
#Region "Constructors"
#End Region
```

- In the Customer.vb code window, type the following code in the **Customer** class, just below the **Properties** region.

```
#Region "Constructors"
#End Region
```

2. In the **Customer** class, within the **Constructors** region, add the default public parameterless constructor that initializes the **customerID** and **customerCreatedDate** member fields by using the public properties.

```
''' <summary>
''' Default parameterless constructor
''' </summary>
''' <remarks></remarks>
Public Sub New()
    ' Initialize backing fields with default values
    Me.ID = Guid.NewGuid()
    Me.CreatedDate = DateTime.Now
End Sub
```

   - In the Customer.vb code window, within the **Constructors** region, append the following code to the **Customer** class.

```
''' <summary>
''' Default parameterless constructor
''' </summary>
''' <remarks></remarks>
Public Sub New()
    ' Initialize backing fields with default values
    Me.ID = Guid.NewGuid()
    Me.CreatedDate = DateTime.Now
End Sub
```

3. Append the remaining properties within the **Constructors** region by using a code snippet named **Customer class constructors**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the default parameterless constructor, and insert the snippet by clicking **Insert Snippet**.

   a. In the CustomerManagement − Microsoft Visual Studio window, right-click the line following the default parameterless constructor, and then click **Insert Snippet**.

b. In the context menu, double-click **My Code Snippets**, and then double-click **Customer class constructors**.

```
''' <summary>
''' Initializes backing fields with passed and default values
''' </summary>
''' <param name="id"></param>
''' <remarks></remarks>
Public Sub New(ByVal id As Guid?)
    ' Initialize backing fields with passed and default values
    Me.ID = id
    Me.CreatedDate = DateTime.Now
End Sub

''' <summary>
''' Initializes with a value for all backing fields
''' </summary>
''' <param name="id"></param>
''' <param name="firstName"></param>
''' <param name="lastName"></param>
''' <param name="address"></param>
''' <param name="zipCode"></param>
''' <param name="city"></param>
''' <param name="state"></param>
''' <param name="countryID"></param>
''' <param name="phone"></param>
''' <param name="emailAddress"></param>
''' <param name="webAddress"></param>
''' <param name="creditLimit"></param>
''' <param name="newsSubscriber"></param>
''' <param name="createdDate"></param>
''' <param name="createdBy"></param>
''' <param name="modifiedDate"></param>
''' <param name="modifiedBy"></param>
''' <remarks></remarks>
Public Sub New(ByVal id As Guid?, ByVal firstName As String, ByVal
lastName As String,
    ByVal address As String, ByVal zipCode As String, ByVal city
As String, ByVal state As String,
    ByVal countryID As Guid?, ByVal phone As String, ByVal
emailAddress As String,
    ByVal webAddress As String, ByVal creditLimit As Integer,
ByVal newsSubscriber As Boolean,
    ByVal createdDate As DateTime?, ByVal createdBy As String,
ByVal modifiedDate As DateTime?,
    ByVal modifiedBy As String)
```

(*Code continued on the following page.*)

```
    ' Initialize member backing fields with passed values
    Me.ID = id
    Me.FirstName = firstName
    Me.LastName = lastName
    Me.Address = address
    Me.ZipCode = zipCode
    Me.City = city
    Me.State = state
    Me.CountryID = countryID
    Me.Phone = phone
    Me.EmailAddress = emailAddress
    Me.WebAddress = webAddress
    Me.CreditLimit = creditLimit
    Me.NewsSubscriber = newsSubscriber

    If Not createdDate Is Nothing Then
        Me.CreatedDate = createdDate
    Else
        Me.CreatedDate = DateTime.Now
    End If

    Me.CreatedBy = createdBy
    Me.ModifiedDate = modifiedDate
    Me.ModifiedBy = modifiedBy
End Sub
```

4. Save the changes to Customer.vb.
   - In the CustomerManagement – Microsoft Visual Studio window, press the CTRL+S keys.
5. Build the component, and fix any errors.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build CustomerManagementEntities**.

▶ Task 5: Reference CustomerManagementEntities project from CustomerManagement project

- Add a reference to the CustomerManagement project by using the **Add Reference** dialog box. Reference the **CustomerManagementEntities** project from the **CustomerManagement** project.

  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M4\VB \CustomerManagement**, and then click **Add Reference**.

  b. In the **Add Reference** dialog box, on the **Projects** tab, under **Project Name**, ensure that **CustomerManagementEntities** is selected, and then click **OK**.

▶ Task 6: Add a customer member declaration

1. Open the **InsertCustomer.aspx.vb** file.

   - In the CustomerManagement - Microsoft Visual Studio window, click **InsertCustomer.aspx.vb**.

2. In the InsertCustomer.aspx.vb code window, add a private class member declaration of the **Customer** class in the **CustomerManagementEntities** namespace named **currentCustomer**, and initialize to **Nothing**.

```
Private currentCustomer As CustomerManagementEntities.Customer =
Nothing
```

   - In the InsertCustomer.aspx.vb window, after the code,

```
Partial Class InsertCustomer
    Inherits System.Web.UI.Page
```

     type the following code:

```
Private currentCustomer As CustomerManagementEntities.Customer
= Nothing
```

Exercise 4: Handling Page and Control Events

▶ Task 1: Instantiate the Customer object

1. Instantiate the **Customer** object by using the following code.

```
''' <summary>
''' Instantiates Customer object
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Instantiate Customer
    instantiateCustomerObject()
End Sub
```

- In the InsertCustomer.aspx.vb code window, replace the following code,

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
End Sub
```

with:

```
''' <summary>
''' Instantiates Customer object
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Instantiate Customer
    instantiateCustomerObject()
End Sub
```

2. Append the following code to the **InsertCustomer** class.

```
''' <summary>
''' Instantiates and populates the Customer member object
''' </summary>
''' <remarks></remarks>
Private Sub instantiateCustomerObject()
    ' First time loading page?
    If Not Me.IsPostBack Then
        ' Instantiate new Customer object
        currentCustomer = New
CustomerManagementEntities.Customer()
    Else
        ' Instantiate new Customer object with user input
        currentCustomer = New CustomerManagementEntities.Customer(
            Nothing, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
            CustomerAddressTextBox.Text,
CustomerZipCodeTextBox.Text, CustomerCityTextBox.Text,
            CustomerStateTextBox.Text, Nothing,
CustomerPhoneTextBox.Text,
            CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text, -1,
            CustomerNewsSubscriberCheckBox.Checked, DateTime.Now,
            "", Nothing, "")
    End If
End Sub
```

- In the InsertCustomer.aspx.vb code window, append the following code to the **instantiateCustomerObject** class.

```
''' <summary>
''' Instantiates and populates the Customer member object
''' </summary>
''' <remarks></remarks>
Private Sub instantiateCustomerObject()
    ' First time loading page?
    If Not Me.IsPostBack Then
        ' Instantiate new Customer object
        currentCustomer = New
CustomerManagementEntities.Customer()
    Else
        ' Instantiate new Customer object with user input
        currentCustomer = New
CustomerManagementEntities.Customer(
            Nothing, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
            CustomerAddressTextBox.Text,
CustomerZipCodeTextBox.Text, CustomerCityTextBox.Text,
            CustomerStateTextBox.Text, Nothing,
CustomerPhoneTextBox.Text,
            CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text, -1,
            CustomerNewsSubscriberCheckBox.Checked,
DateTime.Now,
            "", Nothing, "")
    End If
End Sub
```

▶ Task 2: Populate the UI controls

- In the **InsertCustomer** class, populate the server controls in the user interface (UI) by using the values from the private **Customer** object **currentCustomer**.

```
''' <summary>
''' Populates UI controls
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_LoadComplete(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.LoadComplete
    ' Populate the UI controls
    populateUI()
End Sub

''' <summary>
''' Populates the UI controls with the values in the
''' current Customer object
''' </summary>
''' <remarks></remarks>
Private Sub populateUI()
    CustomerFirstNameTextBox.Text = currentCustomer.FirstName
    CustomerLastNameTextBox.Text = currentCustomer.LastName
    CustomerAddressTextBox.Text = currentCustomer.Address
    CustomerZipCodeTextBox.Text = currentCustomer.ZipCode
    CustomerCityTextBox.Text = currentCustomer.City
    CustomerStateTextBox.Text = currentCustomer.State

    If currentCustomer.CountryID.HasValue Then
        CustomerCountryDropDownList.SelectedValue =
currentCustomer.CountryID.Value.ToString()
    Else
        CustomerCountryDropDownList.SelectedIndex = -1
    End If

    CustomerPhoneTextBox.Text = currentCustomer.Phone
    CustomerEmailAddressTextBox.Text =
currentCustomer.EmailAddress
    CustomerWebAddressTextBox.Text = currentCustomer.WebAddress
    CustomerCreditLimitTextBox.Text =
currentCustomer.CreditLimit.ToString()
    CustomerNewsSubscriberCheckBox.Checked =
currentCustomer.NewsSubscriber
End Sub
```

- In the InsertCustomer.aspx.vb code window, replace the following code,

```
Protected Sub Page_LoadComplete(ByVal sender As Object, ByVal
e As System.EventArgs) Handles Me.LoadComplete

End Sub
```

with:

```
Protected Sub Page_LoadComplete(ByVal sender As Object, ByVal
e As System.EventArgs) Handles Me.LoadComplete
    ' Populate the UI controls
    populateUI()
End Sub

''' <summary>
''' Populates the UI controls with the values in the
''' current Customer object
''' </summary>
''' <remarks></remarks>
Private Sub populateUI()
    CustomerFirstNameTextBox.Text = currentCustomer.FirstName
    CustomerLastNameTextBox.Text = currentCustomer.LastName
    CustomerAddressTextBox.Text = currentCustomer.Address
    CustomerZipCodeTextBox.Text = currentCustomer.ZipCode
    CustomerCityTextBox.Text = currentCustomer.City
    CustomerStateTextBox.Text = currentCustomer.State

    If currentCustomer.CountryID.HasValue Then
        CustomerCountryDropDownList.SelectedValue =
currentCustomer.CountryID.Value.ToString()
    Else
        CustomerCountryDropDownList.SelectedIndex = -1
    End If

    CustomerPhoneTextBox.Text = currentCustomer.Phone
    CustomerEmailAddressTextBox.Text =
currentCustomer.EmailAddress
    CustomerWebAddressTextBox.Text =
currentCustomer.WebAddress
    CustomerCreditLimitTextBox.Text =
currentCustomer.CreditLimit.ToString()
    CustomerNewsSubscriberCheckBox.Checked =
currentCustomer.NewsSubscriber
End Sub
```

▶ Task 3: Destroy the objects

- In the **InsertCustomer** class, destroy the objects used in the class that are not automatically handled by the garbage collector.

```
''' <summary>
''' Destroys objects
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_Unload(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Unload
    ' Destroy Customer object
    currentCustomer = Nothing
End Sub
```

- In the InsertCustomer.aspx.vb code window, replace the following code,

```
Protected Sub Page_Unload(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Unload

End Sub
```

with:

```
''' <summary>
''' Destroys objects
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_Unload(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Unload
    ' Destroy Customer object
    currentCustomer = Nothing
End Sub
```

▶ Task 4: Handle the user cancellation

- In the **InsertCustomer** class, handle user cancellation by redirecting to the home page.

```
''' <summary>
''' Redirects to home page
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerCancelButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CustomerCancelButton.Click
    ' Redirect to home page
    Response.Redirect("~/Default.aspx")
End Sub
```

- In the InsertCustomer.aspx.vb code window, replace the following code,

```
Protected Sub CustomerCancelButton_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
CustomerCancelButton.Click

End Sub
```

with:

```
''' <summary>
''' Redirects to home page
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerCancelButton_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
CustomerCancelButton.Click
    ' Redirect to home page
    Response.Redirect("~/Default.aspx")
End Sub
```

▶ Task 5: Save the customer information

1. In the **InsertCustomer** class, prepare to save the customer input information to persistent storage when the user clicks the **Insert** button.

```
''' <summary>
''' Saves the current customer information and adds default values
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub customerInsertButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles customerInsertButton.Click
    ' Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name
    ' Add the user credit limit
    currentCustomer.CreditLimit = 50000
End Sub
```

- In the InsertCustomer.aspx.vb code window, replace the following code,

```
Protected Sub customerInsertButton_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
customerInsertButton.Click

End Sub
```

with:

```
''' <summary>
''' Saves the current customer information and adds default
values
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub customerInsertButton_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
customerInsertButton.Click
    ' Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name
    ' Add the user credit limit
    currentCustomer.CreditLimit = 50000
End Sub
```

2. Save the changes to **InsertCustomer.aspx.vb**.
   - In the CustomerManagement – Microsoft Visual Studio window, press CTRL+S.
3. Build the solution, and fix any errors.
   - In the CustomerManagement – Microsoft Visual Studio window, press SHIFT+CTRL+B.
4. Close Visual Studio 2010.
   - In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

Note: Notice that the validation successfully completed.

Note: The initial code for saving the customer information is created here. However, you will not create the final code for saving to the database until Module 8.

▶ Task 6: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 4

Lab Answer Key: Adding Functionality to a Microsoft® ASP.NET Web Form (Visual C#)

Contents:

Lab: Adding Functionality to a Microsoft ASP.NET Web Form

(C#)

Exercise 1: Implementing Code in a Web Application

▶ Task 1: Open an existing Web site

1. Log on to the 10267A-GEN-DEV virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M4\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M4\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Open the code-behind file for an existing Web Form

- Open the code-behind file of the InsertCustomer Web Form, by using the **View Code** context menu command.
  - In Solution Explorer, right-click the **InsertCustomer** Web Form, and then click **View Code**.

Exercise 2: Creating Event Procedures

▶ Task 1: Create an event procedure for the **Click** event of the **Insert** button

- Open the **InsertCustomer** Web Form in Design view, and create an event procedure for the **Click** event of the **Insert** button, by double-clicking the **Button** control.
  - In the InsertCustomer.aspx.cs code window, right-click anywhere, and then click **View Designer**.
  - In the InsertCustomer.aspx window, double-click the **Insert** button.

> **Note:** Notice that the initial event procedure for the **Click** event of the **CustomerInsertButton** control is added in the code window.

▶ Task 2: Create an event procedure for the **Click** event of the **Cancel** button

- Open the **InsertCustomer** Web Form in the Design view, and create an event procedure for the **Click** event of the **Cancel** button, by double-clicking the box next to the **Click** event in the Properties window, with the Button control selected in the Designer.
  - In the InsertCustomer.aspx.cs code window, right-click anywhere, and then click **View Designer**.
  - In the InsertCustomer.aspx window, click the **Cancel** button.
  - In the Properties window, click the **Events** icon, and then double-click the box next to the **Click** event.

> **Note:** Notice that the initial event procedure for the **Click** event of the **CustomerCancelButton** is added to the class in the code window.

▶ Task 3: Create an event procedure for the **Page_LoadComplete** event

- Add the following empty method to the **InsertCustomer** class.

```
protected void Page_LoadComplete(object sender, EventArgs e)
{

}
```

- In the InsertCustomer.aspx.cs code window, type the following code in the **InsertCustomer** public class.

```
protected void Page_LoadComplete(object sender, EventArgs e)
{

}
```

▶ Task 4: Create an event procedure for the **Page_Unload** event

- Add the following empty method to the **InsertCustomer** class.

```
protected void Page_Unload(object sender, EventArgs e)
{

}
```

- In the InsertCustomer.aspx.cs code window, type the following code in the **InsertCustomer** public class.

```
protected void Page_Unload(object sender, EventArgs e)
{

}
```

Exercise 3: Creating an Entity Component

▶ Task 1: Create an entity component

1. Create the **CustomerManagementEntities** class library project by using the **Class Library** project item in the **Add New Project** dialog box. Place the new project in the D:\Labfiles\Starter\M4\CS folder.
    a. In Solution Explorer, right-click **Solution 'CustomerManagement' (1 project)**, point to **Add**, and then click **New Project**.
    b. In the **Add New Project** dialog box, in the left pane, click **Visual C#**.
    c. In the middle pane, click **Class Library**.
    d. In the **Name** box, type **CustomerManagementEntities**.
    e. In the **Location** box, ensure that **D:\Labfiles\Starter\M4\CS** is entered, and then click **OK**.

> **Note:** Notice that the **CustomerManagementEntities** class library project is added in Solution Explorer.

2. Rename the default class file, Class1.cs, as **Customer.cs**.
    a. In Solution Explorer, right-click **Class1.cs**, click **Rename**, modify the name as **Customer.cs**, and then press ENTER.
    b. In the Microsoft Visual Studio message box, click **Yes**.

▶ Task 2: Add the class member fields

1. In the **Customer** class, add a region named **Class member fields**, just below the class declaration.

```
#region Class member fields
#endregion
```

- In the Customer.cs code window, type the following code in the **Customer** class, just below the class declaration.

```
#region Class member fields
#endregion
```

2. In the **Customer** class, within the **Class member fields** region, add a private member field for the first name of the customer, name the field **customerFirstName** of type **string**, , and initialize to **null**.

```
private string customerFirstName = null;
```

3. In the **Customer** class, within the **Class member fields** region, add a private member field named **customerLastName**, of type **string**, for the last name of the customer, and initialize to **null**.

```
private string customerLastName = null;
```

4. In the **Customer** class, within the **Class member fields** region, add a private member field named **customerAddress**, of type **string**, for the address of the customer, and initialize to **null**.

```
private string customerAddress = null;
```

5. Append the remaining backing fields by using a Code Snippet named **Customer class backing fields**. The Code Snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the declaration of the **customerAddress** backing field, and insert the snippet by clicking **Insert Snippet**.

   a. In the CustomerManagement − Microsoft Visual Studio window, right-click the line following the declaration of the **customerAddress** backing field, and then click **Insert Snippet**.

   b. In the context menu, double-click **My Code Snippets**, and then double-click **Customer class backing fields**.

```
private string customerZipCode = null;
private string customerCity = null;
private string customerState = null;
private Guid? customerCountryID = null;
private string customerPhone = null;
private string customerEmailAddress = null;
private string customerWebAddress = null;
private int customerCreditLimit = 0;
private bool customerNewsSubscriber = false;
private DateTime? customerCreatedDate = null;
private string customerCreatedBy = null;
private DateTime? customerModifiedDate = null;
private string customerModifiedBy = null;
```

▶ Task 3: Add the properties

1. In the **Customer** class, add a region named **Properties**, below the **Class member fields** region.

   - In the Customer.cs code window, type the following code in the **Customer** class, just below the **Class member fields** region.

   ```
   #region Properties
   #endregion
   ```

2. In the **Customer** class, within the **Properties** region, append an auto-implemented public property named **ID**, of nullable type **Guid**.

   ```
   /// <summary>
   /// The unique customer ID
   /// </summary>
   public Guid? ID { get; set; }
   ```

   - In the Customer.cs code window, within the **Properties** region, append the following code to the **Customer** class.

   ```
   /// <summary>
   /// The unique customer ID
   /// </summary>
   public Guid? ID { get; set; }
   ```

3. In the **Customer** class, within the **Properties** region, add a public property named **FirstName**, of type **string**, that sets and gets the private member backing field named **customerFirstName**. Ensure that the length is no longer than 50 characters by using the following lines of code.

```
/// <summary>
/// The customer first name
/// </summary>
public string FirstName
{
    get
    {
        return this.customerFirstName;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerFirstName = "";
        else
            // Only get the first 50 characters
            if (value.Length > 50)
                this.customerFirstName = value.Substring(0, 50);
            else
                this.customerFirstName = value;
    }
}
```

- In the Customer.cs code window, within the **Properties** region, append the following code to the **Customer** class.

```
/// <summary>
/// The customer first name
/// </summary>
public string FirstName
{
    get
    {
        return this.customerFirstName;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerFirstName = "";
        else
            // Only get the first 50 characters
            if (value.Length > 50)
                this.customerFirstName = value.Substring(0,
50);
            else
                this.customerFirstName = value;
    }
}
```

4. In the **Customer** class, within the **Properties** region, add a public property named **LastName**, of type **string**, that sets and gets the private member backing field named **customerLastName**. Ensure that the length is no longer than 30 characters by using the following lines of code.

```
/// <summary>
/// The customer last name
/// </summary>
public string LastName
{
    get
    {
        return this.customerLastName;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerLastName = "";
        else
            // Only get the first 30 characters
            if (value.Length > 30)
                this.customerLastName = value.Substring(0, 30);
            else
                this.customerLastName = value;
    }
}
```

- In the Customer.cs code window, within the **Properties** region, append the following code to the **Customer** class.

```
/// <summary>
/// The customer last name
/// </summary>
public string LastName
{
    get
    {
        return this.customerLastName;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerLastName = "";
        else
            // Only get the first 30 characters
            if (value.Length > 30)
                this.customerLastName = value.Substring(0,
30);
            else
                this.customerLastName = value;
    }
}
```

5. In the **Customer** class, within the **Properties** region, add a public property named **Address**, of type **string**, that sets and gets the private member backing field named **customerAddress**. Ensure that the length is no longer than 50 characters by using the following lines of code.

```
/// <summary>
/// The customer address, including street name, house number and
floor
/// </summary>
public string Address
{
    get
    {
        return this.customerAddress;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerAddress = "";
        else
            // Only get the first 50 characters
            if (value.Length > 50)
                this.customerAddress = value.Substring(0, 50);
            else
                this.customerAddress = value;
    }
}
```

- In the Customer.cs code window, within the **Properties** region, append the following code to the **Customer** class.

```
/// <summary>
/// The customer address, including street name, house number
and floor
/// </summary>
public string Address
{
    get
    {
        return this.customerAddress;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerAddress = "";
        else
            // Only get the first 50 characters
            if (value.Length > 50)
                this.customerAddress = value.Substring(0, 50);
            else
                this.customerAddress = value;
    }
}
```

6. Append the remaining properties within the **Properties** region by using a code snippet named **Customer class properties**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the declaration of the **Address** property, and insert the snippet by clicking **Insert Snippet**.

   a. In the CustomerManagement – Microsoft Visual Studio window, right-click the line following the declaration of the **Address** property, and then click **Insert Snippet**.

b. In the context menu, double-click **My Code Snippets**, and then double-click **Customer class properties**.

```
/// <summary>
/// The customer zip code or postal code
/// </summary>
public string ZipCode
{
    get
    {
        return this.customerZipCode;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerZipCode = "";
        else
            // Only get the first 10 characters
            if (value.Length > 10)
                this.customerZipCode = value.Substring(0, 10);
            else
                this.customerZipCode = value;
    }
}

/// <summary>
/// The name of the city in which the customer lives
/// </summary>
public string City
{
    get
    {
        return this.customerCity;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerCity = "";
        else
            // Only get the first 30 characters
            if (value.Length > 30)
                this.customerCity = value.Substring(0, 30);
            else
                this.customerCity = value;
    }
}
```

(*Code continued on the following page.*)

```
/// <summary>
/// The name of the state or region in which the customer
lives
/// </summary>
public string State
{
    get
    {
        return this.customerState;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerState = "";
        else
            // Only get the first 30 characters
            if (value.Length > 30)
                this.customerState = value.Substring(0, 30);
            else
                this.customerState = value;
    }
}

/// <summary>
/// The ID of the country in which the customer lives
/// </summary>
public Guid? CountryID
{
    get
    {
        return this.customerCountryID;
    }
    set
    {
        this.customerCountryID = value;
    }
}
```

(*Code continued on the following page.*)

```
/// <summary>
/// The customer phone number
/// </summary>
public string Phone
{
    get
    {
        return this.customerPhone;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerPhone = "";
        else
            // Only get the first 30 characters
            if (value.Length > 30)
                this.customerPhone = value.Substring(0, 30);
            else
                this.customerPhone = value;
    }
}

/// <summary>
/// The customer e-mail address
/// </summary>
public string EmailAddress
{
    get
    {
        return this.customerEmailAddress;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerEmailAddress = "";
        else
            // Only get the first 50 characters
            if (value.Length > 50)
                this.customerEmailAddress = value.Substring(0,
50);
            else
                this.customerEmailAddress = value;
    }
}
```

(*Code continued on the following page.*)

```
/// <summary>
/// The customer Web address
/// </summary>
public string WebAddress
{
    get
    {
        return this.customerWebAddress;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerWebAddress = "";
        else
            // Only get the first 80 characters
            if (value.Length > 80)
                this.customerWebAddress = value.Substring(0,
80);
            else
                this.customerWebAddress = value;
    }
}

/// <summary>
/// The current credit limit of the customer
/// </summary>
public int CreditLimit
{
    get
    {
        return this.customerCreditLimit;
    }
    set
    {
        // Negative value?
        if (value < 0)
            this.customerCreditLimit = 0;
        else
            this.customerCreditLimit = value;
    }
}
```

(*Code continued on the following page.*)

```
/// <summary>
/// Does the customer subscriber to news?
/// </summary>
public bool NewsSubscriber
{
    get
    {
        return this.customerNewsSubscriber;
    }
    set
    {
        this.customerNewsSubscriber = value;
    }
}

/// <summary>
/// The date the customer was created in the system
/// </summary>
public DateTime? CreatedDate
{
    get
    {
        return this.customerCreatedDate;
    }
    private set
    {
        // Date in the past?
        if (value < DateTime.Now)
            this.customerCreatedDate = DateTime.Now;
        else
            this.customerCreatedDate = value;
    }
}
```

(*Code continued on the following page.*)

```
/// <summary>
/// The name of the user creating the customer
/// </summary>
public string CreatedBy
{
    get
    {
        return this.customerCreatedBy;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerCreatedBy = "";
        else
            // Only get the first 15 characters
            if (value.Length > 15)
                this.customerCreatedBy = value.Substring(0,
15);
            else
                this.customerCreatedBy = value;
    }
}

/// <summary>
/// The date the customer was last modified in the system
/// </summary>
public DateTime? ModifiedDate
{
    get
    {
        return this.customerModifiedDate;
    }
    set
    {
        // Date in the past?
        if (value < DateTime.Now)
            this.customerModifiedDate = DateTime.Now;
        else
            this.customerModifiedDate = value;
    }
}
```

(*Code continued on the following page.*)

```
/// <summary>
/// The name of the user who last modified the customer
/// </summary>
public string ModifiedBy
{
    get
    {
        return this.customerModifiedBy;
    }
    set
    {
        // Null value?
        if (value == null)
            this.customerModifiedBy = "";
        else
            // Only get the first 15 characters
            if (value.Length > 15)
                this.customerModifiedBy = value.Substring(0,
15);
            else
                this.customerModifiedBy = value;
    }
}
```

▶ Task 4: Add the constructors

1. In the **Customer** class, add a region named **Constructors**, below the **Properties** region.

```
#region Constructors
#endregion
```

- In the Customer.cs code window, type the following code in the **Customer** class, just below the **Properties** region.

```
#region Constructors
#endregion
```

2. In the **Customer** class, within the **Constructors** region, add the default public parameterless constructor that initializes the **customerID** and **customerCreatedDate** member fields by using the public properties.

```
/// <summary>
/// Default parameterless constructor
/// </summary>
public Customer()
{
    // Initialize backing fields with default values
    this.ID = Guid.NewGuid();
    this.CreatedDate = DateTime.Now;
}
```

   - In the Customer.cs code window, within the **Constructors** region, append the following code to the **Customer** class.

```
/// <summary>
/// Default parameterless constructor
/// </summary>
public Customer()
{
    // Initialize backing fields with default values
    this.ID = Guid.NewGuid();
    this.CreatedDate = DateTime.Now;
}
```

3. Append the remaining properties within the **Constructors** region by using a code snippet named **Customer class constructors**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder. Place the cursor on and right-click the line following the default parameterless constructor, and insert the snippet by clicking **Insert Snippet**.

   a. In the CustomerManagement − Microsoft Visual Studio window, right-click the line following the default parameterless constructor, and then click **Insert Snippet**.

b. In the context menu, double-click **My Code Snippets**, and then double-click **Customer class constructors**.

```
/// <summary>
/// Initializes backing fields with passed and default values
/// </summary>
/// <param name="id"></param>
public Customer(Guid? id)
{
    // Initialize backing fields with passed and default
values
    this.ID = id;
    this.CreatedDate = DateTime.Now;
}

/// <summary>
/// Initializes with a value for all backing fields
/// </summary>
/// <param name="id"></param>
/// <param name="firstName"></param>
/// <param name="lastName"></param>
/// <param name="address"></param>
/// <param name="zipCode"></param>
/// <param name="city"></param>
/// <param name="state"></param>
/// <param name="countryID"></param>
/// <param name="phone"></param>
/// <param name="emailAddress"></param>
/// <param name="webAddress"></param>
/// <param name="creditLimit"></param>
/// <param name="newsSubscriber"></param>
/// <param name="createdDate"></param>
/// <param name="createdBy"></param>
/// <param name="modifiedDate"></param>
/// <param name="modifiedBy"></param>
public Customer(Guid? id, string firstName, string lastName,
string address,
    string zipCode, string city, string state, Guid?
countryID, string phone,
    string emailAddress, string webAddress, int creditLimit,
bool newsSubscriber,
    DateTime? createdDate, string createdBy, DateTime?
modifiedDate, string modifiedBy)
```

(*Code continued on the following page.*)

```
{
    // Initialize member backing fields with passed values
    this.ID = id;
    this.FirstName = firstName;
    this.LastName = lastName;
    this.Address = address;
    this.ZipCode = zipCode;
    this.City = city;
    this.State = state;
    this.CountryID = countryID;
    this.Phone = phone;
    this.EmailAddress = emailAddress;
    this.WebAddress = webAddress;
    this.CreditLimit = creditLimit;
    this.NewsSubscriber = newsSubscriber;

    if (createdDate != null)
        this.CreatedDate = createdDate;
    else
        this.CreatedDate = DateTime.Now;

    this.CreatedBy = createdBy;
    this.ModifiedDate = modifiedDate;
    this.ModifiedBy = modifiedBy;
}
```

4. Save the changes to Customer.cs.
    - In the CustomerManagement – Microsoft Visual Studio window, press the CTRL+S keys.
5. Build the component, and fix any errors.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build CustomerManagementEntities**.

▶ Task 5: Reference CustomerManagementEntities project from CustomerManagement project

- Add a reference to the CustomerManagement project by using the **Add Reference** dialog box. Reference the **CustomerManagementEntities** project from the **CustomerManagement** project.

  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M4 \CS\CustomerManagement**, and then click **Add Reference**.

  b. In the **Add Reference** dialog box, on the **Projects** tab, under **Project Name**, ensure that **CustomerManagementEntities** is selected, and then click **OK**.

▶ Task 6: Add a customer member declaration

1. Open the **InsertCustomer.aspx.cs** file.

   - In the CustomerManagement - Microsoft Visual Studio window, click **InsertCustomer.aspx.cs**.

2. In the InsertCustomer.aspx.cs code window, add a private class member declaration of the **Customer** class in the **CustomerManagementEntities** namespace named **currentCustomer**, and initialize to **null**.

```
private CustomerManagementEntities.Customer currentCustomer =
null;
```

   - In the InsertCustomer.aspx.cs window, after the code,

```
public partial class InsertCustomer : System.Web.UI.Page
{
```

     type the following code:

```
private CustomerManagementEntities.Customer currentCustomer =
null;
```

Exercise 4: Handling Page and Control Events

▶ Task 1: Instantiate the Customer object

1. Instantiate the **Customer** object by using the following code.

```
/// <summary>
/// Instantiates Customer object
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_Load(object sender, EventArgs e)
{
    // Instantiate Customer
    instantiateCustomerObject();
}
```

- In the InsertCustomer.aspx.cs code window, replace the following code,

```
protected void Page_Load(object sender, EventArgs e)
{

}
```

with:

```
/// <summary>
/// Instantiates Customer object
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_Load(object sender, EventArgs e)
{
    // Instantiate Customer
    instantiateCustomerObject();
}
```

2. Append the following code to the **InsertCustomer** class.

```
/// <summary>
/// Instantiates and populates the Customer member object
/// </summary>
private void instantiateCustomerObject()
{
    // First time loading page?
    if (!this.IsPostBack)
        // Instantiate new Customer object
        currentCustomer = new CustomerManagementEntities.Customer(
            null, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
            CustomerAddressTextBox.Text,
CustomerZipCodeTextBox.Text,
            CustomerCityTextBox.Text, CustomerStateTextBox.Text,
            null, CustomerPhoneTextBox.Text,
CustomerEmailAddressTextBox.Text,
            CustomerWebAddressTextBox.Text, -1,
CustomerNewsSubscriberCheckBox.Checked,
            DateTime.Now, "", null, "");
}
```

- In the InsertCustomer.aspx.cs code window, append the following code to the **InsertCustomer** class.

```
/// <summary>
/// Instantiates and populates the Customer member object
/// </summary>
```

```
private void instantiateCustomerObject()
```

```
{
    // First time loading page?
    if (!this.IsPostBack)
        // Instantiate new Customer object
        currentCustomer = new
CustomerManagementEntities.Customer(
            null, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
            CustomerAddressTextBox.Text,
CustomerZipCodeTextBox.Text,
            CustomerCityTextBox.Text,
CustomerStateTextBox.Text,
            null, CustomerPhoneTextBox.Text,
CustomerEmailAddressTextBox.Text,
            CustomerWebAddressTextBox.Text, -1,
CustomerNewsSubscriberCheckBox.Checked,
            DateTime.Now, "", null, "");
}
```

▶ Task 2: Populate the UI controls

- In the **InsertCustomer** class, populate the server controls in the user interface (UI) by using the values from the private **Customer** object, **currentCustomer**.

```
/// <summary>
/// Populates UI controls
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_LoadComplete(object sender, EventArgs e)
{
    // Populate the UI controls
    populateUI();
}

/// <summary>
/// Populates the UI controls with the values in the
/// current Customer object
/// </summary>
private void populateUI()
{
    CustomerFirstNameTextBox.Text = currentCustomer.FirstName;
    CustomerLastNameTextBox.Text = currentCustomer.LastName;
    CustomerAddressTextBox.Text = currentCustomer.Address;
    CustomerZipCodeTextBox.Text = currentCustomer.ZipCode;
    CustomerCityTextBox.Text = currentCustomer.City;
    CustomerStateTextBox.Text = currentCustomer.State;

    if (currentCustomer.CountryID.HasValue)
        CustomerCountryDropDownList.SelectedValue =
currentCustomer.CountryID.Value.ToString();
    else
        CustomerCountryDropDownList.SelectedIndex = -1;

    CustomerPhoneTextBox.Text = currentCustomer.Phone;
    CustomerEmailAddressTextBox.Text =
currentCustomer.EmailAddress;
    CustomerWebAddressTextBox.Text = currentCustomer.WebAddress;
    CustomerCreditLimitTextBox.Text =
currentCustomer.CreditLimit.ToString();
    CustomerNewsSubscriberCheckBox.Checked =
currentCustomer.NewsSubscriber;
}
```

- In the InsertCustomer.aspx.cs code window, replace the following code,

```
protected void Page_LoadComplete(object sender, EventArgs e)
{
}
```

with:

```
/// <summary>
/// Populates UI controls
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_LoadComplete(object sender, EventArgs e)
{
    // Populate the UI controls
    populateUI();
}

/// <summary>
/// Populates the UI controls with the values in the
/// current Customer object
/// </summary>
private void populateUI()
{
    CustomerFirstNameTextBox.Text = currentCustomer.FirstName;
    CustomerLastNameTextBox.Text = currentCustomer.LastName;
    CustomerAddressTextBox.Text = currentCustomer.Address;
    CustomerZipCodeTextBox.Text = currentCustomer.ZipCode;
    CustomerCityTextBox.Text = currentCustomer.City;
    CustomerStateTextBox.Text = currentCustomer.State;

    if (currentCustomer.CountryID.HasValue)
        CustomerCountryDropDownList.SelectedValue =
currentCustomer.CountryID.Value.ToString();
    else
        CustomerCountryDropDownList.SelectedIndex = -1;

    CustomerPhoneTextBox.Text = currentCustomer.Phone;
    CustomerEmailAddressTextBox.Text =
currentCustomer.EmailAddress;
    CustomerWebAddressTextBox.Text =
currentCustomer.WebAddress;
    CustomerCreditLimitTextBox.Text =
currentCustomer.CreditLimit.ToString();
    CustomerNewsSubscriberCheckBox.Checked =
currentCustomer.NewsSubscriber;
}
```

▶ Task 3: Destroy the objects

- In the **InsertCustomer** class, destroy the objects used in the class that are not automatically handled by the garbage collector.

```
/// <summary>
/// Destroys objects
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_Unload(object sender, EventArgs e)
{
    // Destroy Customer object
    currentCustomer = null;
}
```

- In the InsertCustomer.aspx.cs code window, replace the following code,

```
protected void Page_Unload(object sender, EventArgs e)
{

}
```

with:

```
/// <summary>
/// Destroys objects
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_Unload(object sender, EventArgs e)
{
    // Destroy Customer object
    currentCustomer = null;
}
```

▶ Task 4: Handle the user cancellation

- In the **InsertCustomer** class, handle user cancellation by redirecting to the home page.

```
/// <summary>
/// Redirects to home page
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerCancelButton_Click(object sender, EventArgs
e)
{
    // Redirect to home page
    Response.Redirect("~/Default.aspx");
}
```

- In the InsertCustomer.aspx.cs code window, replace the following code,

```
protected void CustomerCancelButton_Click(object sender,
EventArgs e)
{

}
```

with:

```
/// <summary>
/// Redirects to home page
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerCancelButton_Click(object sender,
EventArgs e)
{
    // Redirect to home page
    Response.Redirect("~/Default.aspx");
}
```

▶ Task 5: Save the customer information

1. In the **InsertCustomer** class, prepare to save the customer input information to persistent storage when the user clicks the **Insert** button.

```
/// <summary>
/// Saves the current customer information and adds default values
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerInsertButton_Click(object sender, EventArgs
e)
{
    // Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name;
    // Add the user credit limit
    currentCustomer.CreditLimit = 50000;
}
```

- In the InsertCustomer.aspx.cs code window, replace the following code,

```
protected void CustomerInsertButton_Click(object sender,
EventArgs e)
{

}
```

with:

```
/// <summary>
/// Saves the current customer information and adds default
values
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerInsertButton_Click(object sender,
EventArgs e)
{
    // Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name;
    // Add the user credit limit
    currentCustomer.CreditLimit = 50000;
}
```

2. Save the changes to InsertCustomer.aspx.cs.
   - In the CustomerManagement – Microsoft Visual Studio window, press CTRL+S.
3. Build the solution, and fix any errors.
   - In the CustomerManagement – Microsoft Visual Studio window, press SHIFT+CTRL+B.
4. Close Visual Studio 2010.
   - In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

Note: Notice that the validation successfully completed.

Note: The initial code for saving the customer information is created here. However, you will create the final code for saving to the database in Module 8.

▶ Task 6: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 5

Lab Answer Key: Implementing Master Pages and User Controls (Visual Basic)

Contents:

Lab: Implementing Master Pages and User Controls

Exercise 1: Adding and Applying a Master Page

▶ Task 1: Add a master page to an existing Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M5\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M5\VB\CustomerManagement.sln**, and then click **Open**.
4. Add a new master page named **Site.master**, to the **CustomerManagement** Web site.
   a. In Solution Explorer, right-click **D:\Labfiles\Starter\M5\VB \CustomerManagement**, and then click **Add New Item**.
   b. In the **Add New Item - D:\Labfiles\Starter\M5\VB \CustomerManagement** dialog box, in the left pane, ensure that **Visual Basic** is selected, and in the middle pane, click **Master Page**.
   c. In the **Name** box, type **Site.master**.
   d. In the **Add New Item - D:\Labfiles\Starter\M5\VB \CustomerManagement** dialog box, ensure that the **Place code in separate file** check box is selected and the **Select master page** check box is cleared, and then click **Add**.

▶ Task 2: Initialize the style controls and elements on the master page

1. In the Site.master window, add an **id** property to the **head** element by using the following markup.

```
<head runat="server" id="MainHead">
```

   - In the Site.master window, change the following markup,

```
<head runat="server">
```

     to:

```
<head runat="server" id="MainHead">
```

2. In the Site.master window, change the **id** property of the **form** element to **MainForm**.

```
<form id="MainForm" runat="server">
```

   - In the Site.master window, change the following markup,

```
<form id="form1" runat="server">
```

     to:

```
<form id="MainForm" runat="server">
```

3. Reference the **Site.css** file in the Site.master Web Form, relative to the root folder, by placing the following markup next to the closing tag of the **title** element.

```
<link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />
```

   a. In Solution Explorer, under D:\Labfiles\Starter\M5\VB \CustomerManagement, expand **Styles**, and then drag the **Site.css** file to the **Site.master** master page next to the closing tag of the **title** element, to add the following markup.

```
<link href="Styles/Site.css" rel="stylesheet" type="text/css"
/>
```

b. In the Site.master window, modify the **href** attribute of the self-closing **link** element to include a tilde (~) and a forward slash (/) character at the beginning of the path.

```
<link href="~/Styles/Site.css" rel="stylesheet"
type="text/css" />
```

4. In the Site.master window, add a **Class** property to the **body** element by using the following markup.

```
<body class="template">
```

- In the Site.master window, change the following markup,

```
<body>
```

to:

```
<body class="template">
```

5. In the Site.master window, add a **Class** property to the **div** element by using the following markup.

```
<div class="content">
```

- In the Site.master window, change the following markup,

```
<div>
```

to:

```
<div class="content">
```

6. In the Site.master window, set the value of the **title** element to **Contoso Customer Management** by using the following markup.

```
<title>Contoso Customer Management</title>
```

- In the Site.master window, change the following markup,

```
<title></title>
```

to:

```
<title>Contoso Customer Management</title>
```

▶ Task 3: Define a ContentPlaceHolder control on the master page

1. Remove the **ContentPlaceHolder** from the **head** element.

```
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
```

- In the Site.master window, remove the **ContentPlaceHolder** by removing the following markup within the **head** element.

```
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
```

2. Change the **id** property of the **ContentPlaceHolder** control within the **div** element to **MainContentPlaceHolder**.

```
<div class="content">
<asp:ContentPlaceHolder id="MainContentPlaceHolder"
runat="server">

</asp:ContentPlaceHolder>
</div>
```

- In the Site.master window, change the **id** property of the **ContentPlaceHolder** control within the **div** element to **MainContentPlaceHolder**.

```
<div class="content">
<asp:ContentPlaceHolder id="MainContentPlaceHolder"
runat="server">

</asp:ContentPlaceHolder>
</div>
```

3. Save the master page.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**, or press the CTRL+S keys.

Exercise 2: Converting Web Forms to Content Pages and User Controls

▶ Task 1: Convert the default Web Form into a content page

1. Open the **Default.aspx** Web Form.
   - In Solution Explorer, under D:\Labfiles\Starter\M5\VB\CustomerManagement, right-click **Default.aspx**, and then click **View Markup**.
2. In the Default.aspx window, in the **Page** directive, add a **MasterPageFile** property with a value of **~/Site.master**, by using the following markup.

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="Default.aspx.vb" Inherits="_Default"
MasterPageFile="~/Site.master" %>
```

   - In the Default.aspx window, change the following markup,

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="Default.aspx.vb" Inherits="_Default" %>
```

     to:

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="Default.aspx.vb" Inherits="_Default"
MasterPageFile="~/Site.master" %>
```

3. Remove the top-level HTML elements from the **Default** Web Form.

> **Note:** You should take care not to delete the **div** element and its content within the **form** element.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="Styles/Site.css" rel="stylesheet" type="text/css"
/>
</head>
<body>
<form id="form1" runat="server">
</form>
</body>
</html>
```

- In the Default.aspx window, remove the following top-level HTML elements from the Web Form, except the **div** element.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="Styles/Site.css" rel="stylesheet"
type="text/css" />
</head>
<body>
<form id="form1" runat="server">
</form>
</body>
</html>
```

4. In the Default.aspx window, add a server-side **Content** control.

```
<asp:Content ID="MainContent"
ContentPlaceHolderID="MainContentPlaceHolder" runat="server">
</asp:Content>
```

   - In the Default.aspx window, add the following markup after the **Page** directive, to add a server-side **Content** control.

```
<asp:Content ID="MainContent"
ContentPlaceHolderID="MainContentPlaceHolder" runat="server">
</asp:Content>
```

5. In the Default.aspx window, move the following markup and place it after the opening tag of the **form** element on the **Site.master** master page.

```
<div class="apptitle">
    <asp:Literal ID="AppTitleLiteral" runat="server"
Text="Customer Management"></asp:Literal>
</div>
```

   a. In the Default.aspx window, select the following markup, right-click anywhere on the markup, and then click **Cut**.

```
<div class="apptitle">
    <asp:Literal ID="AppTitleLiteral" runat="server"
Text="Customer Management"></asp:Literal>
</div>
```

   b. In the CustomerManagement – Microsoft Visual Studio window, click **Site.master**.

   c. In the Site.master window, place the cursor after the opening tag of the **form** element,

```
<form id="MainForm" runat="server">
```

   right-click, and then click **Paste**.

6. Format the Site.master master page, by pressing CTRL+K, and then pressing CTRL+D.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, and then press CTRL+D.

7. Save the changes to the **Site.master** master page.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**, or press CTRL+S.
8. Format the Default.aspx Web Form, by pressing CTRL+K, and then pressing CTRL+D.
   a. In the CustomerManagement – Microsoft Visual Studio window, click **Default.aspx**.
   b. In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, and then CTRL+D.
9. Save and close the **Default.aspx** Web Form.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, click the **Close** button.

▶ Task 2: Add navigation to the master page

1. Add a breadcrumb to the master page by adding a **SiteMapPath** control named **MainSiteMapPath**, wrapped in a **div** element with a class attribute value of **siteMapPath**. Add the new **div** element below the existing **div** element with a **class** attribute value of **appTitle**.
   a. In the Site.master window, place the cursor after the closing tag of the **div** element, with a **class** attribute value of **appTitle**, and then press ENTER.

```
<div class="appTitle">
    <asp:Literal ID="AppTitleLiteral" runat="server"
Text="Customer Management"></asp:Literal>
</div>
```

   b. In the Toolbox, expand HTML, and then double-click **Div**.
   c. Add a **class** attribute to the opening **div** tag with a value of **siteMapPath**.

```
<div class="siteMapPath">
```

   d. In the Site.master window, place the cursor after the opening tag of the **div** element with a **class** attribute value of **siteMapPath**, and then press ENTER.

e. In the Toolbox, expand **Navigation**, and then double-click **SiteMapPath**.

f. Change the **id** property of the **SiteMapPath** control to **MainSiteMapPath**, make the opening control tag self-closing, and delete the closing tag.

```
<asp:SiteMapPath ID="MainSiteMapPath" runat="server" />
```

2. Add a menu to the master page by adding a **Menu** control named **MainMenu**, wrapped in a **div** element with a class attribute value of **menu**. Add the new **div** element below the existing **div** element with a **class** attribute value of **siteMapPath**.

a. In the Site.master window, place the cursor after the closing tag of the **div** element with a **class** attribute value of **siteMapPath**, and then press ENTER.

```
<div class="siteMapPath">
    <asp:SiteMapPath ID="MainSiteMapPath" runat="server" />
</div>
```

b. In the Toolbox, expand HTML, and then double-click **div**.

c. Add a **class** attribute to the opening **div** tag with a value of **menu**.

```
<div class="menu">
```

d. In the Site.master window, place the cursor after the opening tag of the **div** element, with a **class** attribute value of **menu**, and then press ENTER.

e. In the Toolbox, expand **Navigation**, and then double-click **Menu**.

f. Change the **id** property of the **Menu** control to **MainMenu**.

```
<asp:Menu ID="MainMenu" runat="server">
</asp:Menu>
```

3. Make the menu layout horizontal by applying the **Orientation** attribute.

```
Orientation="Horizontal"
```

4. Ensure that the built-in image that indicates if a static menu item has a child menu is not displayed, by setting the **StaticEnableDefaultPopOutImage** attribute.

```
StaticEnableDefaultPopOutImage="false"
```

5. Get the items for the **Menu** control from the **MainSiteMapDataSource** data source control, by applying the DataSourceID attribute.

```
DataSourceID="MainSiteMapDataSource"
```

6. Add the following child elements to the **Menu** control, by placing them between the opening and closing **Menu** tags.

```
<StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px"
/>
<StaticHoverStyle BackColor="White" ForeColor="Black" />
<DynamicHoverStyle BackColor="White" ForeColor="Black" />
<DynamicMenuItemStyle ItemSpacing="2px" HorizontalPadding="5px"
VerticalPadding="2px" />
```

7. Add a **SiteMapDataSource** control named **MainSiteMapDataSource** to the master page, after the closing tag of the **div** element, with a **class** attribute value of **menu**. The **SiteMapDataSource** control should not show the starting node.

```
<asp:SiteMapDataSource ID="MainSiteMapDataSource" runat="server"
ShowStartingNode="false" />
```

   a. In the Site.master window, place the cursor after the closing tag of the **div** element, with a **class** attribute value of **menu**, and then press ENTER.

   b. In the Toolbox, expand **Data**, and then double-click **SiteMapDataSource**.

   c. Change the **id** property of the **SiteMapDataSource** control to **MainSiteMapDataSource**.

   d. Make sure that the starting node is not shown, by applying the **ShowStartingNode** attribute.

```
ShowStartingNode="false"
```

8. Format the Site.master master page.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, and then CTRL+D.

9. Add the **D:\Labfiles\Starter\M5\web.sitemap** site map file to the project.

   a. In Solution Explorer, right-click **D:\Labfiles\Starter\M5\VB \CustomerManagement**, and then click **Add Existing Item**.

b. In the **Add Existing Item** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M5\web.sitemap**, and then click **Add**.

10. Modify the **div.menu** style, by using the Manage Styles window. The menu style must have the following definition:

    - Category: **Position**
    - position: **relative**
    - z-index: **1**
    - Top: **62px**

    a. Open the Manage Styles window, by clicking **Manage Styles** on the **View** menu.

    b. In the **Site.css** list, right-click **div.menu**, and then click **Modify Style**.

    c. In the **Modify Style** dialog box, in the **Category** list, click **Position**, in the **position** box, click **relative**, in the **z-index** box, type **1**, in the **top** box, type **62px**, and then click **OK**.

11. Add a **siteMapPath** style, by using the Manage Styles window.

    a. In the Manage Styles window, click the **New Style** button.

    b. In the New Style window, in the **Selector** box, type **div.siteMapPath**, in the **Define in** list, click **Existing Style Sheet**, and in the **URL** box, click **Site.css**.

    c. In the **Category** list, click **Position**, in the **position** box, click **fixed**, and in the **top** box, type **42px**.

    d. In the **Category** list, click **Box**, in the **padding** section, clear the **Same for all** check box, in the bottom box, type **5px**, and then click **OK**.

    e. In the **Manage Styles** window, click the **Close** button.

12. Save all modified files, and run the Web site.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**, or press CTRL+SHIFT+S.

    b. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.

    c. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Convert the Web Form into a user control

1. Open the **InsertCustomer.aspx** Web Form, and change its **Page** directive to a **Control** directive.

```
<%@ Control Language="VB" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.vb" Inherits="InsertCustomer" %>
```

   a. In Solution Explorer, under D:\Labfiles\Starter\M5\VB\CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View Markup**.

   b. In the InsertCustomer.aspx window, change the following markup,

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="InsertCustomer.aspx.vb" Inherits="InsertCustomer" %>
```

   to:

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="InsertCustomer.aspx.vb" Inherits="InsertCustomer" %>
```

2. Add a **ClassName** property with a value of **Customer** to the **Control** directive.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="InsertCustomer.aspx.vb" Inherits="InsertCustomer"
ClassName="Customer" %>
```

   - In the InsertCustomer.aspx window, add a **ClassName** property with a value of **Customer** to the **Control** directive.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="InsertCustomer.aspx.vb" Inherits="InsertCustomer"
ClassName="Customer" %>
```

3. Change the **Inherits** property value from **InsertCustomer** to **Customer**.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="InsertCustomer.aspx.vb" Inherits="Customer"
ClassName="Customer" %>
```

   - In the InsertCustomer.aspx window, change the value of the **Inherits** property from **InsertCustomer** to **Customer**.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="InsertCustomer.aspx.vb" Inherits="Customer"
ClassName="Customer" %>
```

4. Change the **CodeFile** property value from **InsertCustomer.aspx.vb** to **Customer.aspx.vb**.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="Customer.aspx.vb" Inherits="Customer"
ClassName="Customer" %>
```

   - In the InsertCustomer.aspx window, change the value of the **CodeFile** property from **InsertCustomer.aspx.vb** to **Customer.aspx.vb**.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="Customer.aspx.vb" Inherits="Customer"
ClassName="Customer" %>
```

5. Remove all the top-level HTML elements, such as the **DOCTYPE**, **html**, **head**, **body**, **title**, and **form** elements.

6. Format the document.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, and then press CTRL+D.

Note: After removing all top-level HTML elements, you can view the following markup in the InsertCustomer.aspx window.

```
<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="Customer.ascx.vb" Inherits="Customer"
    ClassName="Customer" %>
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerFirstNameLabel" runat="server"
Text="First Name:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerFirstNameTextBox" runat="server"
MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerLastNameLabel" runat="server"
Text="Last Name:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerLastNameTextBox"
runat="server"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerAddressLabel" runat="server"
Text="Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerAddressTextBox" runat="server"
```

(*Code continued on the following page.*)

```
MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerZipCodeLabel" runat="server"
Text="Zip Code:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerZipCodeTextBox" runat="server"
MaxLength="10"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCityLabel" runat="server"
Text="City:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerCityTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerStateLabel" runat="server"
Text="State:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerStateTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCountryLabel" runat="server"
Text="Country:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:DropDownList ID="CustomerCountryDropDownList"
runat="server">
            </asp:DropDownList>
        </div>
```

(*Code continued on the following page.*)

```
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerPhoneLabel" runat="server"
Text="Phone:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerPhoneTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerEmailAddressLabel" runat="server"
Text="Email Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerEmailAddressTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerWebAddressLabel" runat="server"
Text="Web Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerWebAddressTextBox" runat="server"
MaxLength="80"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCreditLimitLabel" runat="server"
Text="Credit Limit:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerCreditLimitTextBox"
runat="server" MaxLength="10"></asp:TextBox>
        </div>
    </div>
```

(*Code continued on the following page.*)

```
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerNewsSubscriberLabel" runat="server"
Text="News Subscriber:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:CheckBox ID="CustomerNewsSubscriberCheckBox"
runat="server" />
        </div>
    </div>
    <div class="customerTableFooter">
        <asp:Button ID="CustomerInsertButton" runat="server"
Text="Insert" />
         <asp:Button ID="CustomerCancelButton" runat="server"
Text="Cancel" />
    </div>
</div>
```

7. Save the **InsertCustomer.aspx** Web Form.
    - In the CustomerManagement – Microsoft Visual Studio window, press CTRL+S.
8. Change the Web Form name from **InsertCustomer.aspx** to **Customer.ascx**.
    - In Solution Explorer, under D:\Labfiles\Starter\M5\VB\CustomerManagement, right-click **InsertCustomer.aspx**, click **Rename**, modify the file name as **Customer.ascx**, and then press ENTER.
    - In the Microsoft Visual Studio message box, click **Yes**.
9. Open the **Customer.ascx.vb** user control code-behind file, change the class name to **Customer**, and change its base class from **System.Web.UI.Page** to **System.Web.UI.UserControl**.
    a. In Solution Explorer, under D:\Labfiles\Starter\M5\VB\CustomerManagement, right-click **Customer.ascx**, and then click **View Code**.

b. In the Customer.ascx.vb code window, change the following markup,

```
Partial Class InsertCustomer
    Inherits System.Web.UI.Page
```

to:

```
Partial Class Customer
    Inherits System.Web.UI.UserControl
```

10. Move the content from the **Page_LoadComplete** event method, and append it to the **Page_Load** event method.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Instantiate Customer
    instantiateCustomerObject()
    ' Populate the UI controls
    populateUI()
End Sub
```

a. In the Customer.ascx.vb code window, in the **Page_LoadComplete** event method, select the following code, right-click, and then click **Cut**.

```
' Populate the UI controls
populateUI()
```

b. In the Customer.ascx.vb code window, paste the code within the **Page_Load** event method.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Instantiate Customer
    instantiateCustomerObject()
    ' Populate the UI controls
    populateUI()
End Sub
```

11. Remove the **Page_LoadComplete** event method.

```
''' <summary>
''' Populates UI controls
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_LoadComplete(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.LoadComplete
End Sub
```

- In the Customer.ascx.vb code window, delete the following code to remove the **Page_LoadComplete** event method.

```
''' <summary>
''' Populates UI controls
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_LoadComplete(ByVal sender As Object, ByVal
e As System.EventArgs) Handles Me.LoadComplete
End Sub
```

12. Save the modified files, and close the **Customer.ascx.vb** user control.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**, or press CTRL+SHIFT+S.

    b. In the Customer.ascx.vb code window, click the **Close** button.

▶ Task 4: Create a content page and insert a user control

1. Add a new content page named **InsertCustomer.aspx**, with a code-behind file based on the **Site.master** master page.

    a. In Solution Explorer, right-click **D:\Labfiles\Starter\M5\VB \CustomerManagement**, and then click **Add New Item**.

    b. In the **Add New Item - D:\Labfiles\Starter\M5\VB \CustomerManagement** dialog box, in the left pane, ensure that **Visual Basic** is selected.

c. In the middle pane, click **Web Form**, in the **Name** box, type **InsertCustomer.aspx**, ensure that the **Place code in separate file** check box is selected, select the **Select master page** check box, and then click **Add**.

d. In the **Select a Master Page** dialog box, in the **Contents of folder** box, click **Site.master**, and then click **OK**.

2. Open the **InsertCustomer.aspx** content page in the Design view, and drag the **Customer.ascx** user control to the **MainContentPlaceHolder** control.

a. In the InsertCustomer.aspx window, click **Design**.

b. In the InsertCustomer.aspx window, click the **MainContentPlaceHolder** control.

c. In Solution Explorer, drag the **Customer.ascx** user control to the **MainContentPlaceHolder** control.

d. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.

e. In the InsertCustomer.aspx window, click the **Close** button.

3. Run the **CustomerManagement** Web application.

a. In Solution Explorer, click **Default.aspx**.

b. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

4. Verify the Contoso Customer Management Web site.

a. In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.

Note: Notice that the new user control displays on the InsertCustomer Web Form.

b. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

c. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 5: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 5

Lab Answer Key: Implementing Master Pages and User Controls (Visual C#)

Contents:

Lab: Implementing Master Pages and User Controls

(C#)

Exercise 1: Adding and Applying a Master Page

▶ Task 1: Add a master page to an existing Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M5\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M5\CS\CustomerManagement.sln**, and then click **Open**.
4. Add a new master page named **Site.master**, to the **CustomerManagement** Web site.
   a. In Solution Explorer, right-click **D:\Labfiles\Starter\M5\CS \CustomerManagement**, and then click **Add New Item**.
   b. In the **Add New Item - D:\Labfiles\Starter\M5\CS \CustomerManagement**\ dialog box, in the left pane, ensure that **Visual C#** is selected, and in the middle pane, click **Master Page**.
   c. In the **Name** box, type **Site.master**.
   d. In the **Add New Item - D:\Labfiles\Starter\M5\CS \CustomerManagement**\ dialog box, ensure that the **Place code in separate file** check box is selected and the **Select master page** check box is not selected, and then click **Add**.

▶ Task 2: Initialize the style controls and elements on the master page

1. In the Site.master window, add an **id** property to the **head** element by using the following markup.

```
<head runat="server" id="MainHead">
```

   - In the Site.master window, change the following markup,

```
<head runat="server">
```

     to:

```
<head runat="server" id="MainHead">
```

2. In the Site.master window, change the **id** property of the **form** element to **MainForm**.

```
<form id="MainForm" runat="server">
```

   - In the Site.master window, change the following markup,

```
<form id="form1" runat="server">
```

     to:

```
<form id="MainForm" runat="server">
```

3. Reference the **Site.css** file in the Site.master Web Form, relative to the root folder, by placing the following markup next to the closing tag of the **title** element.

```
<link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />
```

   a. In Solution Explorer, under D:\Labfiles\Starter\M5\CS \CustomerManagement, expand **Styles**, and then drag the **Site.css** file to the **Site.master** master page next to the closing tag of the **title** element, to add the following markup.

```
<link href="Styles/Site.css" rel="stylesheet" type="text/css"
/>
```

b. In the Site.master window, modify the **href** attribute of the self-closing **link** element to include a tilde (~) and a forward slash (/) character at the beginning of the path.

```
<link href="~/Styles/Site.css" rel="stylesheet"
type="text/css" />
```

4. In the Site.master window, add a **Class** property to the **body** element by using the following markup.

```
<body class="template">
```

- In the Site.master window, change the following markup,

```
<body>
```

to:

```
<body class="template">
```

5. In the Site.master window, add a **Class** property to the **div** element by using the following markup.

```
<div class="content">
```

- In the Site.master window, change the following markup,

```
<div>
```

to:

```
<div class="content">
```

6. In the Site.master window, set the value of the **title** element to **Contoso Customer Management** by using the following markup.

```
<title>Contoso Customer Management</title>
```

- In the Site.master window, change the following markup,

```
<title></title>
```

to:

```
<title>Contoso Customer Management</title>
```

▶ Task 3: Define a ContentPlaceHolder control on the master page

1. Remove the **ContentPlaceHolder** from the **head** element.

```
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
```

- In the Site.master window, remove the **ContentPlaceHolder** by removing the following markup within the **head** element.

```
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
```

2. Change the **id** property of the **ContentPlaceHolder** control within the **div** element to **MainContentPlaceHolder**.

```
<div class="content">
<asp:ContentPlaceHolder id="MainContentPlaceHolder"
runat="server">

</asp:ContentPlaceHolder>
</div>
```

- In the Site.master window, change the **id** property of the **ContentPlaceHolder** control within the **div** element to **MainContentPlaceHolder**.

```
<div class="content">
<asp:ContentPlaceHolder id="MainContentPlaceHolder"
runat="server">

</asp:ContentPlaceHolder>
</div>
```

3. Save the master page.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**, or press the CTRL+S keys.

Exercise 2: Converting Web Forms to Content Pages and User Controls

▶ Task 1: Convert the default Web Form into a content page

1. Open the **Default.aspx** Web Form.
   - In Solution Explorer, under D:\Labfiles\Starter\M5\CS \CustomerManagement, right-click **Default.aspx**, and then click **View Markup**.
2. In the Default.aspx window, in the **Page** directive, add a **MasterPageFile** property, with a value of **~/Site.master**, by using the following markup.

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs"
Inherits="_Default" MasterPageFile="~/Site.master" %>
```

   - In the Default.aspx window, change the following markup,

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs"
Inherits="_Default"%>
```

     to:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs"
Inherits="_Default" MasterPageFile="~/Site.master" %>
```

3. Remove the top-level HTML elements from the **Default** Web Form.

> **Note:** You should take care not to delete the **div** element and its content within the **form** element.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="Styles/Site.css" rel="stylesheet" type="text/css"
/>
</head>
<body>
<form id="form1" runat="server">
</form>
</body>
</html>
```

- In the Default.aspx window, remove the following top-level HTML elements from the Web Form, except the **div** element.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="Styles/Site.css" rel="stylesheet"
type="text/css" />
</head>
<body>
<form id="form1" runat="server">
</form>
</body>
</html>
```

4. In the Default.aspx window, add a server-side **Content** control.

```
<asp:Content ID="MainContent"
ContentPlaceHolderID="MainContentPlaceHolder" runat="server">
</asp:Content>
```

   - In the Default.aspx window, add the following markup after the **Page** directive, to add a server-side **Content** control.

```
<asp:Content ID="MainContent"
ContentPlaceHolderID="MainContentPlaceHolder" runat="server">
</asp:Content>
```

5. In the Default.aspx window, move the following markup and place it after the opening tag of the **form** element on the **Site.master** master page.

```
<div class="appTitle">
    <asp:Literal ID="AppTitleLiteral" runat="server"
Text="Customer Management"></asp:Literal>
</div>
```

   a. In the Default.aspx window, select the following markup, right-click anywhere on the markup, and then click **Cut**.

```
<div class="appTitle">
    <asp:Literal ID="AppTitleLiteral" runat="server"
Text="Customer Management"></asp:Literal>
</div>
```

   b. In the CustomerManagement – Microsoft Visual Studio window, click **Site.master**.

   c. In the Site.master window, place the cursor after the opening tag of the **form** element,

```
<form id="MainForm" runat="server">
```

   right-click, and then click **Paste**.

6. Format the **Site.master** master page by pressing CTRL+K, and then pressing CTRL+D.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, and then press CTRL+D.

7. Save the changes to the **Site.master** master page.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**.
8. Format the Default.aspx Web Form, by pressing CTRL+K, and then pressing CTRL+D.
   a. In the CustomerManagement – Microsoft Visual Studio window, click **Default.aspx**.
   b. In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, CTRL+D.
9. Save and close the Default.aspx Web Form.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, click the **Close** button.

▶ Task 2: Add navigation to the master page

1. Add a breadcrumb to the master page by adding a **SiteMapPath** control named **MainSiteMapPath**, wrapped in a **div** element with a class attribute value of **siteMapPath**. Add the new **div** element below the existing **div** element with a **class** attribute value of **appTitle**.
   a. In the Site.master window, place the cursor after the closing tag of the **div** element, with a **class** attribute value of **appTitle**, and then press ENTER.

```
<div class="appTitle">
    <asp:Literal ID="AppTitleLiteral" runat="server"
Text="Customer Management"></asp:Literal>
</div>
```

   b. In the Toolbox, expand HTML, and then double-click **Div**.
   c. Add a **class** attribute to the opening **div** tag with a value of **siteMapPath**.

```
<div class="siteMapPath">
```

   d. In the Site.master window, place the cursor after the opening tag of the **div** element, with a **class** attribute value of **siteMapPath**, and then press ENTER.

e. In the Toolbox, expand **Navigation**, and then double-click **SiteMapPath**.

f. Change the **id** property of the **SiteMapPath** control to **MainSiteMapPath**, make the opening control tag self-closing, and delete the closing tag.

```
<asp:SiteMapPath ID="MainSiteMapPath" runat="server" />
```

2. Add a menu to the master page, by adding a **Menu** control named **MainMenu**, wrapped in a **div** element with a class attribute value of **menu**. Add the new **div** element below the existing **div** element with a **class** attribute value of **siteMapPath**.

a. In the Site.master window, place the cursor after the closing tag of the **div** element, with a **class** attribute value of **siteMapPath**, and then press ENTER.

```
<div class="siteMapPath">
    <asp:SiteMapPath ID="MainSiteMapPath" runat="server" />
</div>
```

b. In the Toolbox, expand **HTML**, and then double-click **div**.

c. Add a **class** attribute to the opening **div** tag with a value of **menu**.

```
<div class="menu">
```

d. In the Site.master window, place the cursor after the opening tag of the **div** element, with a **class** attribute value of **menu**, and then press ENTER.

e. In the Toolbox, expand **Navigation**, and then double-click **Menu**.

f. Change the **id** property of the **Menu** control to **MainMenu**.

```
<asp:Menu ID="MainMenu" runat="server">
</asp:Menu>
```

3. Make the menu layout horizontal, by applying the **Orientation** attribute.

```
Orientation="Horizontal"
```

4. Ensure that the built-in image that indicates if a static menu item has a child menu, is not displayed, by setting the **StaticEnableDefaultPopOutImage** attribute.

```
StaticEnableDefaultPopOutImage="false"
```

5. Get the items for the Menu control from the **MainSiteMapDataSource** data source control, by applying the **DataSourceID** attribute.

```
DataSourceID="MainSiteMapDataSource"
```

6. Add the following child elements to the **Menu** control, by placing them between the opening and closing **Menu** tags.

```
<StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px"
/>
<StaticHoverStyle BackColor="White" ForeColor="Black" />
<DynamicHoverStyle BackColor="White" ForeColor="Black" />
<DynamicMenuItemStyle ItemSpacing="2px" HorizontalPadding="5px"
VerticalPadding="2px" />
```

7. Add a **SiteMapDataSource** control named **MainSiteMapDataSource** to the master page, after the closing tag of the **div** element, with a **class** attribute value of **menu**. The **SiteMapDataSource** control should not show the starting node.

```
<asp:SiteMapDataSource ID="MainSiteMapDataSource" runat="server"
ShowStartingNode="false" />
```

   a. In the Site.master window, place the cursor after the closing tag of the **div** element, with a **class** attribute value of **menu**, and then press ENTER.

   b. In the Toolbox, expand **Data**, and then double-click **SiteMapDataSource**.

   c. Change the **id** property of the **SiteMapDataSource** control to **MainSiteMapDataSource**.

   d. Make sure that the starting node isn't shown, by applying the **ShowStartingNode** attribute.

```
ShowStartingNode="false"
```

8. Format the Site.master master page.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, CTRL+D.

9. Add the **D:\Labfiles\Starter\M5\web.sitemap** site map file to the project.

   - In Solution Explorer, right-click **D:\Labfiles\Starter\M5\CS \CustomerManagement**, and then click **Add Existing Item**.

- In the **Add Existing Item** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M5\web.sitemap**, and then click **Add**.

10. Modify the **div.menu** style, by using the **Manage Styles** window. The menu style must have the following definition:
    - Category: **Position**
    - position: **relative**
    - z-index: **1**
    - Top: **62px**

    a. Open the Manage Styles window, by clicking **Manage Styles** on the **View** menu.

    b. In the **Site.css** list, right-click **div.menu**, and then click **Modify Style**.

    c. In the **Modify Style** dialog box, in the **Category** list, click **Position**, in the **position** box, click **relative**, in the **z-index** box, type **1**, in the **top** box, type **62px**, and then click **OK**.

11. Add a **siteMapPath** style, by using the Manage Styles window.

    a. In the Manage Styles window, click the **New Style** button.

    b. In the New Style window, in the **Selector** box, type **div.siteMapPath**, in the **Define in** list, click **Existing Style Sheet**, and in the **URL** box, click **Styles/Site.css**.

    c. In the **Category** list, click **Position**, in the **position** box, click **fixed**, and in the **top** box, type **42px**.

    d. In the **Category** list, click **Box**, in the **padding** section, clear the **Same for all** check box, in the bottom box, type **5px**, and then click **OK**.

    e. In the Manage Styles window, click the **Close** button.

12. Save all modified files, and run the Web site.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**, or press CTRL+SHIFT+S.

    b. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.

    c. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Convert the Web Form into a user control

1. Open the **InsertCustomer.aspx** Web Form, and change its **Page** directive to a **Control** directive.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="InsertCustomer" %>
```

   a. In Solution Explorer, under D:\Labfiles\Starter\M5\CS \CustomerManagement, right-click **InsertCustomer.aspx**, and then click **View Markup**.

   b. In the InsertCustomer.aspx window, change the following markup,

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="InsertCustomer" %>
```

   to:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="InsertCustomer" %>
```

2. Add a **ClassName** property with a value of **Customer** to the **Control** directive.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="InsertCustomer"
ClassName="Customer" %>
```

   - In the InsertCustomer.aspx window, add a **ClassName** property with a value of **Customer** to the **Control** directive.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="InsertCustomer"
ClassName="Customer" %>
```

3. Change the **Inherits** property value from **InsertCustomer** to **Customer**.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="Customer"
ClassName="Customer" %>
```

- In the InsertCustomer.aspx window, change the value of the **Inherits** property from **InsertCustomer** to **Customer**.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="InsertCustomer.aspx.cs" Inherits="Customer"
ClassName="Customer" %>
```

4. Change the **CodeFile** property value from **InsertCustomer.aspx.cs** to **Customer.aspx.cs**.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Customer.aspx.cs" Inherits="Customer"
ClassName="Customer" %>
```

- In the InsertCustomer.aspx window, change the value of the **CodeFile** property from **InsertCustomer.aspx.cs** to **Customer.aspx.cs**.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Customer.aspx.cs" Inherits="Customer"
ClassName="Customer" %>
```

5. Remove all the top-level HTML elements, such as the **DOCTYPE**, **html**, **head**, **body**, **title**, and **form** elements.
6. Format the document.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, and then press CTRL+D.

Note: After removing all top-level HTML elements, you can view the following markup in the InsertCustomer.aspx window.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Customer.ascx.cs" Inherits="Customer"
    ClassName="Customer" %>
<div class="customerTable">
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerFirstNameLabel" runat="server"
Text="First Name:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerFirstNameTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerLastNameLabel" runat="server"
Text="Last Name:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerLastNameTextBox"
runat="server"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerAddressLabel" runat="server"
Text="Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerAddressTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerZipCodeLabel" runat="server"
Text="Zip Code:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerZipCodeTextBox"
```

(*Code continued on the following page.*)

```
runat="server" MaxLength="10"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCityLabel" runat="server"
Text="City:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerCityTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerStateLabel" runat="server"
Text="State:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerStateTextBox" runat="server"
MaxLength="30"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCountryLabel" runat="server"
Text="Country:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:DropDownList ID="CustomerCountryDropDownList"
runat="server">
            </asp:DropDownList>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerPhoneLabel" runat="server"
Text="Phone:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerPhoneTextBox"
runat="server" MaxLength="30"></asp:TextBox>
        </div>
    </div>
```

(*Code continued on the following page.*)

```
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerEmailAddressLabel"
runat="server" Text="Email Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerEmailAddressTextBox"
runat="server" MaxLength="50"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerWebAddressLabel" runat="server"
Text="Web Address:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerWebAddressTextBox"
runat="server" MaxLength="80"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerCreditLimitLabel"
runat="server" Text="Credit Limit:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:TextBox ID="CustomerCreditLimitTextBox"
runat="server" MaxLength="10"></asp:TextBox>
        </div>
    </div>
    <div class="customerTableRow">
        <div class="customerTableLeftCol">
            <asp:Label ID="CustomerNewsSubscriberLabel"
runat="server" Text="News Subscriber:"></asp:Label>
        </div>
        <div class="customerTableRightCol">
            <asp:CheckBox ID="CustomerNewsSubscriberCheckBox"
runat="server" />
        </div>
    </div>
    <div class="customerTableFooter">
        <asp:Button ID="CustomerInsertButton" runat="server"
Text="Insert" OnClick="CustomerInsertButton_Click" />
         <asp:Button ID="CustomerCancelButton" runat="server"
Text="Cancel" OnClick="CustomerCancelButton_Click" />
    </div>
</div>
```

7. Save the **InsertCustomer.aspx** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, press CTRL+S.
8. Change the Web Form name from **InsertCustomer.aspx** to **Customer.ascx**.
   a. In Solution Explorer, under D:\Labfiles\Starter\M5\CS \CustomerManagement, right-click **InsertCustomer.aspx**, click **Rename**, modify the file name as **Customer.ascx**, and then press ENTER.
   b. In the Microsoft Visual Studio message box, click **Yes**.
9. Open the **Customer.ascx.cs** user control code-behind file, change the class name to Customer, and change its base class from **System.Web.UI.Page** to **System.Web.UI.UserControl**.
   a. In Solution Explorer, under D:\Labfiles\Starter\M5\CS \CustomerManagement, right-click **Customer.ascx**, and then click **View Code**.
   b. In the Customer.ascx.cs code window, change the following markup,

```
public partial class InsertCustomer : System.Web.UI.Page
```

   to:

```
public partial class Customer : System.Web.UI.UserControl
```

10. Move the content from the **Page_LoadComplete** event method and append it to the **Page_Load** event method.

```
protected void Page_Load(object sender, EventArgs e)
{
    // Instantiate Customer
    instantiateCustomerObject();
    // Populate the UI controls
    populateUI();
}
```

   a. In the Customer.ascx.cs code window, in the **Page_LoadComplete** event method, select the following code, right-click, and then click **Cut**.

```
// Populate the UI controls
populateUI();
```

b. In the Customer.ascx.cs code window, paste the code within the **Page_Load** event method.

```
protected void Page_Load(object sender, EventArgs e)
{
    // Instantiate Customer
    instantiateCustomerObject();
    // Populate the UI controls
    populateUI();
}
```

11. Remove the **Page_LoadComplete** event method.

```
/// <summary>
/// Populates UI controls
/// </summary>
/// <param name="sender">>/param>
/// <param name="e"></param>
protected void Page__LoadComplete(object sender, EventArgs e)
{

}
```

- In the Customer.ascx.cs code window, delete the following code to remove the **Page_LoadComplete** event method.

```
/// <summary>
/// Populates UI controls
/// </summary>
/// <param name="sender">>/param>
/// <param name="e"></param>
protected void Page__LoadComplete(object sender, EventArgs e)
{

}
```

12. Save the modified files and close the **Customer.ascx.cs** user control.

a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**, or press CTRL+SHIFT+S.

b. In the Customer.ascx.cs code window, click the **Close** button.

▶ Task 4: Create a content page and insert a user control

1. Add a new content page named **InsertCustomer.aspx**, with a code-behind file, based on the **Site.master** master page.
   a. In Solution Explorer, right-click **D:\Labfiles\Starter\M5\CS \CustomerManagement**, and then click **Add New Item**.
   b. In the **Add New Item - D:\Labfiles\Starter\M5\CS \CustomerManagement** dialog box, in the left pane, ensure that **Visual C#** is selected.
   c. In the middle pane, click **Web Form**, in the **Name** box, type **InsertCustomer.aspx**, ensure that the **Place code in separate file** check box is selected, select the **Select master page** check box, and then click **Add**.
   d. In the **Select a Master Page** dialog box, in the **Contents of folder** box, click **Site.master**, and then click **OK**.
2. Open the **InsertCustomer.aspx** content page in the Design view, and drag the **Customer.ascx** user control to the **MainContentPlaceHolder** control.
   a. In the InsertCustomer.aspx window, click **Design**.
   b. In the InsertCustomer.aspx window, click the **MainContentPlaceHolder** control.
   c. In Solution Explorer, drag the **Customer.ascx** user control to the **MainContentPlaceHolder** control.
   d. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save InsertCustomer.aspx**.
   e. In the InsertCustomer.aspx window, click the **Close** button.
3. Run the **CustomerManagement** Web application.
   a. In Solution Explorer, click **Default.aspx**.
   b. In the CustomerManagement − Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

4. Verify the Contoso Customer Management Web site.
   a. In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.

> **Note:** Notice that the new user control displays on the InsertCustomer Web Form.

   b. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
   c. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 5: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 6

Lab Answer Key: Validating User Input (Visual Basic)

Contents:

Lab: Validating User Input

(Visual Basic)

Exercise 1: Adding Validation Controls

▶ Task 1: Open an existing Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M6\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M6\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Add validation controls to the user control

1. View the markup of the **Customer** user control.
   - In Solution Explorer, under D:\Labfiles\Starter\M6\VB \CustomerManagement, right-click **Customer.ascx** user control, and then click **View Markup**.

2. Add a **RequiredFieldValidator** control named **CustomerFirstNameRequiredFieldValidator**, for the **CustomerFirstNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerFirstNameRequiredFieldValidator"
ControlToValidate="CustomerFirstNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerFirstNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerFirstNameRequiredFieldValidator"
ControlToValidate="CustomerFirstNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

3. Add a **RequiredFieldValidator** control named **CustomerLastNameRequiredFieldValidator**, for the **CustomerLastNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerLastNameRequiredFieldValidator"
ControlToValidate="CustomerLastNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerLastNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerLastNameRequiredFieldValidator"
ControlToValidate="CustomerLastNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

4. Add a **RequiredFieldValidator** control named **CustomerAddressRequiredFieldValidator**, for the **CustomerAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerAddressRequiredFieldValidator"
ControlToValidate="CustomerAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerAddressRequiredFieldValidator"
ControlToValidate="CustomerAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

5. Add a **RequiredFieldValidator** control named **CustomerZipCodeRequiredFieldValidator**, for the **CustomerZipCodeTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerZipCodeRequiredFieldValidator"
ControlToValidate="CustomerZipCodeTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerZipCodeTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerZipCodeRequiredFieldValidator"
ControlToValidate="CustomerZipCodeTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

6. Add a **RequiredFieldValidator** control named **CustomerCityRequiredFieldValidator**, for the **CustomerCityTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCityRequiredFieldValidator"
ControlToValidate="CustomerCityTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerCityTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCityRequiredFieldValidator"
ControlToValidate="CustomerCityTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

7. Add a **RequiredFieldValidator** control named **CustomerCountryRequiredFieldValidator**, for the **CustomerCountryDropDownList** control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerCountryDropDownList** control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

8. Add a **RequiredFieldValidator** control named **CustomerWebAddressRequiredFieldValidator**, for the **CustomerWebAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerWebAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

9. Add a **RequiredFieldValidator** control named **CustomerCreditLimitRequiredFieldValidator**, for the **CustomerCreditLimitTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerCreditLimitTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

10. Add a **RegularExpressionValidator** control named **CustomerEmailAddressRegularExpressionValidator**, for the **CustomerEmailAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpressionV
alidator>
```

- In the Customer.ascx window, type the following markup below the **CustomerEmailAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpress
ionValidator>
```

11. Add a **RegularExpressionValidator** control named **CustomerWebAddressRegularExpressionValidator**, for the **CustomerWebAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpressionV
alidator>
```

- In the Customer.ascx window, type the following markup below the **CustomerWebAddressRequiredFieldValidator** control.

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpress
ionValidator>
```

12. Add a **RangeValidator** control named **CustomerCreditLimitRangeValidator**, for the **CustomerCreditLimitTextBox** control.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000"
ErrorMessage="RangeValidator"></asp:RangeValidator>
```

- In the Customer.ascx window, type the following markup below the **CustomerCreditLimitRequiredFieldValidator** control.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000"
ErrorMessage="RangeValidator"></asp:RangeValidator>
```

13. Add a **ValidationSummary** control named **CustomerValidationSummary**, for the **CustomerInsertButton** control.

```
<asp:ValidationSummary ID="CustomerValidationSummary"
runat="server"></asp:ValidationSummary>
```

- In the Customer.ascx window, type the following markup above the **CustomerInsertButton** control.

```
<asp:ValidationSummary ID="CustomerValidationSummary"
runat="server"></asp:ValidationSummary>
```

14. Format the **Customer.ascx** user control.

- In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press the CTRL+K keys, and then press the CTRL+D keys.

15. Save the **Customer** user control, and view the changes in the browser.

a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**.

b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

Note: Notice that a default value of **0** is added to the **Credit Limit** TextBox. This value is retrieved from the **Customer** class.

16. Test the functionality of the **Customer** user control.
    a. In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.
    b. In the Contoso Customer Management – Windows Internet Explorer window, click the **Refresh (F5)** button.
    c. In the Contoso Customer Management – Windows Internet Explorer window, click the **Cancel** button.

> **Note:** Notice that there is no difference to the rendered Web page, whether clicking the **Insert** or **Cancel** button.

17. Close Windows® Internet Explorer®.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Configuring Validation Controls

▶ Task 1: Remove validation from the Cancel button

1. View the default **Response.Redirect** method of the **Cancel** button.

```
''' <summary>
''' Redirects to home page
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerCancelButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CustomerCancelButton.Click
    ' Redirect to home page
    Response.Redirect("~/Default.aspx")
End Sub
```

   a. In Solution Explorer, right-click **Customer.ascx**, and then click **View Code**.

   b. In the Customer.ascx.vb code window, locate the **CustomerCancelButton_Click** event method, and notice that this method is redirected to the Default.aspx Web Form on the CustomerManagement Web site:

```
''' <summary>
''' Redirects to home page
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerCancelButton_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
CustomerCancelButton.Click
    ' Redirect to home page
    Response.Redirect("~/Default.aspx")
End Sub
```

2. Disable the validation caused by the **CustomerCancelButton** control, by setting the **CausesValidation** property in the user control to **false**.

```
<asp:Button ID="CustomerCancelButton" runat="server" Text="Cancel"
OnClick="CustomerCancelButton_Click" CausesValidation="false" />
```

   a. In the CustomerManagement – Microsoft Visual Studio window, click **Customer.ascx**.

b. In the Customer.ascx window, locate the **CustomerCancelButton** control,

```
<asp:Button ID="CustomerCancelButton" runat="server"
Text="Cancel" OnClick="CustomerCancelButton_Click" />
```

and then set the **CausesValidation** property to **false**.

```
CausesValidation="false"
```

3. Save the **Customer** user control and view the changes in the browser.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**.
   b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.
   c. In the Contoso Customer Management – Windows Internet Explorer window, click the **Cancel** button.

Note: Notice that the Web browser is redirected to the default Web Form, instead of displaying the error messages.

4. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 2: Add error indicators and error messages to the validation controls

1. Add a **Text** property with the value of * to the **CustomerFirstNameRequiredFieldValidator** control.
   - In the Customer.ascx window, locate the **CustomerFirstNameRequiredFieldValidator** control,

```
<asp:RequiredFieldValidator
ID="CustomerFirstNameRequiredFieldValidator"
ControlToValidate="CustomerFirstNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

and then add a **Text** property with the value of *.

```
Text="*"
```

2. Change the **ErrorMessage** property in the **CustomerFirstNameRequiredFieldValidator** control to **The Customer First Name must be filled in**.

   - In the Customer.ascx window, in the **CustomerFirstNameRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

   ```
   ErrorMessage="The Customer First Name must be filled in."
   ```

3. Add a **Text** property with the value of * to the **CustomerLastNameRequiredFieldValidator** control.

   - In the Customer.ascx window, locate the **CustomerLastNameRequiredFieldValidator** control,

   ```
   <asp:RequiredFieldValidator
   ID="CustomerLastNameRequiredFieldValidator"
   ControlToValidate="CustomerLastNameTextBox" runat="server"
   ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
   ator>
   ```

   and then add a **Text** property with the value of *.

   ```
   Text="*"
   ```

4. Change the **ErrorMessage** property in the **CustomerLastNameRequiredFieldValidator** control to **The Customer Last Name must be filled in**.

   - In the Customer.ascx window, in the **CustomerLastNameRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

   ```
   ErrorMessage="The Customer Last Name must be filled in."
   ```

5. Add a **Text** property with the value of * to the **CustomerAddressRequiredFieldValidator** control.

   - In the Customer.ascx window, locate the **CustomerAddressRequiredFieldValidator** control,

     ```
     <asp:RequiredFieldValidator
     ID="CustomerAddressRequiredFieldValidator"
     ControlToValidate="CustomerAddressTextBox" runat="server"
     ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
     ator>
     ```

     and then add a **Text** property with the value of *.

     ```
     Text="*"
     ```

6. Change the **ErrorMessage** property in the **CustomerAddressRequiredFieldValidator** control to **The Address must be filled in**.

   - In the Customer.ascx window, in the **CustomerAddressRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

     ```
     ErrorMessage="The Address must be filled in."
     ```

7. Add a **Text** property with the value of * to the **CustomerZipCodeRequiredFieldValidator** control.

   - In the Customer.ascx window, locate the **CustomerZipCodeRequiredFieldValidator** control,

     ```
     <asp:RequiredFieldValidator
     ID="CustomerZipCodeRequiredFieldValidator"
     ControlToValidate="CustomerZipCodeTextBox" runat="server"
     ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
     ator>
     ```

     and then add a **Text** property with the value of *.

     ```
     Text="*"
     ```

8. Change the **ErrorMessage** property in the **CustomerZipCodeRequiredFieldValidator** control to **The Zip Code must be filled in**.

   - In the Customer.ascx window, in the **CustomerZipCodeRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

     ```
     ErrorMessage="The Zip Code must be filled in."
     ```

9. Add a **Text** property with the value of * to the **CustomerCityRequiredFieldValidator** control.

   - In the Customer.ascx window, locate the **CustomerCityRequiredFieldValidator** control,

     ```
     <asp:RequiredFieldValidator
     ID="CustomerCityRequiredFieldValidator"
     ControlToValidate="CustomerCityTextBox" runat="server"
     ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
     ator>
     ```

     and then add a **Text** property with the value of *.

     ```
     Text="*"
     ```

10. Change the **ErrorMessage** property in the **CustomerCityRequiredFieldValidator** control to **The City must be filled in**.

    - In the Customer.ascx window, in the **CustomerCityRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The City must be filled in."
      ```

11. Add a **Text** property with the value of * to the **CustomerCountryRequiredFieldValidator** control.

- In the Customer.ascx window, locate the **CustomerCountryRequiredFieldValidator** control,

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

  and then add a **Text** property with the value of *.

```
Text="*"
```

12. Change the **ErrorMessage** property in the **CustomerCountryRequiredFieldValidator** control to **A country must be selected**.

    - In the Customer.ascx window, in the **CustomerCountryRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

```
ErrorMessage="A country must be selected."
```

13. Add a **Text** property with the value of * to the **CustomerEmailAddressRegularExpressionValidator** control.

    - In the Customer.ascx window, locate the **CustomerEmailAddressRegularExpressionValidator** control,

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RequiredFieldV
alidator>
```

  and then add a **Text** property with the value of *.

```
Text="*"
```

14. Change the **ErrorMessage** property in the **CustomerEmailAddressRegularExpressionValidator** control to **The Email Address must be valid**.

    - In the Customer.ascx window, in the **CustomerEmailAddressRegularExpressionValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The Email Address must be valid."
      ```

15. Add a **Text** property with the value of * to the **CustomerWebAddressRequiredFieldValidator** control.

    - In the Customer.ascx window, locate the **CustomerWebAddressRequiredFieldValidator** control,

      ```
      <asp:RequiredFieldValidator
      ID="CustomerWebAddressRequiredFieldValidator"
      ControlToValidate="CustomerWebAddressTextBox" runat="server"
      ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
      ator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

16. Change the **ErrorMessage** property in the **CustomerWebAddressRequiredFieldValidator** control to **The Web Address must be filled in**.

    - In the Customer.ascx window, in the **CustomerWebAddressRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The Web Address must be filled in."
      ```

17. Add a **Text** property with the value of * to the **CustomerWebAddressRegularExpressionValidator** control.

    - In the Customer.ascx window, locate the **CustomerWebAddressRegularExpressionValidator** control,

      ```
      <asp:RegularExpressionValidator
      ID="CustomerWebAddressRegularExpressionValidator"
      ControlToValidate="CustomerWebAddressTextBox" runat="server"
      ErrorMessage="RegularExpressionValidator"></asp:RequiredFieldV
      alidator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

18. Change the **ErrorMessage** property in the **CustomerWebAddressRegularExpressionValidator** control to **The Web Address must be valid**.

    - In the Customer.ascx window, in the **CustomerWebAddressRegularExpressionValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The Web Address must be valid."
      ```

19. Add a **Text** property with the value of * to the **CustomerCreditLimitRequiredFieldValidator** control.

    - In the Customer.ascx window, locate the **CustomerCreditLimitRequiredFieldValidator** control,

      ```
      <asp:RequiredFieldValidator
      ID="CustomerCreditLimitRequiredFieldValidator"
      ControlToValidate="CustomerCreditLimitTextBox" runat="server"
      ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
      ator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

20. Change the **ErrorMessage** property in the **CustomerCreditLimitRequiredFieldValidator** control to **The Credit Limit must be filled in**.

    - In the Customer.ascx window, in the **CustomerCreditLimitRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The Credit Limit must be filled in."
      ```

21. Add a **Text** property with the value of * to the **CustomerCreditLimitRangeValidator** control.

    - In the Customer.ascx window, locate the **CustomerCreditLimitRangeValidator** control,

      ```
      <asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
      ControlToValidate="CustomerCreditLimitTextBox" runat="server"
      MinimumValue="500" MaximumValue="50000"
      ErrorMessage="RangeValidator"></asp:RangeValidator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

22. Change the **ErrorMessage** property in the **CustomerCreditLimitRangeValidator** control to **The Credit Limit must be within the valid range**.

    - In the Customer.ascx window, in the **CustomerCreditLimitRangeValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The Credit Limit must be within the valid
      range."
      ```

23. Save the **Customer** user control, and view the changes in the browser.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**.

    b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

c. In the Contoso Customer Management – Windows Internet Explorer window, in the **Email Address** box, type **contoso.com**, and then press the TAB key.

d. In the Contoso Customer Management – Windows Internet Explorer window, in the **Web Address** box, type **www.contoso**, and then press the TAB key.

Note: If an AutoComplete message box displays, click the **No** button.

Note: Notice the error indicator that displays next to the **Email Address** and **Web Address** boxes, because of the invalid e-mail and web addresses.

e. In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

Note: Notice the error indicator and error messages next to the **First Name**, **Last Name**, **Address**, **Zip Code**, **City**, **Country**, and **Credit Limit** controls.

24. Close Windows® Internet Explorer®.

- In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Set the e-mail address, Web address, and credit limit validation controls

1. View the InsertCustomer.aspx Web Form in a browser.

a. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

b. In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

Note: Notice that the value for **Credit Limit** box is set to **0**, and the error indicator text for the **Web Address** box is not aligned with the other error indictors. Also notice that the error message for the **Credit Limit** box is **The Credit Limit must be within the valid range**.

2. Close Windows® Internet Explorer®.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
3. Add a **Display** property with the value of **Dynamic** to the **CustomerWebAddressRequiredFieldValidator** control to change the display of the error indicator text.

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="The Web Address must be filled in." Text="*"
Display="Dynamic"></asp:RequiredFieldValidator>
```

   - In the Customer.ascx window, locate the **CustomerWebAddressRequiredFieldValidator** control,

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="The Web Address must be filled in."
Text="*"></asp:RequiredFieldValidator>
```

     and then set the **Display** attribute to **Dynamic**.

```
Display="Dynamic"
```

4. Add a **Display** property with the value of **Dynamic** to the **CustomerCreditLimitRequiredFieldValidator** control to change the display of the error indicator text.

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="The Credit Limit must be filled in." Text="*"
Display="Dynamic"></asp:RequiredFieldValidator>
```

- In the Customer.ascx window, locate the **CustomerCreditLimitRequiredFieldValidator** control,

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="The Credit Limit must be filled in."
Text="*"></asp:RequiredFieldValidator>
```

  and then set the **Display** attribute to **Dynamic**.

```
Display="Dynamic"
```

5. Save the **Customer** user control, and view the changes in the browser.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**.
   b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.
6. Click the **Insert** button.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

> **Note:** Notice that the location of the error indicator for the **Credit Limit** box has changed. However, the error message for the **Credit Limit** box is still **The Credit Limit must be within the valid range**.

7. Close Windows® Internet Explorer®.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

8. Add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*** to the **CustomerEmailAddressRegularExpressionValidator** control.

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="The Email Address must be valid." Text="*"
ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*"></asp:RegularExpressionValidator>
```

   - In the Customer.ascx window, locate the **CustomerEmailAddressRegularExpressionValidator** control,

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="The Email Address must be valid."
Text="*"></asp:RegularExpressionValidator>
```

     and then, add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)***.

```
ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*"
```

9. Add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*** to the **CustomerWebAddressRegularExpressionValidator** control.

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="The Web Address must be valid." Text="*"
ValidationExpression="^http(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-
Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&amp;%\$#_]*)?$"
```

- In the Customer.ascx window, locate the **CustomerWebAddressRegularExpressionValidator** control,

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="The Web Address must be valid."
Text="*"></asp:RegularExpressionValidator>
```

and then, add a **ValidationExpression** property with the value of **^http(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&%\$#_]*)?$**.

```
ValidationExpression="^http(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-
zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-
\.\?\,\'\/\\\+&amp;%\$#_]*)?$"
```

10. Set the **Type** property with the value of **Integer** to the **CustomerCreditLimitRangeValidator** control.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000" ErrorMessage="The Credit
Limit must be within the valid range." Text="*"
Type="Integer"></asp:RangeValidator>
```

- In the Customer.ascx window, locate the **CustomerCreditLimitRangeValidator** control,

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000" ErrorMessage="The
Credit Limit must be within the valid range."
Text="*"></asp:RangeValidator>
```

and set the **Type** property with the value of **Integer**.

```
Type="Integer"
```

11. Change the **MinimumValue** property of the **CustomerCreditLimitRangeValidator** control to **0**.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="0" MaximumValue="50000" ErrorMessage="The Credit
Limit must be within the valid range." Text="*"
Type="Integer"></asp:RangeValidator>
```

- In the Customer.ascx window, locate the **CustomerCreditLimitRangeValidator** control,

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000" ErrorMessage="The
Credit Limit must be within the valid range." Text="*"
Type="Integer"></asp:RangeValidator>
```

  and change the **MinimumValue** property to **0**.

```
MinimumValue="0"
```

12. Save the **Customer** user control, and view the changes in the browser.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press CTRL+S.

    b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

    c. In the Contoso Customer Management – Windows Internet Explorer window, in the **Email Address** box, type **claus@contoso.com**, in the **Web Address** box, type **http://www.contoso.com**, and then click the **Insert** button.

> **Note**: Notice that the error indicator and error messages do not display for the **Email Address**, **Web Address**, and **Credit Limit** boxes.

    d. In the Contoso **Customer** Management – Windows Internet Explorer window, click the **Close** button.

Exercise 3: Adding Server-Side Validation

▶ Task 1 : Validate the Customer User Control

1. Open the Customer.ascx.vb code window.
   - In Solution Explorer, right-click **Customer.ascx**, and then click **View Code**.
2. Add the code to validate the user control within the **CustomerInsertButton_Click** event handler method.

```
''' <summary>
''' Saves the current customer information and adds default values
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerInsertButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CustomerInsertButton.Click
    ' Did page validation succeed?
    If Not Page.IsValid Then
        Return
    End If

    ' Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name
    ' Add the user credit limit
    currentCustomer.CreditLimit = 50000
End Sub
```

- In the Customer.ascx.vb window, add the following code,

```
' Did page validation succeed?
If Not Page.IsValid Then
    Return
End If
```

below the code:

```
''' <summary>
''' Saves the current customer information and adds default
values
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerInsertButton_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
CustomerInsertButton.Click
```

3. Add postback validation to the **Page_Load** event handler method.

```
''' <summary>
''' Instantiates Customer object
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    If Page.IsPostBack Then
        ' Validate Page
        Page.Validate()

        ' Did page validation succeed?
        If Not Page.IsValid Then
            Return
        End If
    End If

    ' Instantiate Customer
    instantiateCustomerObject()
    ' Populate the UI controls
    populateUI()
End Sub
```

- In the Customer.ascx.vb window, add the following code,

```
If Page.IsPostBack Then
    ' Validate Page
    Page.Validate()

    ' Did page validation succeed?
    If Not Page.IsValid Then
        Return
    End If
End If
```

below the code:

```
''' <summary>
''' Instantiates Customer object
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
```

4. Disable the client-side validation for the **CustomerCountryDropDownList** control by setting the **EnableClientScript** property of the **CustomerCountryRequiredFieldValidator** control to **false**.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected." Text="*"
EnableClientScript="false"></asp:RequiredFieldValidator>
```

   a. In the CustomerManagement – Microsoft Visual Studio window, click **Customer.ascx**.

   b. In the Customer.ascx window, set the **EnableClientScript** property of the **CustomerCountryRequiredFieldValidator** control to **false**.

```
EnableClientScript="false"
```

5. Save the changes and view the changes in the browser.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**, or press CTRL+SHIFT+S.
   b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.
6. In the Contoso Customer Management – Windows Internet Explorer, type the following settings, and then click the **Insert** button.
   - First Name: **Claus**
   - Last Name: **Hansen**
   - Address: **4567 Main St.**
   - Zip Code: **98052**
   - City: **Buffalo**
   - State: **NY**
   - Web Address: **http://www.cohowinery.com**

> **Note:** Notice the postback of the Web page with the inputs. Also notice that after the postback, the error indicator for the **Country** list and its associated error message display.

7. Close Windows Internet Explorer.
   - In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.
8. Remove the **EnableClientScript** property for the **CustomerCountryRequiredFieldValidator** control to enable the client-side validation for the **CustomerCountryDropDownList** control, and format and save the **Customer** user control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected."
Text="*"></asp:RequiredFieldValidator>
```

a. In the Customer.ascx window, remove the **EnableClientScript** property of the **CustomerCountryRequiredFieldValidator** control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected."
Text="*"></asp:RequiredFieldValidator>
```

b. In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, and then press CTRL+D.

c. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press CTRL+S.

9. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 2: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 6

Lab Answer Key: Validating User Input (Visual C#)

Contents:

Lab: Validating User Input

(C#)

Exercise 1: Adding Validation Controls

▶ Task 1: Open an existing Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M6\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M6\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Add validation controls to the user control

1. View the markup of the **Customer** user control.
   - In Solution Explorer, under D:\Labfiles\Starter\M6\CS \CustomerManagement, right-click **Customer.ascx** user control, and then click **View Markup**.

2. Add a **RequiredFieldValidator** control named **CustomerFirstNameRequiredFieldValidator**, for the **CustomerFirstNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerFirstNameRequiredFieldValidator"
ControlToValidate="CustomerFirstNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerFirstNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerFirstNameRequiredFieldValidator"
ControlToValidate="CustomerFirstNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

3. Add a **RequiredFieldValidator** control named **CustomerLastNameRequiredFieldValidator**, for the **CustomerLastNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerLastNameRequiredFieldValidator"
ControlToValidate="CustomerLastNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerLastNameTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerLastNameRequiredFieldValidator"
ControlToValidate="CustomerLastNameTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

4. Add a **RequiredFieldValidator** control named **CustomerAddressRequiredFieldValidator**, for the **CustomerAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerAddressRequiredFieldValidator"
ControlToValidate="CustomerAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerAddressRequiredFieldValidator"
ControlToValidate="CustomerAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

5. Add a **RequiredFieldValidator** control named **CustomerZipCodeRequiredFieldValidator**, for the **CustomerZipCodeTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerZipCodeRequiredFieldValidator"
ControlToValidate="CustomerZipCodeTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerZipCodeTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerZipCodeRequiredFieldValidator"
ControlToValidate="CustomerZipCodeTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

6. Add a **RequiredFieldValidator** control named **CustomerCityRequiredFieldValidator**, for the **CustomerCityTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCityRequiredFieldValidator"
ControlToValidate="CustomerCityTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerCityTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCityRequiredFieldValidator"
ControlToValidate="CustomerCityTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

7. Add a **RequiredFieldValidator** control named **CustomerCountryRequiredFieldValidator**, for the **CustomerCountryDropDownList** control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the .ascx window, type the following markup below the **CustomerCountryDropDownList** control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

8. Add a **RequiredFieldValidator** control named **CustomerWebAddressRequiredFieldValidator**, for the **CustomerWebAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerWebAddressTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

9. Add a **RequiredFieldValidator** control named **CustomerCreditLimitRequiredFieldValidator**, for the **CustomerCreditLimitTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator
>
```

   - In the Customer.ascx window, type the following markup below the **CustomerCreditLimitTextBox** control.

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
ator>
```

10. Add a **RegularExpressionValidator** control named **CustomerEmailAddressRegularExpressionValidator**, for the **CustomerEmailAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpressionV
alidator>
```

- In the Customer.ascx window, type the following markup below the **CustomerEmailAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpress
ionValidator>
```

11. Add a **RegularExpressionValidator** control named **CustomerWebAddressRegularExpressionValidator**, for the **CustomerWebAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpressionV
alidator>
```

- In the Customer.ascx window, type the following markup below the **CustomerWebAddressTextBox** control.

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="RegularExpressionValidator"></asp:RegularExpress
ionValidator>
```

12. Add a **RangeValidator** control named **CustomerCreditLimitRangeValidator**, for the **CustomerCreditLimitTextBox** control. The minimum value is **500** and the maximum value is **50000**.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000"
ErrorMessage="RangeValidator"></asp:RangeValidator>
```

- In the Customer.ascx window, type the following markup below the **CustomerCreditLimitRequiredFieldValidator** control.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000"
ErrorMessage="RangeValidator"></asp:RangeValidator>
```

13. Add a **ValidationSummary** control named **CustomerValidationSummary**, for the **CustomerInsertButton** control.

```
<asp:ValidationSummary ID="CustomerValidationSummary"
runat="server"></asp:ValidationSummary>
```

- In the Customer.ascx window, type the following markup above the **CustomerInsertButton** control.

```
<asp:ValidationSummary ID="CustomerValidationSummary"
runat="server"></asp:ValidationSummary>
```

14. Format the **Customer.ascx** user control.

- In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press the CTRL+K keys, and then press the CTRL+D keys.

15. Save the **Customer** user control, and view the changes in the browser.
    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**.
    b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

> **Note:** Notice that a default value of **0** is added to the **Credit Limit** TextBox. This value is retrieved from the **Customer** class.

16. Test the functionality of the **Customer** user control.
    a. In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.
    b. In the Contoso Customer Management – Windows Internet Explorer window, click the **Refresh (F5)** button.
    c. In the Contoso Customer Management – Windows Internet Explorer window, click the **Cancel** button.

> **Note:** Notice that there is no difference to the rendered Web page, whether clicking the **Insert** or **Cancel** button.

17. Close Windows® Internet Explorer®.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Configuring Validation Controls

▶ Task 1: Remove validation from the Cancel button

1. View the default **Response.Redirect** method of the **Cancel** button.

```
/// <summary>
/// Redirects to home page
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerCancelButton_Click(object sender, EventArgs
e)
{
    // Redirect to home page
    Response.Redirect("~/Default.aspx");
}
```

- In Solution Explorer, right-click **Customer.ascx**, and then click **View Code**.
- In the Customer.ascx.cs code window, locate the **CustomerCancelButton_Click** event method and notice that this method is redirected to the Default.aspx Web Form on the CustomerManagement Web site:

```
/// <summary>
/// Redirects to home page
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerCancelButton_Click(object sender,
EventArgs e)
{
    // Redirect to home page
    Response.Redirect("~/Default.aspx");
}
```

2. Disable the validation caused by the **CustomerCancelButton** control by setting the **CausesValidation** property in the user control to **false**.

```
<asp:Button ID="CustomerCancelButton" runat="server" Text="Cancel"
OnClick="CustomerCancelButton_Click" CausesValidation="false" />
```

   a. In the CustomerManagement – Microsoft Visual Studio window, click **Customer.ascx**.

   b. In the Customer.ascx window, locate the **CustomerCancelButton** control,

```
<asp:Button ID="CustomerCancelButton" runat="server"
Text="Cancel" OnClick="CustomerCancelButton_Click" />
```

   and then set the **CausesValidation** property to **false**.

```
CausesValidation="false"
```

3. Save the **Customer** user control and view the changes in the browser.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**.

   b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

   c. In the Contoso Customer Management – Windows Internet Explorer window, click the **Cancel** button.

> **Note:** Notice that the Web browser is redirected to the default Web Form, instead of displaying the error messages.

   d. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 2: Add error indicators and error messages to the validation controls

1. Add a **Text** property with the value of * to the **CustomerFirstNameRequiredFieldValidator** control.

   - In the Customer.ascx window, locate the **CustomerFirstNameRequiredFieldValidator** control,

     ```
     <asp:RequiredFieldValidator
     ID="CustomerFirstNameRequiredFieldValidator"
     ControlToValidate="CustomerFirstNameTextBox" runat="server"
     ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
     ator>
     ```

     and then add a **Text** property with the value of *.

     ```
     Text="*"
     ```

2. Change the **ErrorMessage** property in the **CustomerFirstNameRequiredFieldValidator** control to **The Customer First Name must be filled in**.

   - In the Customer.ascx window, in the **CustomerFirstNameRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

     ```
     ErrorMessage="The Customer First Name must be filled in."
     ```

3. Add a **Text** property with the value of * to the **CustomerLastNameRequiredFieldValidator** control.

   - In the Customer.ascx window, locate the **CustomerLastNameRequiredFieldValidator** control,

     ```
     <asp:RequiredFieldValidator
     ID="CustomerLastNameRequiredFieldValidator"
     ControlToValidate="CustomerLastNameTextBox" runat="server"
     ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
     ator>
     ```

     and then add a **Text** property with the value of *.

     ```
     Text="*"
     ```

4. Change the **ErrorMessage** property in the **CustomerLastNameRequiredFieldValidator** control to **The Customer Last Name must be filled in**.

   - In the Customer.ascx window, in the **CustomerLastNameRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

     ```
     ErrorMessage="The Customer Last Name must be filled in."
     ```

5. Add a **Text** property with the value of * to the **CustomerAddressRequiredFieldValidator** control.

   - In the Customer.ascx window, locate the **CustomerAddressRequiredFieldValidator** control,

     ```
     <asp:RequiredFieldValidator
     ID="CustomerAddressRequiredFieldValidator"
     ControlToValidate="CustomerAddressTextBox" runat="server"
     ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
     ator>
     ```

     and then add a **Text** property with the value of *.

     ```
     Text="*"
     ```

6. Change the **ErrorMessage** property in the **CustomerAddressRequiredFieldValidator** control to **The Address must be filled in**.

   - In the Customer.ascx window, in the **CustomerAddressRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

     ```
     ErrorMessage="The Address must be filled in."
     ```

7. Add a **Text** property with the value of * to the **CustomerZipCodeRequiredFieldValidator** control.

   - In the Customer.ascx window, locate the **CustomerZipCodeRequiredFieldValidator** control,

     ```
     <asp:RequiredFieldValidator
     ID="CustomerZipCodeRequiredFieldValidator"
     ControlToValidate="CustomerZipCodeTextBox" runat="server"
     ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
     ator>
     ```

     and then add a **Text** property with the value of *.

     ```
     Text="*"
     ```

8. Change the **ErrorMessage** property in the **CustomerZipCodeRequiredFieldValidator** control to **The Zip Code must be filled in**.

   - In the Customer.ascx window, in the **CustomerZipCodeRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

     ```
     ErrorMessage="The Zip Code must be filled in."
     ```

9. Add a **Text** property with the value of * to the **CustomerCityRequiredFieldValidator** control.

   - In the Customer.ascx window, locate the **CustomerCityRequiredFieldValidator** control,

     ```
     <asp:RequiredFieldValidator
     ID="CustomerCityRequiredFieldValidator"
     ControlToValidate="CustomerCityTextBox" runat="server"
     ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
     ator>
     ```

     and then add a **Text** property with the value of *.

     ```
     Text="*"
     ```

10. Change the **ErrorMessage** property in the **CustomerCityRequiredFieldValidator** control to **The City must be filled in**.

    - In the Customer.ascx window, in the **CustomerCityRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The City must be filled in."
      ```

11. Add a **Text** property with the value of * to the **CustomerCountryRequiredFieldValidator** control.

    - In the Customer.ascx window, locate the **CustomerCountryRequiredFieldValidator** control,

      ```
      <asp:RequiredFieldValidator
      ID="CustomerCountryRequiredFieldValidator"
      ControlToValidate="CustomerCountryDropDownList" runat="server"
      ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
      ator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

12. Change the **ErrorMessage** property in the **CustomerCountryRequiredFieldValidator** control to **A country must be selected**.

    - In the Customer.ascx window, in the **CustomerCountryRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="A country must be selected."
      ```

13. Add a **Text** property with the value of * to the **CustomerEmailAddressRegularExpressionValidator** control.

    - In the Customer.ascx window, locate the **CustomerEmailAddressRegularExpressionValidator** control,

      ```
      <asp:RegularExpressionValidator
      ID="CustomerEmailAddressRegularExpressionValidator"
      ControlToValidate="CustomerEmailAddressTextBox" runat="server"
      ErrorMessage="RegularExpressionValidator"></asp:RequiredFieldV
      alidator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

14. Change the **ErrorMessage** property in the **CustomerEmailAddressRegularExpressionValidator** control to **The Email Address must be valid**.

    - In the Customer.ascx window, in the **CustomerEmailAddressRegularExpressionValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The Email Address must be valid."
      ```

15. Add a **Text** property with the value of * to the **CustomerWebAddressRequiredFieldValidator** control.

    - In the Customer.ascx window, locate the **CustomerWebAddressRequiredFieldValidator** control,

      ```
      <asp:RequiredFieldValidator
      ID="CustomerWebAddressRequiredFieldValidator"
      ControlToValidate="CustomerWebAddressTextBox" runat="server"
      ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
      ator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

16. Change the **ErrorMessage** property in the **CustomerWebAddressRequiredFieldValidator** control to **The Web Address must be filled in**.

    - In the Customer.ascx window, in the **CustomerWebAddressRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The Web Address must be filled in."
      ```

17. Add a **Text** property with the value of * to the **CustomerWebAddressRegularExpressionValidator** control.

    - In the Customer.ascx window, locate the **CustomerWebAddressRegularExpressionValidator** control,

      ```
      <asp:RegularExpressionValidator
      ID="CustomerWebAddressRegularExpressionValidator"
      ControlToValidate="CustomerWebAddressTextBox" runat="server"
      ErrorMessage="RegularExpressionValidator"></asp:RequiredFieldV
      alidator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

18. Change the **ErrorMessage** property in the **CustomerWebAddressRegularExpressionValidator** control to **The Web Address must be valid**.

    - In the Customer.ascx window, in the **CustomerWebAddressRegularExpressionValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The Web Address must be valid."
      ```

19. Add a **Text** property with the value of * to the **CustomerCreditLimitRequiredFieldValidator** control.

    - In the Customer.ascx window, locate the **CustomerCreditLimitRequiredFieldValidator** control,

      ```
      <asp:RequiredFieldValidator
      ID="CustomerCreditLimitRequiredFieldValidator"
      ControlToValidate="CustomerCreditLimitTextBox" runat="server"
      ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValid
      ator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

20. Change the **ErrorMessage** property in the **CustomerCreditLimitRequiredFieldValidator** control to **The Credit Limit must be filled in**.

    - In the Customer.ascx window, in the **CustomerCreditLimitRequiredFieldValidator** control, change the **ErrorMessage** property to the following example.

      ```
      ErrorMessage="The Credit Limit must be filled in."
      ```

21. Add a **Text** property with the value of * to the **CustomerCreditLimitRangeValidator** control.

    - In the Customer.ascx window, locate the **CustomerCreditLimitRangeValidator** control,

      ```
      <asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
      ControlToValidate="CustomerCreditLimitTextBox" runat="server"
      MinimumValue="500" MaximumValue="50000"
      ErrorMessage="RangeValidator"></asp:RangeValidator>
      ```

      and then add a **Text** property with the value of *.

      ```
      Text="*"
      ```

22. Change the **ErrorMessage** property in the **CustomerCreditLimitRangeValidator** control to **The Credit Limit must be within the valid range**.

    - In the Customer.ascx window, in the **CustomerCreditLimitRangeValidator** control, change the **ErrorMessage** property to the following example.

```
ErrorMessage="The Credit Limit must be within the valid
range."
```

23. Save the **Customer** user control and view the changes in the browser.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**.

    b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

    c. In the Contoso Customer Management – Windows Internet Explorer window, in the **Email Address** box, type **contoso.com**, and then press the TAB key.

    d. In the Contoso Customer Management – Windows Internet Explorer window, in the **Web Address** box, type **www.contoso**, and then press the TAB key.

> **Note:** If an AutoComplete message displays, click the **No** button.

> **Note:** Notice the error indicator that displays next to the **Email Address** and **Web Address** boxes because of the invalid e-mail and web addresses.

    e. In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

> **Note:** Notice the error indicator and error messages next to the **First Name**, **Last Name**, **Address**, **Zip Code**, **City**, **Country**, and **Credit Limit** controls.

24. Close Windows Internet Explorer.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Set the e-mail address, Web address, and credit limit validation controls

1. View the InsertCustomer.aspx Web Form in a browser.
    a. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.
    b. In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

> **Note:** Notice that the value for **Credit Limit** box is set to **0**, and the error indicator text for the **Web Address** box is not aligned with the other error indictors. Also notice that the error message for the **Credit Limit** box is **The Credit Limit must be within the valid range**.

2. Close Windows Internet Explorer.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
3. Add a **Display** property with the value of **Dynamic** to the **CustomerWebAddressRequiredFieldValidator** control to change the display of the error indicator text.

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="The Web Address must be filled in." Text="*"
Display="Dynamic"></asp:RequiredFieldValidator>
```

- In the Customer.ascx window, locate the **CustomerWebAddressRequiredFieldValidator** control,

```
<asp:RequiredFieldValidator
ID="CustomerWebAddressRequiredFieldValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="The Web Address must be filled in."
Text="*"></asp:RequiredFieldValidator>
```

  and then set the **Display** attribute to **Dynamic**.

```
Display="Dynamic"
```

4. Add a **Display** property with the value of **Dynamic** to the **CustomerCreditLimitRequiredFieldValidator** control to change the display of the error indicator text.

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="The Credit Limit must be filled in." Text="*"
Display="Dynamic"></asp:RequiredFieldValidator>
```

- In the Customer.ascx window, locate the **CustomerCreditLimitRequiredFieldValidator** control,

```
<asp:RequiredFieldValidator
ID="CustomerCreditLimitRequiredFieldValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
ErrorMessage="The Credit Limit must be filled in."
Text="*"></asp:RequiredFieldValidator>
```

  and then set the **Display** attribute to **Dynamic**.

```
Display="Dynamic"
```

5. Save the **Customer** user control and view the changes in the browser.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**.

   b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

6. Click the **Insert** button.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

> **Note:** Notice that the location of the error indicator for the **Credit Limit** box has changed. However, the error message for the **Credit Limit** box is still **The Credit Limit must be within the valid range**.

7. Close Windows Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
8. Add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*** to the **CustomerEmailAddressRegularExpressionValidator** control.

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="The Email Address must be valid." Text="*"
ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*"></asp:RegularExpressionValidator>
```

   - In the Customer.ascx window, locate the **CustomerEmailAddressRegularExpressionValidator** control,

```
<asp:RegularExpressionValidator
ID="CustomerEmailAddressRegularExpressionValidator"
ControlToValidate="CustomerEmailAddressTextBox" runat="server"
ErrorMessage="The Email Address must be valid."
Text="*"></asp:RegularExpressionValidator>
```

     and then, add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)***.

```
ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*"
```

9. Add a **ValidationExpression** property with the value of **\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*** to the **CustomerWebAddressRegularExpressionValidator** control.

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="The Web Address must be valid." Text="*"
ValidationExpression="^http(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-
Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&amp;%\$#_]*)?$"
```

   - In the Customer.ascx window, locate the **CustomerWebAddressRegularExpressionValidator** control,

```
<asp:RegularExpressionValidator
ID="CustomerWebAddressRegularExpressionValidator"
ControlToValidate="CustomerWebAddressTextBox" runat="server"
ErrorMessage="The Web Address must be valid."
Text="*"></asp:RegularExpressionValidator>
```

     and then, add a **ValidationExpression** property with the value of **^http(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&%\$#_]*)?$**.

```
ValidationExpression="^http(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-
zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-
\.\?\,\'\/\\\+&amp;%\$#_]*)?$"
```

10. Set the **Type** property with the value of **Integer** to the **CustomerCreditLimitRangeValidator** control.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000" ErrorMessage="The Credit
Limit must be within the valid range." Text="*"
Type="Integer"></asp:RangeValidator>
```

- In the Customer.ascx window, locate the **CustomerCreditLimitRangeValidator** control,

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000" ErrorMessage="The
Credit Limit must be within the valid range."
Text="*"></asp:RangeValidator>
```

  and set the **Type** property with the value of **Integer**.

```
Type="Integer"
```

11. Change the **MinimumValue** property of the **CustomerCreditLimitRangeValidator** control to **0**.

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="0" MaximumValue="50000" ErrorMessage="The Credit
Limit must be within the valid range." Text="*"
Type="Integer"></asp:RangeValidator>
```

- In the Customer.ascx window, locate the **CustomerCreditLimitRangeValidator** control,

```
<asp:RangeValidator ID="CustomerCreditLimitRangeValidator"
ControlToValidate="CustomerCreditLimitTextBox" runat="server"
MinimumValue="500" MaximumValue="50000" ErrorMessage="The
Credit Limit must be within the valid range." Text="*"
Type="Integer"></asp:RangeValidator>
```

  and change the **MinimumValue** property to **0**.

```
MinimumValue="0"
```

12. Save the **Customer** user control and view the changes in the browser.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press CTRL+S.

    b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

    c. In the Contoso Customer Management – Windows Internet Explorer window, in the **Email Address** box, type **claus@contoso.com**, in the **Web Address** box, type **http://www.contoso.com**, and then click the **Insert** button.

> **Note:** Notice that the error indicator and error messages do not display for the **Email Address**, **Web Address**, and **Credit Limit** boxes.

    d. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 3: Adding Server-Side Validation

▶ Task1: Validate the Customer User Control

1. Open the Customer.ascx.cs code window.
   - In Solution Explorer, right-click **Customer.ascx**, and then click **View Code**.
2. Add the code to validate the user control within the **CustomerInsertButton_Click** event handler method.

```
/// <summary>
/// Saves the current customer information and adds default values
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerInsertButton_Click(object sender, EventArgs
e)
{
    // Did page validation succeed?
    if (!Page.IsValid)
        return;

    // Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name;
    // Add the user credit limit
    currentCustomer.CreditLimit = 50000;
}
```

   - In the Customer.ascx.cs window, add the following code,

```
// Did page validation succeed?
if (!Page.IsValid)
    return;
```

   below the code:

```
/// <summary>
/// Saves the current customer information and adds default
values
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void CustomerInsertButton_Click(object sender,
EventArgs e)
{
```

3. Add postback validation to the **Page_Load** event handler method.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.IsPostBack)
    {
        // Validate Page
        Page.Validate();

        // Did page validation succeed?
        if (!Page.IsValid)
            return;
    }

    // Instantiate Customer
    instantiateCustomerObject();
    // Populate the UI controls
    populateUI();
}
```

   - In the Customer.ascx.cs window, add the following code,

```
if (Page.IsPostBack)
{
    // Validate Page
    Page.Validate();

    // Did page validation succeed?
    if (!Page.IsValid)
        return;
}
```

     below the code:

```
protected void Page_Load(object sender, EventArgs e)
{
```

4. Disable the client-side validation for the **CustomerCountryDropDownList** control, by setting the **EnableClientScript** property of the **CustomerCountryRequiredFieldValidator** control to **false**.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected." Text="*"
EnableClientScript="false"></asp:RequiredFieldValidator>
```

a. In the CustomerManagement – Microsoft Visual Studio window, click **Customer.ascx**.

b. In the Customer.ascx window, set the **EnableClientScript** property of the **CustomerCountryRequiredFieldValidator** control to **false**.

```
EnableClientScript="false"
```

5. Save the changes and view the changes in the browser.

a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**, or press CTRL+SHIFT+S.

b. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **View in Browser**.

6. In the Contoso Customer Management – Windows Internet Explorer, type the following settings, and then click the **Insert** button.

- First Name: **Claus**
- Last Name: **Hansen**
- Address: **4567 Main St.**
- Zip Code: **98052**
- City: **Buffalo**
- State: **NY**
- Web Address: **http://www.cohowinery.com**

Note: Notice the postback of the Web page with the inputs. Also notice that after the postback, the error indicator for the **Country** list and its associated error message displays.

7. Close Windows Internet Explorer.

- In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

8. Remove the **EnableClientScript** property for the **CustomerCountryRequiredFieldValidator** control to enable the client-side validation for the **CustomerCountryDropDownList** control, and format and save the **Customer** user control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected."
Text="*"></asp:RequiredFieldValidator>
```

   a. In the Customer.ascx window, remove the **EnableClientScript** property of the **CustomerCountryRequiredFieldValidator** control.

```
<asp:RequiredFieldValidator
ID="CustomerCountryRequiredFieldValidator"
ControlToValidate="CustomerCountryDropDownList" runat="server"
ErrorMessage="A country must be selected."
Text="*"></asp:RequiredFieldValidator>
```

   b. In the CustomerManagement − Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, and then press CTRL+D.

   c. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press CTRL+S.

9. In the CustomerManagement − Microsoft Visual Studio window, click the **Close** button.

▶ Task 2: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 7

Lab Answer Key: Troubleshooting Microsoft® ASP.NET Web Applications (Visual Basic)

Contents:

Lab: Troubleshooting Microsoft® ASP.NET Web Applications

(Visual Basic)

Exercise 1: Debugging a Web Application

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M7\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M7\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Enable debugging of the CustomerManagement Web project

1. Open the **web.config** file of the **CustomerManagement** Web project.
   - In Solution Explorer, under D:\Labfiles\Starter\M7\VB\CustomerManagement, right-click **web.config**, and then click **Open**.
2. Set the **debug** attribute of the **compilation** element to **true**.

```
<compilation debug="true" strict="false" explicit="true"
targetFramework="4.0" />
```

   - In the web.config window, change the value of the **debug** attribute of the **compilation** element to **true**.

```
<compilation debug="true" strict="false" explicit="true"
targetFramework="4.0" />
```

3. Save and close the **web.config** file.

   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save web.config**, or press the CTRL+S keys.

   b. In the web.config window, click the **Close** button, or press CTRL+F4.

▶ Task 3: Add debug output statements to the user control

1. Open the **Customer** user control in Code view.

   - In Solution Explorer, under D:\Labfiles\Starter\M7\VB \CustomerManagement, right-click **Customer.ascx**, and then click **View Code**.

2. Import the **System.Diagnostics** namespace to the user control.

```
Imports System.Diagnostics
```

   - In the Customer.ascx.vb code window, add the **System.Diagnostics** namespace at the top of all the coding.

```
Imports System.Diagnostics
```

3. In the **Page_Load** event handler, send the message, "Page Postback detected in Page_Load" to the trace listeners, when the page loads in response to a postback.

```
Debug.WriteLine("Page Postback detected in Page_Load")
```

   - In the Customer.ascx.vb code window, add the following code,

```
Debug.WriteLine("Page Postback detected in Page_Load")
```

     below the first **If** statement declaration, as defined in the following statement.

```
If Page.IsPostBack Then
```

4. In the **Page_Load** event handler, send the message, "No Page Postback detected in Page_Load" to the trace listeners, when the page is not loaded in response to a postback.

```
Debug.WriteLineIf(Not Page.IsPostBack, "No Page Postback detected
in Page_Load")
```

   - In the Customer.ascx.vb code window, add the following code,

```
Debug.WriteLineIf(Not Page.IsPostBack, "No Page Postback
detected in Page_Load")
```

     below the end of the first **If** statement, as demonstrated in the following statement.

```
If Page.IsPostBack Then
    Debug.WriteLine("Page Postback detected in Page_Load")

    ' Validate Page
    Page.Validate()

    ' Did page validation succeed?
    If Not Page.IsValid Then
        Return
    End If
End If
```

5. At the end of the **Page_Unload** event handler, send the message, "Page has been unloaded", to the trace listeners.

```
Debug.WriteLine("Page has been unloaded")
```

   - In the Customer.ascx.vb code window, send the message, "Page has been unloaded" to the trace listeners, by appending the following code to the **Page_Unload** event handler.

```
Debug.WriteLine("Page has been unloaded")
```

6. At the end of the **Click** event of the **CustomerInsertButton** control, send the message, "Customer has been inserted in CustomerInsertButton_Click" to the trace listeners.

```
Debug.WriteLine("Customer has been inserted in
CustomerInsertButton_Click")
```

   - In the Customer.ascx.vb code window, send the message, "Customer has been inserted in CustomerInsertButton_Click" to the trace listeners, by appending the following code to the **Click** event of the **CustomerInsertButton** control.

```
Debug.WriteLine("Customer has been inserted in
CustomerInsertButton_Click")
```

7. At the end of the private **populateUI** method, send the message, "UI controls have been populated" to the trace listeners.

```
Debug.WriteLine("UI controls have been populated")
```

   - In the Customer.ascx.vb code window, send the message, "UI controls have been populated" to the trace listeners, by appending the following code to the private **populateUI** method.

```
Debug.WriteLine("UI controls have been populated")
```

8. At the end of the private **instantiateCustomerObject** method, send the message, "Customer object has been instantiated" to the trace listeners.

```
Debug.WriteLine("Customer object has been instantiated")
```

   - In the Customer.ascx.vb code window, send the message, "Customer object has been instantiated" to the trace listeners, by appending the following code to the private **instantiateCustomerObject** method.

```
Debug.WriteLine("Customer object has been instantiated")
```

9. Save the user control code file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx.vb**, or press CTRL+S.

10. Add a default item to the **CountryDropDownList** control.
    a. In the Customer.ascx.vb window, right-click and then click **View Designer**.
    b. In the Customer.ascx window, click the **CustomerCountryDropDownList** control.
    c. Click the **Smart Tag** button, and then click **Edit Items**.
    d. In the **ListItem Collection Editor** dialog box, click **Add**.
    e. In the ListItem properties pane, in the **Text** box, type **USA**, and then click **OK**.
11. Save and close the user control file.
    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press CTRL+S.
    b. In the Customer.ascx window, click the **Close** button, or press CTRL+F4.
12. Run the Web application in the debug mode.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**, or press F5.
13. Verify the output for the **Page_Load**, **Page_Unload**, and **instantiateCustomerObject** methods.
    a. In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
    b. In the CustomerManagement (Running) – Microsoft Visual Studio window, in the Debugger variable windows, click **Output**.

> **Note:** In the **Debug** pane of the Output window, notice that the output of the **Debug** statements display. You may have to scroll up to see the statements.

14. Verify the output for the **Click** event handler of the **CustomerInsertButton** control by entering the following information, and then clicking the **Insert** button:

- In the Contoso Customer Management – Windows Internet Explorer window, enter the following information, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**

15. Close Windows® Internet Explorer®.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

> **Note:** In the **Debug** pane of the Output window, notice that the **Customer has been inserted in CustomerInsertButton_Click** message is displayed.

▶ Task 4: Find and fix a bug

1. Run the **CustomerManagement** Web application to test its functionality.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.
2. Create a new customer, by using the following information:
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
     - First Name: **Claus**
     - Last Name: **Hansen**
     - Address: **4567 Main St.**
     - Zip Code: **98052**
     - City: **Buffalo**
     - State: **NY**
     - Web Address: **http://www.cohowinery.com**

Note: Notice that the value for the **Credit Limit** box is set to **0** by default.

3. Click the **Insert** button.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

Note: Notice that the value for the **Credit Limit** box is set to **50**, which is incorrect.

4. Close Windows Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
5. Add a breakpoint in the **Page_Load** event handler in the line of code that calls the **instantiateCustomerObject** method.

```
instantiateCustomerObject()
```

   - In the Customer.ascx.vb code window, in the **Page_Load** event handler, place the cursor in the line of code that calls the **instantiateCustomerObject** method, and then on the **Debug** menu, click **Toggle Breakpoint**, or press F9.

```
instantiateCustomerObject()
```

6. Run the Web application in the debug mode.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**.
   b. In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers menu**, click **New**.
7. Step into the **instantiateCustomerObject** method
   - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Into**, or press F11.
8. Step over the first line of code and then check for postback in the **instantiateCustomerObject** method.

a. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.

b. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.

9. Add a watch to the **Text** property of the **CustomerCreditLimitTextBox** control.

   - In the **instantiateCustomerObject** method, in the following code,

     ```
     CustomerWebAddressTextBox.Text,
     Integer.Parse(CustomerCreditLimitTextBox.Text),
     ```

     select the text, **CustomerCreditLimitTextBox.Text**, right-click the selection, and then click **Add Watch**.

10. Continue to debug the Web application.

    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Continue**, or press F5.

11. Create a new customer, by using the following information, and then click the **Insert** button.

    - In the Contoso Customer Management – Windows Internet Explorer window enter the following information, and then click the **Insert** button:
      - First Name: **Claus**
      - Last Name: **Hansen**
      - Address: **4567 Main St.**
      - Zip Code: **98052**
      - City: **Buffalo**
      - State: **NY**
      - Web Address: **http://www.cohowinery.com**

Note: Notice that in the Watch 1 window, the value of the **CustomerCreditLimitTextBox** Text property is set to **0**.

12. Step over the call to the **instantiateCustomerObject** method.
    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.
13. Step over the call to the **populateUI** method.
    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.

> **Note:** In the Watch1 window, notice that the value for the **CustomerCreditLimitTextBox** is changed to **50**.

14. Stop debugging the Web project.
    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Stop Debugging**, or press SHIFT+F5.
15. Examine the code in the **populateUI** method that assigns a value to the **CustomerCreditLimitTextBox.Text** property.
    - In the Customer.ascx.vb code window, locate the code that assigns a value to the **CustomerCreditLimitTextBox.Text** property.

```
CustomerCreditLimitTextBox.Text =
currentCustomer.CreditLimit.ToString()
```

> **Note:** Notice that the value of 50 for the CreditLimit property is not assigned here.

16. Re-run the Web application in the debug mode to examine the **Customer** class.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**.
17. Create a new customer, and ignore the first breakpoint, by continuing program execution.
    a. In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers menu**, click **New**.

b. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Continue**.

18. Create a new customer, by using the following information, and then click the **Insert** button:

- In the Contoso Customer Management – Windows Internet Explorer window, enter the following information, and then click the **Insert** button:
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**

19. Step into the **instantiateCustomerObject** method.

- In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Into**, or press F11.

20. Step over the first line of code, and then the check for the postback.

a. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.

b. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.

c. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.

21. Step into the method that instantiates the customer object.

- In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Into**, or press F11.

> **Note:** Stepping into the instantiation of the customer object action may take some time.

22. Step over each line of code in the Customer.vb code window, until the following line of code in the constructor that initializes the **CreditLimit** property is reached.

```
Me.CreditLimit = creditLimit
```

- In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**. Repeat this step, until the debugger highlights the following code.

```
Me.CreditLimit = creditLimit
```

23. Step through the assignment of the passed value to the **CreditLimit** property.

- In the Customer.vb code window, right-click the highlighted line of code, point to **Step Into Specific**, and then click **CustomerManagementEntities.Customer.set_CreditLimit**.

24. Locate the step that adds the extra **50** to the private **customerCreditLimit** member **Credit Limit** box.

```
Me.customerCreditLimit = value + 50
```

- In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**. Repeat this step until the debugger highlights the following code.

```
Me.customerCreditLimit = value + 50
```

25. View the functions and procedure calls that are currently on the stack, by viewing the Call Stack window.

a. On the **Debug** menu, select **Windows**, and then click **Call Stack**, or press CTRL+ALT+C.

b. In the Call Stack window, notice that four methods are listed in the order in which they were called, in descending order.

26. Stop debugging the Web project.

- In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click. **Stop Debugging**, or press SHIFT+F5.

27. Remove the extra value from the **customerCreditLimit** property.

```
Me.customerCreditLimit = value
```

- In the Customer.vb code window, from the **customerCreditLimit** property, remove the assignment of the extra **50**.

```
Me.customerCreditLimit = value
```

28. Save and close the class file.

a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.vb**, or press CTRL+S.

b. In the Customer.vb window, click the **Close** button, or press CTRL+F4.

29. Run the Web application to verify the **Credit Limit** value.

- In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.

30. Create a new customer, by using the following information:

- In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.First Name: Claus
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**

31. Click the **Insert** button.

- In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

Note: Notice that the value for the **Credit Limit** box is still **0**.

32. Close Windows Internet Explorer.

- In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Tracing a Web Application

▶ Task 1: Enable application-level tracing of the CustomerManagement Web project

1. Add a trace element to the **web.config** file as the first element within the **system.web** element, and set the value of the **enabled** attribute to **true**.

```
<trace enabled="true" />
```

   a. In Solution Explorer, right-click the **web.config** file, and then click **Open**.

   b. In the web.config window, add a self-closing trace element with the **enabled** attribute value set to **true**,

```
<trace enabled="true" />
```

   within the **system.web** element:

```
<system.web>
```

2. Save and close the **web.config** file.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**, or press CTRL+S.

   b. In the web.config window, click the **Close** button, or press CTRL+F4.

▶ Task 2: Implement application tracing

1. Run the Web application, and view the trace details.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

2. Create a new customer, by using the following information, and then click the **Insert** button:

   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
     - First Name: **Claus**
     - Last Name: **Hansen**
     - Address: **4567 Main St.**

- Zip Code: **98052**
- City: **Buffalo**
- State: **NY**
- Web Address: **http://www.cohowinery.com**

3. View the trace details for the application, by using the **http://localhost:1111/CustomerManagement/trace.axd** URL in the browser.
   - In the Address bar of the Contoso Customer Management – Windows Internet Explorer window, type **http://localhost:1111 /CustomerManagement/trace.axd**, and then press ENTER.
4. View the details for the **InsertCustomer.aspx** Web page.
   - In the http://localhost:1111/CustomerManagement/trace.axd - Windows Internet Explorer window, click **View Details** of **/InsertCustomer.aspx** corresponding to the **Verb**, **GET**.

> **Note:** Scroll down to see all of the information output, including the control tree, session and application state, form and querystring collection, and server variables of the **InsertCustomer.aspx** Web page. Verify that you can see the Trace Information section, which gives you information about how much time is spent when loading, rendering, and unloading the Web Form.

5. Close Internet Explorer.
   a. In the http://localhost:1111/CustomerManagement/trace.axd?=2 – Windows Internet Explorer window, click the **Close** button.
   b. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 3: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 7

Lab Answer Key: Troubleshooting Microsoft® ASP.NET Web Applications (Visual C#)

Contents:

Lab: Troubleshooting Microsoft® ASP.NET Web Applications

(C#)

Exercise 1: Debugging a Web Application

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M7\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M7\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Enable debugging of the CustomerManagement Web project

1. Open the **web.config** file of the **CustomerManagement** Web project.
   - In Solution Explorer, under D:\Labfiles\Starter\M7\CS\CustomerManagement, right-click **web.config**, and then click **Open**.
2. Set the **debug** attribute of the **compilation** element to **true**.

```
<compilation debug="true" targetFramework="4.0">
```

- In the web.config window, change the value of the **debug** attribute of the **compilation** element to **true**.

```
<compilation debug="true" targetFramework="4.0">
```

3. Save and close the **web.config** file.
   a. In the **CustomerManagement – Microsoft Visual Studio** window, on the **File** menu, click **Save web.config**, or press the CTRL+S keys.
   b. In the web.config window, click the **Close** button, or press CTRL+F4.

▶ Task 3: Add debug output statements to the user control

1. Open the **Customer** user control in Code view.
   - In Solution Explorer, under D:\Labfiles\Starter\M7\CS\CustomerManagement, right-click **Customer.ascx**, and then click **View Code**.
2. Import the **System.Diagnostics** namespace to the user control.

```
using System.Diagnostics;
```

   - In the Customer.ascx.cs code window, add the **System.Diagnostics** namespace at the top of all the coding.

```
using System.Diagnostics;
```

3. In the **Page_Load** event handler, send the message, "Page Postback detected in Page_Load" to the trace listeners, when the page is loaded in response to a postback.

```
Debug.WriteLine("Page Postback detected in Page_Load");
```

   - In the Customer.ascx.cs code window, add the following code,

```
Debug.WriteLine("Page Postback detected in Page_Load");
```

   below the first **if** statement declaration, as defined in the following statement.

```
if (Page.IsPostBack)
{
```

4. In the **Page_Load** event handler, send the message, "No Page Postback detected in Page_Load" to the trace listeners, when the page is not loaded in response to a postback.

```
Debug.WriteLineIf(!Page.IsPostBack, "No Page Postback detected in
Page_Load");
```

   - In the Customer.ascx.cs code window, add the following code,

```
Debug.WriteLineIf(!Page.IsPostBack, "No Page Postback detected
in Page_Load");
```

     below the end of the first **if** statement, as defined in the following statement.

```
if (Page.IsPostBack)
{
    Debug.WriteLine("Page Postback detected in Page_Load");

    // Validate Page
    Page.Validate();

    // Did page validation succeed?
    if (!Page.IsValid)
        return;
}
```

5. At the end of the **Page_Unload** event handler, send the message, "Page has been unloaded" to the trace listeners.

```
Debug.WriteLine("Page has been unloaded");
```

   - In the Customer.ascx.cs code window, send the message, "Page has been unloaded" to the trace listeners, by appending the following code to the **Page_Unload** event handler.

```
Debug.WriteLine("Page has been unloaded");
```

6. At the end of the **Click** event of the **CustomerInsertButton** control, send the message, "Customer has been inserted in CustomerInsertButton_Click" to the trace listeners.

```
Debug.WriteLine("Customer has been inserted in
CustomerInsertButton_Click");
```

   - In the Customer.ascx.cs code window, send the message, "Customer has been inserted in CustomerInsertButton_Click" to the trace listeners, by appending the following code to the **Click** event of the **CustomerInsertButton** control.

```
Debug.WriteLine("Customer has been inserted in
CustomerInsertButton_Click");
```

7. At the end of the private **populateUI** method, send the message, "UI controls have been populated" to the trace listeners.

```
Debug.WriteLine("UI controls have been populated");
```

   - In the Customer.ascx.cs code window, send the message, "UI controls have been populated" to the trace listeners, by appending the following code to the private **populateUI** method.

```
Debug.WriteLine("UI controls have been populated");
```

8. At the end of the private **instantiateCustomerObject** method, send the message, "Customer object has been instantiated" to the trace listeners.

```
Debug.WriteLine("Customer object has been instantiated");
```

   - In the Customer.ascx.cs code window, send the message, "Customer object has been instantiated" to the trace listeners, by appending the following code to the private **instantiateCustomerObject** method.

```
Debug.WriteLine("Customer object has been instantiated");
```

9. Save the user control code file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx.cs**, or press CTRL+S.

10. Add a default item to the **CountryDropDownList** control.
    a. In the Customer.ascx.cs window, right-click and then click **View Designer**.
    b. In the Customer.ascx window, click the **CustomerCountryDropDownList** control.
    c. Click the **Smart Tag** button, and then click **Edit Items**.
    d. In the **ListItem Collection Editor** dialog box, click **Add**.
    e. In the ListItem properties pane, in the **Text** box, type **USA**, and then click **OK**.
11. Save and close the user control file.
    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press CTRL+S.
    b. In the Customer.ascx window, click the **Close** button, or press CTRL+F4.
12. Run the Web application in the debug mode.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**, or press F5.
13. Verify the output for the Page_Load, Page_Unload, and **instantiateCustomerObject** methods.
    a. In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
    b. In the CustomerManagement (Running) – Microsoft Visual Studio window, in the Debugger variable windows, click **Output**.

> **Note:** In the **Debug** pane of the Output window, notice that the output of the **Debug** statements display. You may have to scroll up to see the statements.

14. Verify the output for the **Click** event handler of the **CustomerInsertButton** control by creating a new customer using the following information, and then Click the **Insert** button.
    - In the Contoso Customer Management – Windows Internet Explorer window, enter the following information, and then click the **Insert** button:
      - First Name: **Claus**
      - Last Name: **Hansen**

- Address: **4567 Main St.**
- Zip Code: **98052**
- City: **Buffalo**
- State: **NY**
- Web Address: **http://www.cohowinery.com**

15. Close Windows Internet Explorer®.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

> **Note:** In the **Debug** pane of the Output window, notice that the **Customer has been inserted in CustomerInsertButton_Click** message is displayed.

▶ Task 4: Find and fix a bug

1. Run the **CustomerManagement** Web application to test its functionality.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.
2. Create a new customer, by using the following information:
    - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
        - First Name: **Claus**
        - Last Name: **Hansen**
        - Address: **4567 Main St.**
        - Zip Code: **98052**
        - City: **Buffalo**
        - State: **NY**
        - Web Address: **http://www.cohowinery.com**

> **Note:** Notice that the value for the **Credit Limit** box is set to **0** by default.

3. Click the **Insert** button.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

> **Note:** Notice that the value for the **Credit Limit** box is set to **50**, which is incorrect.

4. Close Windows Internet Explorer®.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
5. Add a breakpoint in the **Page_Load** event handler in the line of code that calls the **instantiateCustomerObject** method.

```
instantiateCustomerObject();
```

   - In the Customer.ascx.cs code window, in the **Page_Load** event handler, place the cursor in the line of code that calls the **instantiateCustomerObject** method, and then on the **Debug** menu, click **Toggle Breakpoint**, or press F9.

```
instantiateCustomerObject();
```

6. Run the Web application in the debug mode.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**.
   b. In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
7. Step into the **instantiateCustomerObject** method.
   - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Into**, or press F11.
8. Step over the first line of code and then check for postback in the **instantiateCustomerObject** method.
   a. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.
   b. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.

9. Add a watch to the **Text** property of the **CustomerCreditLimitTextBox** control.
   - In the **instantiateCustomerObject** method, in the following code,

     ```
     CustomerWebAddressTextBox.Text,
     int.Parse(CustomerCreditLimitTextBox.Text),
     ```

     select the text, **CustomerCreditLimitTextBox.Text**, right-click the selection, and then click **Add Watch**.
10. Continue to debug the Web application.
    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Continue**, or press F5.
11. Create a new customer, by using the following information, and then click the **Insert** button.
    - In the Contoso Customer Management – Windows Internet Explorer window, enter the following information, and then click the **Insert** button:
      - First Name: **Claus**
      - Last Name: **Hansen**
      - Address: **4567 Main St.**
      - Zip Code: **98052**
      - City: **Buffalo**
      - State: **NY**
      - Web Address: **http://www.cohowinery.com**

> **Note:** Notice that in the Watch 1 window, the value of the **CustomerCreditLimitTextBox** Text property is set to **0**.

12. Step over the call to the **instantiateCustomerObject** method.
    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.

13. Step over the call to the **populateUI** method.
    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.

> **Note:** In the Watch1 window, notice that the value for the **CustomerCreditLimitTextBox** is changed to **50**.

14. Stop debugging the Web project.
    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Stop Debugging**, or press SHIFT+F5.
15. Examine the code in the **populateUI** method that assigns a value to the **CustomerCreditLimitTextBox.Text** property.
    - In the Customer.ascx.cs code window, locate the code that assigns a value to the **CustomerCreditLimitTextBox.Text** property.

```
CustomerCreditLimitTextBox.Text =
currentCustomer.CreditLimit.ToString();
```

> **Note:** Notice that the value of 50 for the CreditLimit property is not assigned here.

16. Re-run the Web application in the debug mode to examine the **Customer** class.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**.
17. Create a new customer, and ignore the first breakpoint, by continuing program execution.
    a. In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
    b. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Continue**.

18. Create a new customer, by using the following information, and then click the **Insert** button:
    - First Name: **Claus**
    - Last Name: **Hansen**
    - Address: **4567 Main St.**
    - Zip Code: **98052**
    - City: **Buffalo**
    - State: **NY**
    - Web Address: **http://www.cohowinery.com**
19. Step into the **instantiateCustomerObject** method.
    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Into**, or press F11.
20. Step over the first line of code and then the check for the postback.
    a. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.
    b. In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**, or press F10.
21. Step into the method that instantiates the customer object.
    - In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Into**, or press F11.

> **Note:** Stepping into the instantiation of the customer object action may take some time.

22. Step over each line of code in the Customer.cs code window, until the following line of code in the constructor that initializes the **CreditLimit** property is reached.

```
this.CreditLimit = creditLimit;
```

- In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**. Repeat this step, until the debugger highlights the following code.

```
this.CreditLimit = creditLimit;
```

23. Step through the assignment of the passed value to the **CreditLimit** property.

- In the Customer.cs code window, right-click the highlighted line of code, point to **Step Into Specific**, and then click **CustomerManagementEntities.Customer.set_CreditLimit**.

24. Locate the step that adds the extra **50** to the private **customerCreditLimit** member **Credit Limit** box.

```
this.customerCreditLimit = value + 50;
```

- In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click **Step Over**. Repeat this step until the debugger highlights the following code.

```
this.customerCreditLimit = value + 50;
```

25. View the functions and procedure calls that are currently on the stack, by viewing the Call Stack window.

- On the **Debug** menu, select **Windows** and then click **Call Stack**, or press CTRL+ALT+C.
- In the Call Stack window, notice that four methods are listed in the order in which they were called, in descending order.

26. Stop debugging the Web project.

- In the CustomerManagement (Debugging) – Microsoft Visual Studio window, on the **Debug** menu, click. **Stop Debugging**, or press SHIFT+F5.

27. Remove the extra value from the **customerCreditLimit** property.

```
this.customerCreditLimit = value;
```

- In the Customer.cs code window, from the **customerCreditLimit** property, remove the assignment of the extra **50**.

```
this.customerCreditLimit = value;
```

28. Save and close the class file.

- In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.cs**, or press CTRL+S.
- In the Customer.cs window, click the **Close** button, or press CTRL+F4.

29. Run the Web application to verify the **Credit Limit** value.

- In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.

30. Create a new customer, by using the following information:

- In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
  - First Name: **Claus**
  - Last Name: **Hansen**
  - Address: **4567 Main St.**
  - Zip Code: **98052**
  - City: **Buffalo**
  - State: **NY**
  - Web Address: **http://www.cohowinery.com**

31. Click the **Insert** button.

    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Insert** button.

> **Note:** Notice that the value for the **Credit Limit** box is still **0**.

32. Close Windows Internet Explorer.

    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Tracing a Web Application

▶ Task 1: Enable application-level tracing of the CustomerManagement Web project

1. Add a trace element to the **web.config** file as the first element within the **system.web** element, and set the value of the **enabled** attribute to **true**.

```
<trace enabled="true" />
```

   - In Solution Explorer, right-click the **web.config** file, and then click **Open**.
   - In the web.config window, add a self-closing trace element with the **enabled** attribute value set to **true**,

```
<trace enabled="true" />
```

     within the **system.web** element:

```
<system.web>
```

2. Save and close the **web.config** file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**, or press CTRL+S.
   - In the web.config window, click the **Close** button, or press CTRL+F4.

▶ Task 2: Implement application tracing

1. Run the Web application, and view the trace details.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.
2. Create a new customer, by using the following information, and then click the **Insert** button:
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
     - First Name: **Claus**
     - Last Name: **Hansen**
     - Address: **4567 Main St.**

- Zip Code: **98052**
- City: **Buffalo**
- State: **NY**
- Web Address: **http://www.cohowinery.com**

3. View the trace details for the application, by using the **http://localhost:1110/CustomerManagement/trace.axd** URL in the browser.
   - In the Address bar of the Contoso Customer Management – Windows Internet Explorer window, type **http://localhost:1110 /CustomerManagement/trace.axd**, and then press ENTER.
4. View the details for the **InsertCustomer.aspx** Web page.
   - In the http://localhost:1110/CustomerManagement/trace.axd - Windows Internet Explorer window, click **View Details** of **/InsertCustomer.aspx** corresponding to the **Verb**, **GET**.

> **Note:** Scroll down to see all of the information output, including the control tree, session and application state, form and querystring collection, and server variables of the **InsertCustomer.aspx** Web page. Verify that you can see the Trace Information section, which gives you information about how much time is spent when loading, rendering, and unloading the Web Form.

5. Close Internet Explorer.
   a. In the http://localhost:1110/CustomerManagement /trace.axd?=2 – Windows Internet Explorer window, click the **Close** button.
   b. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 3: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 8

Lab Answer Key: Managing Data in a Microsoft® ASP.NET 4.0 Web Application (Visual Basic)

Contents:

Lab: Managing Data in an ASP.NET 4 Web Application

(Visual Basic)

Exercise 1: Connecting to a Data Source

▶ Task 1: Open an existing ASP.NET Web project

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M8\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M8\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Add a SQL Server 2008 Express Database

- Add a new database folder named **App_Data**, and an existing database to the database folder, **D:\LabFiles\Starter\M8\CustomerManagement.mdf**.
  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M8\VB \CustomerManagement**, point to **Add ASP.NET Folder**, and then click **App_Data**.
  b. In Solution Explorer, right-click **App_Data**, and then click **Add Existing Item**.
  c. In the **Add Existing Item** dialog box, in the **File name** box, type **D:\LabFiles\Starter\M8\CustomerManagement.mdf**, and then click **Add**.

▶ Task 3: Add a data source control to the user control

1. Open the **Customer** user control in Design view.
   - In Solution Explorer, under D:\Labfiles\Starter\M8\VB \CustomerManagement, right-click **Customer.ascx**, and then click **View Designer**.
2. Add a **SqlDataSource** control to the user control to connect to an SQL Server database.
   a. In the Customer.ascx window, place the cursor below the **Insert** and **Cancel** buttons, and then point to **Toolbox**.
   b. In the **Toolbox**, expand **Data**, and then double-click the **SqlDataSource** control.
3. Rename the **SqlDataSource** control as **CountriesSqlDataSource**.
   - With the **SqlDataSource** control selected, in the Properties window, change the value of the **ID** property from **SqlDataSource1** to **CountriesSqlDataSource**.
4. Save the **Customer** user control.
   - In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press the CTRL+S keys.

▶ Task 4: Configure a data source control

1. In Design view, with the **CountriesSqlDataSource** control selected, display the Smart Tag.
   - In the Customer.ascx window, click the button with the arrow on the right side of the control, or right-click the **CountriesSqlDataSource** control, and then click **Show Smart Tag**.
2. Open the Configure Data Source Wizard, connect to the **CustomerManagement.mdf** database, and then create a new connection string named **CustomerManagementConnectionString**.
   a. In the Customer.ascx window, on the **Smart Tag**, click **Configure Data Source**.

b. On the **Choose Your Data Connection** page of the Configure Data Source - CountriesSqlDataSource Wizard, in the **Which data connection should your application use to connect to the database?** list, click **CustomerManagement.mdf**, and then click **Next**.

c. On the **Save the Connection String to the Application Configuration File** page of the Configure Data Source - CountriesSqlDataSource Wizard, in the **Yes, save this connection as** box, type **CustomerManagementConnectionString**, and then click **Next**.

3. Configure the **SELECT** statement to include the **ID** and **Name** columns from the **Countries** table.

a. On the **Configure the Select Statement** page, ensure that the **Specify columns from a table or view** option is selected, and ensure that **Countries** is selected in the **Name** list.

b. In the **Columns** list, select the **ID** and **Name** check boxes, and then click **Next**.

4. Test the query, and check whether you get the correct data from the Countries table.

a. On the **Test Query** page, click **Test Query**.

> **Note:** Ensure that the returned rows include the values for the ID and Name columns for various countries.

b. On the **Test Query** page, click **Finish**.

5. Save the **Customer** user control.

- In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press the CTRL+S keys.

Exercise 2: Binding a Server Control to a Data Source

▶ Task 1: Bind the DropDownList control to a data source

1. Open the **Customer** user control in Source view.
   - In the Customer.ascx window, click **Source**.
2. Locate the markup for the **CustomerCountryDropDownList** control.
3. Remove the static **ListItem** element from the **CustomerCountryDropDownList** control.

```
<asp:ListItem>USA</asp:ListItem>
```

4. Bind the **CustomerCountryDropDownList** control to the **CountriesSqlDataSource** control by using the **DataSourceID** attribute.
   - In Customer.ascx window, append the opening **CustomerCountryDropDownList** tag with the following markup.

```
DataSourceID="CountriesSqlDataSource"
```

5. Set the value field of the **CustomerCountryDropDownList** control to the **ID** column of the database by using the **DataValueField** attribute.
   - In the Customer.ascx window, append the opening **CustomerCountryDropDownList** tag with the following markup.

```
DataValueField="ID"
```

6. Set the text field of the **CustomerCountryDropDownList** control to the **Name** column of the database by using the **DataTextField** attribute.
   - In the Customer.ascx window, append the opening **CustomerCountryDropDownList** tag with the following markup.

```
DataTextField="Name"
```

7. Build the user control, and fix any errors.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.

> **Note:** Notice the Build succeeded message that displays in the **Build** pane of the Output window.

▶ Task 2: Pass the values to the Customer object

1. Open the **Customer** user control in the Code view.
   - In Solution Explorer, right-click **Customer.ascx**, and then click **View Code**.
2. Locate the code for the private **instantiateCustomerObject** method, and pass the selected value of the **CustomerCountryDropDownList** control to the **Customer** class constructor by using the **SelectedValue** property wrapped in a new **Guid** object.

```
' Instantiate new Customer object with user input
currentCustomer = New CustomerManagementEntities.Customer(
    Nothing, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
    CustomerAddressTextBox.Text, CustomerZipCodeTextBox.Text,
CustomerCityTextBox.Text,
    CustomerStateTextBox.Text, New
Guid(CustomerCountryDropDownList.SelectedValue),
CustomerPhoneTextBox.Text,
    CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text,
CustomerNewsSubscriberCheckBox.Checked,
    Integer.Parse(CustomerCreditLimitTextBox.Text), DateTime.Now,
"", Nothing, "")
```

- In the Customer.ascx.vb code window, replace the following lines of code,

```
' Instantiate new Customer object with user input
currentCustomer = New CustomerManagementEntities.Customer(
    Nothing, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
    CustomerAddressTextBox.Text, CustomerZipCodeTextBox.Text,
CustomerCityTextBox.Text,
    CustomerStateTextBox.Text, Nothing,
CustomerPhoneTextBox.Text,
    CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text,
    Integer.Parse(CustomerCreditLimitTextBox.Text),
CustomerNewsSubscriberCheckBox.Checked,
    DateTime.Now, "", Nothing, "")
```

with

```
' Instantiate new Customer object with user input
currentCustomer = New CustomerManagementEntities.Customer(
    Nothing, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
    CustomerAddressTextBox.Text, CustomerZipCodeTextBox.Text,
CustomerCityTextBox.Text,
    CustomerStateTextBox.Text, New
Guid(CustomerCountryDropDownList.SelectedValue),
CustomerPhoneTextBox.Text,
    CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text,
CustomerNewsSubscriberCheckBox.Checked,
    Integer.Parse(CustomerCreditLimitTextBox.Text),
DateTime.Now, "", Nothing, "")
```

3. Build the user control, and fix any errors.

- In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.

Note: Note that the Build succeeded message appears in the **Build** pane of the Output window.

Exercise 3: Modifying a Data Source

▶ Task 1: Create the Customers Web Form

1. Create the **Customers** Web Form based on the Site.master master page.
    a. In Solution Explorer, right-click **D:\Labfiles\Starter\M8\VB \CustomerManagement**, and then click **Add New Item**.
    b. In the **Add New Item - D:\Labfiles\Starter\M8\VB \CustomerManagement** dialog box, in the middle pane, click **Web Form**, and in the **Name** box, type **Customers.aspx**.
    c. In the **Add New Item - D:\Labfiles\Starter\M8\VB \CustomerManagement** dialog box, select both the **Place code in separate file** and **Select master page** check boxes, and then click **Add**.
    d. In the **Select a Master Page** dialog box, in the **Contents of folder** box, click **Site.master**, and then click **OK**.
2. Open the **Customers** Web Form in the Design view.
    - In the Customers.aspx window, click **Design**.

▶ Task 2: Add the SqlDataSource control to the Web Form

1. Add the **SqlDataSource** control to the **Customers.aspx** Web Form.
    a. In the Customers.aspx window, point to **Toolbox**.
    b. In the Toolbox, expand **Data**, and then double-click the **SqlDataSource** control.
2. Rename the **SqlDataSource** control to **CustomersSqlDataSource**.
    - In the Properties window, change the value of the **ID** property of the **SqlDataSource1** control to **CustomersSqlDataSource**.
3. Save the **Customers** Web Form.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customers.aspx**, or press CTRL+S.

▶ Task 3: Configure the SqlDataSource control

1. In Design view, with the **CustomersSqlDataSource** control selected, display the Smart Tag.
   - In the Customers.aspx window, click the button with the arrow on the right side of the control, or right-click the **CustomersSqlDataSource** control, and then click **Show Smart Tag**.
2. Open the Configure Data Source Wizard.
   - In the Customers.aspx window, on the Smart Tag, click **Configure Data Source**.
3. Select the **CustomerManagementConnectionString** connection string for the **CustomersSqlDataSource** control.
   - On the **Choose Your Data Connection** page of the Configure Data Source – CustomersSqlDataSource Wizard, in the **Which data connection should your application use to connect to the database?** list, click **CustomerManagementConnectionString**, and then click **Next**.
4. Configure the SELECT statement to include all the columns from the **Customers** table, and ensure that it is possible to manipulate the data in the data source, and use optimistic concurrency.
   a. On the **Configure the Select Statement** page, ensure that the **Specify columns from a table or view** option is selected, and then in the **Name** list, click **Customers**.
   b. On the **Configure the Select Statement** page, in the **Columns** box, select the asterisk (*****) check box.
   c. On the **Configure the Select Statement** page, click **Advanced**.
   d. In the **Advanced SQL Generation Options** dialog box, select both the **Generate INSERT, UPDATE, and DELETE statements**, and the **Use optimistic concurrency** check boxes, and then click **OK**.
   e. On the **Configure the Select Statement** page, click **Next**.
   f. On the **Test Query** page, click **Finish**.
5. Save the **Customers** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customers.aspx**, or press CTRL+S.

▶ Task 4: Add the ListView control to the Web Form

1. Add the **ListView** control to the **Customers** Web Form.
   - In the Customers.aspx window, point to **Toolbox**, expand **Data**, and then double-click the **ListView** control.
2. Rename the **ListView** control as **CustomersListView**.
   - In the Properties window of the **ListView** control, change the value of the **ID** property to **CustomersListView**, and then press ENTER.
3. Bind the **CustomersListView** control to the **CustomersSqlDataSource** control by using the Smart Tag.
   a. In the Customers.aspx window, right-click the **CustomersListView** control, and then click **Show Smart Tag**.
   b. In the **ListView Tasks** dialog box, in the **Choose Data Source** list, click **CustomersSqlDataSource**.
4. Enable paging in the **CustomersListView** control.
   a. In the **ListView Tasks** dialog box, click **Configure ListView**.
   b. In the **Configure ListView** dialog box, select the **Enable Paging** check box, and then click **OK**.
5. Save the **Customers** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customers.aspx**, or press CTRL+S.
6. Build the user control, and fix any errors.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.

> **Note:** Notice the Build succeeded message that displays in the **Build** pane of the Output window.

7. View the **Customers** Web Form in the browser.
   - In Solution Explorer, under D:\Labfiles\Starter\M8\VB \CustomerManagement, right-click **Customers.aspx**, and then click **View in Browser**.

Note: Notice that the **CustomersListView** control contains no customers.

8. Close Windows® Internet Explorer®.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 5: Add code to manually save a customer to a data source

1. Open the **Customer** user control in the Code view.
   - In the CustomerManagement – Microsoft Visual Studio window, click **Customer.ascx.vb**.
2. Import the namespace used for accessing a SQL Server 2008 database, **System.Data.SqlClient**.
   - In the Customer.ascx.vb window, insert the following code at the top of the code file:

```
Imports System.Data.SqlClient
```

3. Import the namespace used with the disconnected ADO.NET layer, **System.Data**.
   - In the Customer.ascx.vb window, insert the following code at the top of the code file:

```
Imports System.Data
```

4. Import the namespace for reading the connection string from the web.config file, **System.Configuration**.
   - In the Customer.ascx.vb window, insert the following code at the top of the code file:

```
Imports System.Configuration
```

5. Locate the **CustomerInsertButton_Click** event handler.

```
''' <summary>
''' Saves the current customer information and adds default values
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerInsertButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CustomerInsertButton.Click
    ' Did page validation succeed?
    If Not Page.IsValid Then
        Return
    End If

    ' Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name
    ' Add the user credit limit
    currentCustomer.CreditLimit = 50000
End Sub
```

- In the Customer.ascx.vb window, locate the **CustomerInsertButton_Click** event handler.

```
''' <summary>
''' Saves the current customer information and adds default
values
''' </summary>
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub CustomerInsertButton_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
CustomerInsertButton.Click
    ' Did page validation succeed?
    If Not Page.IsValid Then
        Return
    End If

    ' Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name
    ' Add the user credit limit
    currentCustomer.CreditLimit = 50000
End Sub
```

6. Remove the assignment of the value **50000** to the **CreditLimit** property.

```
' Add the user credit limit
currentCustomer.CreditLimit = 50000
```

- In the **CustomerInsertButton_Click** event handler, delete the following code.

```
' Add the user credit limit
currentCustomer.CreditLimit = 50000
```

7. Append the following code to the **CustomerInsertButton_Click** event handler, by using a code snippet named **ADO.NET Insert Customer**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder.

```
' Create and instantiate connection
Using customerManagementConnection As New SqlConnection()
    ' Initialize connection string from web.config
    customerManagementConnection.ConnectionString =

ConfigurationManager.ConnectionStrings("CustomerManagementConnecti
onString").ConnectionString

    ' Open connection
    customerManagementConnection.Open()
    ' Declare and instantiate data adapter
    Dim customerManagementDataAdapter As New SqlDataAdapter()

    ' Declare and instantiate command objects
    Dim selectCommand As New SqlCommand("SELECT * FROM Customers",
        customerManagementConnection)
    Dim insertCommand As New SqlCommand(
        "INSERT INTO Customers (FirstName, LastName, Address,
ZipCode, City, State, CountryID, Phone, EmailAddress, " &
        "Url, CreditLimit, NewsSubscriber, CreatedDate, CreatedBy)
VALUES(@FirstName, @LastName, @Address, @ZipCode, @City, @State, "
&
        "@CountryID, @Phone, @EmailAddress, @WebAddress,
@CreditLimit, @NewsSubscriber, @CreatedDate, @CreatedBy)",
customerManagementConnection)
```

(*Code continued on the following page.*)

```
    ' Assign command objects
    customerManagementDataAdapter.SelectCommand = selectCommand
    customerManagementDataAdapter.InsertCommand = insertCommand

    ' Declare and instantiate parameter objects
    Dim insertFirstNameParameter As New SqlParameter("@FirstName",
SqlDbType.NVarChar, 50, "FirstName")
    Dim insertLastNameParameter As New SqlParameter("@LastName",
SqlDbType.NVarChar, 30, "LastName")
    Dim insertAddressParameter As New SqlParameter("@Address",
SqlDbType.NVarChar, 50, "Address")
    Dim insertZipCodeParameter As New SqlParameter("@ZipCode",
SqlDbType.NVarChar, 10, "ZipCode")
    Dim insertCityParameter As New SqlParameter("@City",
SqlDbType.NVarChar, 30, "City")
    Dim insertStateParameter As New SqlParameter("@State",
SqlDbType.NVarChar, 30, "State")
    Dim insertCountryIDParameter As New SqlParameter("@CountryID",
SqlDbType.UniqueIdentifier, 0, "CountryID")
    Dim insertPhoneParameter As New SqlParameter("@Phone",
SqlDbType.VarChar, 30, "Phone")
    Dim insertEmailAddressParameter As New
SqlParameter("@EmailAddress", SqlDbType.NVarChar, 50,
"EmailAddress")
    Dim insertWebAddressParameter As New
SqlParameter("@WebAddress", SqlDbType.NVarChar, 80, "Url")
    Dim insertCreditLimitParameter As New
SqlParameter("@CreditLimit", SqlDbType.Int, 0, "CreditLimit")
    Dim insertNewsSubscriberParameter As New
SqlParameter("@NewsSubscriber", SqlDbType.Bit, 0,
"NewsSubscriber")
    Dim insertCreatedDateParameter As New
SqlParameter("@CreatedDate", SqlDbType.SmallDateTime, 0,
"CreatedDate")
    Dim insertCreatedByParameter As New SqlParameter("@CreatedBy",
SqlDbType.VarChar, 15, "CreatedBy")
```

(*Code continued on the following page.*)

```
    ' Assign parameters to command object
    insertCommand.Parameters.Add(insertFirstNameParameter)
    insertCommand.Parameters.Add(insertLastNameParameter)
    insertCommand.Parameters.Add(insertAddressParameter)
    insertCommand.Parameters.Add(insertZipCodeParameter)
    insertCommand.Parameters.Add(insertCityParameter)
    insertCommand.Parameters.Add(insertStateParameter)
    insertCommand.Parameters.Add(insertCountryIDParameter)
    insertCommand.Parameters.Add(insertPhoneParameter)
    insertCommand.Parameters.Add(insertEmailAddressParameter)
    insertCommand.Parameters.Add(insertWebAddressParameter)
    insertCommand.Parameters.Add(insertCreditLimitParameter)
    insertCommand.Parameters.Add(insertNewsSubscriberParameter)
    insertCommand.Parameters.Add(insertCreatedDateParameter)
    insertCommand.Parameters.Add(insertCreatedByParameter)

    ' Declare and instantiate dataset
    Dim customerManagementDataSet As New
DataSet("CustomerManagementDataSet")
    ' Apply the full schema from the data source

customerManagementDataAdapter.FillSchema(customerManagementDataSet
, SchemaType.Source, "Customers")
    customerManagementDataAdapter.MissingSchemaAction =
MissingSchemaAction.AddWithKey
    customerManagementDataAdapter.MissingMappingAction =
MissingMappingAction.Passthrough
    ' Populate Customers DataTable
    customerManagementDataAdapter.Fill(customerManagementDataSet,
"Customers")

    ' Create new row locally
    Dim newCustomerDataRow As DataRow =
customerManagementDataSet.Tables("Customers").NewRow()
    newCustomerDataRow("ID") = Guid.NewGuid()
    newCustomerDataRow("FirstName") = currentCustomer.FirstName
    newCustomerDataRow("LastName") = currentCustomer.LastName
    newCustomerDataRow("Address") = currentCustomer.Address
    newCustomerDataRow("ZipCode") = currentCustomer.ZipCode
    newCustomerDataRow("City") = currentCustomer.City
    newCustomerDataRow("State") = currentCustomer.State
    newCustomerDataRow("CountryID") = currentCustomer.CountryID
```

(*Code continued on the following page.*)

```
    newCustomerDataRow("Phone") = currentCustomer.Phone
    newCustomerDataRow("EmailAddress") =
currentCustomer.EmailAddress
    newCustomerDataRow("Url") = currentCustomer.EmailAddress
    newCustomerDataRow("CreditLimit") =
currentCustomer.CreditLimit
    newCustomerDataRow("NewsSubscriber") =
currentCustomer.NewsSubscriber
    newCustomerDataRow("CreatedDate") =
currentCustomer.CreatedDate
    newCustomerDataRow("CreatedBy") = currentCustomer.CreatedBy

    ' Insert new row locally

customerManagementDataSet.Tables("Customers").Rows.Add(newCustomer
DataRow)

    ' Update data source
    If
customerManagementDataAdapter.Update(customerManagementDataSet,
"Customers") = 1 Then
        ' Instantiate new Customer object
        currentCustomer = New
CustomerManagementEntities.Customer()
        ' Reload page to refresh with "blank" input controls
        Response.Redirect("~/InsertCustomer.aspx")
    End If
End Using
```

a. In the CustomerManagement – Microsoft Visual Studio window, right-click the blank line following **currentCustomer.CreatedBy = Context.User.Identity.Name**, and then click **Insert Snippet**.

b. In the context menu, double-click **My Code Snippets**, and then double-click **ADO.NET Insert Customer**.

```
' Create and instantiate connection
Using customerManagementConnection As New SqlConnection()
    ' Initialize connection string from web.config
    customerManagementConnection.ConnectionString =

ConfigurationManager.ConnectionStrings("CustomerManagementConn
ectionString").ConnectionString
```

(*Code continued on the following page.*)

```
    ' Open connection
    customerManagementConnection.Open()
    ' Declare and instantiate data adapter
    Dim customerManagementDataAdapter As New SqlDataAdapter()

    ' Declare and instantiate command objects
    Dim selectCommand As New SqlCommand("SELECT * FROM
Customers",
        customerManagementConnection)
    Dim insertCommand As New SqlCommand(
        "INSERT INTO Customers (FirstName, LastName, Address,
ZipCode, City, State, CountryID, Phone, EmailAddress, " &
        "Url, CreditLimit, NewsSubscriber, CreatedDate,
CreatedBy) VALUES(@FirstName, @LastName, @Address, @ZipCode,
@City, @State, " &
        "@CountryID, @Phone, @EmailAddress, @EmailAddress,
@CreditLimit, @NewsSubscriber, @CreatedDate, @CreatedBy)",
customerManagementConnection)

    ' Assign command objects
    customerManagementDataAdapter.SelectCommand =
selectCommand
    customerManagementDataAdapter.InsertCommand =
insertCommand

    ' Declare and instantiate parameter objects
    Dim insertFirstNameParameter As New
SqlParameter("@FirstName", SqlDbType.NVarChar, 50,
"FirstName")
    Dim insertLastNameParameter As New
SqlParameter("@LastName", SqlDbType.NVarChar, 30, "LastName")
    Dim insertAddressParameter As New SqlParameter("@Address",
SqlDbType.NVarChar, 50, "Address")
    Dim insertZipCodeParameter As New SqlParameter("@ZipCode",
SqlDbType.NVarChar, 10, "ZipCode")
    Dim insertCityParameter As New SqlParameter("@City",
SqlDbType.NVarChar, 30, "City")
    Dim insertStateParameter As New SqlParameter("@State",
SqlDbType.NVarChar, 30, "State")
    Dim insertCountryIDParameter As New
SqlParameter("@CountryID", SqlDbType.UniqueIdentifier, 0,
"CountryID")
```

(*Code continued on the following page.*)

```
    Dim insertPhoneParameter As New SqlParameter("@Phone",
SqlDbType.VarChar, 30, "Phone")
    Dim insertEmailAddressParameter As New
SqlParameter("@EmailAddress", SqlDbType.NVarChar, 50,
"EmailAddress")
    Dim insertWebAddressParameter As New
SqlParameter("@WebAddress", SqlDbType.NVarChar, 80, "Url")
    Dim insertCreditLimitParameter As New
SqlParameter("@CreditLimit", SqlDbType.Int, 0, "CreditLimit")
    Dim insertNewsSubscriberParameter As New
SqlParameter("@NewsSubscriber", SqlDbType.Bit, 0,
"NewsSubscriber")
    Dim insertCreatedDateParameter As New
SqlParameter("@CreatedDate", SqlDbType.SmallDateTime, 0,
"CreatedDate")
    Dim insertCreatedByParameter As New
SqlParameter("@CreatedBy", SqlDbType.VarChar, 15, "CreatedBy")

    ' Assign parameters to command object
    insertCommand.Parameters.Add(insertFirstNameParameter)
    insertCommand.Parameters.Add(insertLastNameParameter)
    insertCommand.Parameters.Add(insertAddressParameter)
    insertCommand.Parameters.Add(insertZipCodeParameter)
    insertCommand.Parameters.Add(insertCityParameter)
    insertCommand.Parameters.Add(insertStateParameter)
    insertCommand.Parameters.Add(insertCountryIDParameter)
    insertCommand.Parameters.Add(insertPhoneParameter)
    insertCommand.Parameters.Add(insertEmailAddressParameter)
    insertCommand.Parameters.Add(insertWebAddressParameter)
    insertCommand.Parameters.Add(insertCreditLimitParameter)

insertCommand.Parameters.Add(insertNewsSubscriberParameter)
    insertCommand.Parameters.Add(insertCreatedDateParameter)
    insertCommand.Parameters.Add(insertCreatedByParameter)

    ' Declare and instantiate dataset
    Dim customerManagementDataSet As New
DataSet("CustomerManagementDataSet")
    ' Apply the full schema from the data source

customerManagementDataAdapter.FillSchema(customerManagementDat
aSet, SchemaType.Source, "Customers")
    customerManagementDataAdapter.MissingSchemaAction =
MissingSchemaAction.AddWithKey
    customerManagementDataAdapter.MissingMappingAction =
MissingMappingAction.Passthrough
```

(*Code continued on the following page.*)

```
    ' Populate Customers DataTable

customerManagementDataAdapter.Fill(customerManagementDataSet,
"Customers")

    ' Create new row locally
    Dim newCustomerDataRow As DataRow =
customerManagementDataSet.Tables("Customers").NewRow()
    newCustomerDataRow("ID") = Guid.NewGuid()
    newCustomerDataRow("FirstName") =
currentCustomer.FirstName
    newCustomerDataRow("LastName") = currentCustomer.LastName
    newCustomerDataRow("Address") = currentCustomer.Address
    newCustomerDataRow("ZipCode") = currentCustomer.ZipCode
    newCustomerDataRow("City") = currentCustomer.City
    newCustomerDataRow("State") = currentCustomer.State
    newCustomerDataRow("CountryID") =
currentCustomer.CountryID
    newCustomerDataRow("Phone") = currentCustomer.Phone
    newCustomerDataRow("EmailAddress") =
currentCustomer.EmailAddress
    newCustomerDataRow("Url") = currentCustomer.EmailAddress
    newCustomerDataRow("CreditLimit") =
currentCustomer.CreditLimit
    newCustomerDataRow("NewsSubscriber") =
currentCustomer.NewsSubscriber
    newCustomerDataRow("CreatedDate") =
currentCustomer.CreatedDate
    newCustomerDataRow("CreatedBy") =
currentCustomer.CreatedBy

    ' Insert new row locally

customerManagementDataSet.Tables("Customers").Rows.Add(newCust
omerDataRow)

    ' Update data source
    If
customerManagementDataAdapter.Update(customerManagementDataSet
, "Customers") = 1 Then
        ' Instantiate new Customer object
        currentCustomer = New
CustomerManagementEntities.Customer()
        ' Reload page to refresh with "blank" input controls
        Response.Redirect("~/InsertCustomer.aspx")
    End If
End Using
```

8. Save the Customer code-behind file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx.vb**.

▶ Task 6: Update sitemap to enable view all customers

1. Open the **web.sitemap** file.
   - In Solution Explorer, double-click **web.sitemap**.
2. Append the new siteMapNode element to the Customers siteMapNode.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers.aspx" />
```

   - Append the following markup to the **siteMapNode** element with a **title** attribute value of **Customers**.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers.aspx" />
```

3. Save and close the **web.sitemap** file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**, or press CTRL+S.
   b. In the web.sitemap window, click the **Close** button.

▶ Task 7: Create a customer and verify the Data Source

1. Build the **CustomerManagement** solution.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Solution**.

> **Note:** Notice the Validation Complete message that displays in the **Build** pane of the Output window.

2. Run the **CustomerManagement** Web application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

3. Open the **InsertCustomer** Web Form, and click **New** on the **Customers** menu.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
4. Create a new customer by using the following information, and then click the **Insert** button:
   - First Name: **Kim**
   - Last Name: **Abercrombie**
   - Address: **9876 Maine Road**
   - Zip Code: **M24NG**
   - City: **Manchester**
   - Country: **Great Britain**
   - Phone: **0161-123 555**
   - Email Address: **kim@litwareinc.com**
   - Web Address: **http://www.litwareinc.com**
   - Credit Limit: **50000**
   - News Subscriber: **Yes**

> **Note:** If an AutoComplete message box displays, click **No**.

5. Check that the new customer has been added to the data source by using the new **Customers** Web Form.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **All**.

> **Note:** Verify that the new customer has indeed been added in the data source.

6. Close Internet Explorer.
    a. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
    b. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 8: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 8

Lab Answer Key: Managing Data in a Microsoft® ASP.NET 4.0 Web Application (Visual C#)

Contents:

Lab: Managing Data in an ASP.NET 4 Web Application

(C#)

Exercise 1: Connecting to a Data Source

▶ Task 1: Open an existing ASP.NET Web project

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M8\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M8\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Add a SQL Server 2008 Express Database

- Add a new database folder named **App_Data**, and an existing database to the database folder, **D:\LabFiles\Starter\M8\CustomerManagement.mdf**.
  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M8\CS\CustomerManagement**, point to **Add ASP.NET Folder**, and then click **App_Data**.
  b. In Solution Explorer, right-click **App_Data**, and then click **Add Existing Item**.
  c. In the **Add Existing Item** dialog box, in the **File name** box, type **D:\LabFiles\Starter\M8\CustomerManagement.mdf**, and then click **Add**.

▶ Task 3: Add a data source control to the user control

1. Open the **Customer** user control in the Design view.
   - In Solution Explorer, under D:\Labfiles\Starter\M8\CS \CustomerManagement, right-click **Customer.ascx**, and then click **View Designer**.
2. Add a **SqlDataSource** control to the user control to connect to a SQL Server database.
   a. In the Customer.ascx window, place the cursor below the **Insert** and **Cancel** buttons, and then point to **Toolbox**.
   b. In the **Toolbox**, expand **Data**, and then double-click the **SqlDataSource** control.
3. Rename the **SqlDataSource** control as **CountriesSqlDataSource**.
   - With the **SqlDataSource** control selected, in the Properties window, change the value of the **ID** property from **SqlDataSource1** to **CountriesSqlDataSource**.
4. Save the **Customer** user control.
   - In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press the CTRL+S keys.

▶ Task 4: Configure a data source control

1. In Design view, with the **CountriesSqlDataSource** control selected, display the Smart Tag.
   - In the Customer.ascx window, click the button with the arrow on the right side of the control, or right-click the **CountriesSqlDataSource** control, and then click **Show Smart Tag**.
2. Open the Configure Data Source Wizard, connect to the **CustomerManagement.mdf** database, and then create a new connection string named **CustomerManagementConnectionString**.
   a. In the Customer.ascx window, on the Smart Tag, click **Configure Data Source**.
   b. On the **Choose Your Data Connection** page of the Configure Data Source - CountriesSqlDataSource Wizard, in the **Which data connection should your application use to connect to the database?** list, click **CustomerManagement.mdf**, and then click **Next**.

c. On the **Save the Connection String to the Application Configuration File** page of the Configure Data Source - CountriesSqlDataSource Wizard, in the **Yes, save this connection as** box, type **CustomerManagementConnectionString**, and then click **Next**.

3. Configure the SELECT statement to include the **ID** and **Name** columns from the **Countries** table.

   a. On the **Configure the Select Statement** page, ensure that the **Specify columns from a table or view** option is selected, and ensure that **Countries** is selected in the **Name** list.

   b. In the **Columns** list, select the **ID** and **Name** check boxes, and then click **Next**.

4. Test the query, and check whether you get the correct data from the Countries table.

   a. On the **Test Query** page, click **Test Query**.

> **Note:** Ensure that the returned rows include the values for the ID and Name columns for various countries.

   b. On the **Test Query** page, click **Finish**.

5. Save the **Customer** user control.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx**, or press the CTRL+S keys.

Exercise 2: Binding a Server Control to a Data Source

▶ Task 1: Bind the DropDownList control to a data source

1. Open the **Customer** user control in the Source view.
   - In the Customer.ascx window, click **Source**.
2. Locate the markup for the **CustomerCountryDropDownList** control.
3. Remove the static **ListItem** element from the **CustomerCountryDropDownList** control.

```
<asp:ListItem>USA</asp:ListItem>
```

4. Bind the **CustomerCountryDropDownList** control to the **CountriesSqlDataSource** control by using the **DataSourceID** attribute.
   - In Customer.ascx window, append the opening **CustomerCountryDropDownList** tag with the following markup.

```
DataSourceID="CountriesSqlDataSource"
```

5. Set the value field of the **CustomerCountryDropDownList** control to the **ID** column of the database by using the **DataValueField** attribute.
   - In the Customer.ascx window, append the opening **CustomerCountryDropDownList** tag with the following markup.

```
DataValueField="ID"
```

6. Set the text field of the **CustomerCountryDropDownList** control to the **Name** column of the database by using the **DataTextField** attribute.
   - In the Customer.ascx window, append the opening **CustomerCountryDropDownList** tag with the following markup.

```
DataTextField="Name"
```

- Build the user control, and fix any errors.
  - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.

> **Note:** Notice the Build succeeded message that displays in the **Build** pane of the Output window.

▶ Task 2: Pass the values to the Customer object

1. Open the **Customer** user control in the Code view.
   - In Solution Explorer, right-click **Customer.ascx**, and then click **View Code**.
2. Locate the code for the private **instantiateCustomerObject** method, and pass the selected value of the **CustomerCountryDropDownList** control to the **Customer** class constructor by using the **SelectedValue** property wrapped in a new **Guid** object.

```
// Instantiate new Customer object with user input
currentCustomer = new CustomerManagementEntities.Customer(
    null, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
    CustomerAddressTextBox.Text, CustomerZipCodeTextBox.Text,
    CustomerCityTextBox.Text, CustomerStateTextBox.Text,
    new Guid(CustomerCountryDropDownList.SelectedValue),
CustomerPhoneTextBox.Text,
    CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text,
    int.Parse(CustomerCreditLimitTextBox.Text),
CustomerNewsSubscriberCheckBox.Checked,
    DateTime.Now, "", null, "");
```

- In the Customer.ascx.cs code window, replace the following lines of code,

```
// Instantiate new Customer object with user input
currentCustomer = new CustomerManagementEntities.Customer(
    null, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
    CustomerAddressTextBox.Text, CustomerZipCodeTextBox.Text,
    CustomerCityTextBox.Text, CustomerStateTextBox.Text,
    null, CustomerPhoneTextBox.Text,
CustomerEmailAddressTextBox.Text,
    CustomerWebAddressTextBox.Text,
int.Parse(CustomerCreditLimitTextBox.Text),
    CustomerNewsSubscriberCheckBox.Checked, DateTime.Now, "",
null, "");
```

with

```
// Instantiate new Customer object with user input
currentCustomer = new CustomerManagementEntities.Customer(
    null, CustomerFirstNameTextBox.Text,
CustomerLastNameTextBox.Text,
    CustomerAddressTextBox.Text, CustomerZipCodeTextBox.Text,
    CustomerCityTextBox.Text, CustomerStateTextBox.Text,
    new Guid(CustomerCountryDropDownList.SelectedValue),
CustomerPhoneTextBox.Text,
    CustomerEmailAddressTextBox.Text,
CustomerWebAddressTextBox.Text,
    int.Parse(CustomerCreditLimitTextBox.Text),
CustomerNewsSubscriberCheckBox.Checked,
    DateTime.Now, "", null, "");
```

3. Build the user control, and fix any errors.

- In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.

Note: Note that the Build succeeded message appears in the **Build** pane of the Output window.

Exercise 3: Modifying a Data Source

▶ Task 1: Create the Customers Web Form

1. Create the **Customers** Web Form, based on the Site.master master page.
    a. In Solution Explorer, right-click **D:\Labfiles\Starter\M8\CS \CustomerManagement**, and then click **Add New Item**.
    b. In the **Add New Item - D:\Labfiles\Starter\M8\CS \CustomerManagement** dialog box, in the middle pane, click **Web Form**, and in the **Name** box, type **Customers.aspx**.
    c. In the **Add New Item - D:\Labfiles\Starter\M8\CS \CustomerManagement** dialog box, select both the **Place code in separate file** and **Select master page** check boxes, and then click **Add**.
    d. In the **Select a Master Page** dialog box, in the **Contents of folder** box, click **Site.master**, and then click **OK**.
2. Open the **Customers** Web Form in the Design view.
    - In the Customers.aspx window, click **Design**.

▶ Task 2: Add the SqlDataSource control to the Web Form

1. Add the **SqlDataSource** control to the **Customers.aspx** Web Form.
    a. In the Customers.aspx window, point to **Toolbox**.
    b. In the Toolbox, expand **Data**, and then double-click the **SqlDataSource** control.
2. Rename the **SqlDataSource** control to **CustomersSqlDataSource**.
    - In the Properties window, change the value of the **ID** property of the **SqlDataSource1** control to **CustomersSqlDataSource**.
3. Save the Customers Web Form.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customers.aspx**, or press CTRL+S.

▶ Task 3: Configure the SqlDataSource control

1. In Design view, with the **CustomersSqlDataSource** control selected, display the Smart Tag.
   - In the Customers.aspx window, click the button with the arrow on the right side of the control, or right-click the **CustomersSqlDataSource** control, and then click **Show Smart Tag**.
2. Open the Configure Data Source Wizard.
   - In the Customers.aspx window, on the Smart Tag, click **Configure Data Source**.
3. Select the **CustomerManagementConnectionString** connection string for the **CustomersSqlDataSource** control.
   - On the **Choose Your Data Connection** page of the Configure Data Source – CustomersSqlDataSource Wizard, in the **Which data connection should your application use to connect to the database?** list, click **CustomerManagementConnectionString**, and then click **Next**.
4. Configure the SELECT statement to include all the columns from the **Customers** table, and ensure that it is possible to manipulate the data in the data source, and use optimistic concurrency.
   a. On the **Configure the Select Statement** page, ensure that the **Specify columns from a table or view** option is selected, and then in the **Name** list, click **Customers**.
   b. On the **Configure the Select Statement** page, in the **Columns** box, select the asterisk (*****) check box.
   c. On the **Configure the Select Statement** page, click **Advanced**.
   d. In the **Advanced SQL Generation Options** dialog box, select both the **Generate INSERT, UPDATE, and DELETE statements**, and the **Use optimistic concurrency** check boxes, and then click **OK**.
   e. On the **Configure the Select Statement** page, click **Next**.
   f. On the **Test Query** page, click **Finish**.
5. Save the Customers Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customers.aspx**, or press CTRL+S.

▶ Task 4: Add the ListView control to the Web Form

1. Add the **ListView** control to the **Customers** Web Form.
   - In the Customers.aspx window, point to **Toolbox**, expand **Data**, and then double-click the **ListView** control.
2. Rename the **ListView** control as **CustomersListView**.
   - In the Properties window of the **ListView** control, change the value of the **ID** property to **CustomersListView**, and then press ENTER.
3. Bind the **CustomersListView** control to the **CustomersSqlDataSource** control by using the Smart Tag.
   a. In the Customers.aspx window, right-click the **CustomersListView** control, and then click **Show Smart Tag**.
   b. In the **ListView Tasks** dialog box, in the **Choose Data Source** list, click **CustomersSqlDataSource**.
4. Enable paging in the **CustomersListView** control.
   a. In the **ListView Tasks** dialog box, click **Configure ListView**.
   b. In the Configure **ListView** dialog box, select the **Enable Paging** check box, and then click **OK**.
5. Save the Customers Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customers.aspx**, or press CTRL+S.
6. Build the user control, and fix any errors.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.

> **Note:** Notice the Build succeeded message that displays in the **Build** pane of the Output window.

7. View the **Customers** Web Form in the browser.
   - In Solution Explorer, under D:\Labfiles\Starter\M8\CS \CustomerManagement, right-click **Customers.aspx**, and then click **View in Browser**.

> **Note:** Notice that the **CustomersListView** control contains no customers.

8. Close Windows® Internet Explorer®.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 5: Add code to manually save a customer to a data source

1. Open the **Customer** user control in the Code view.
   - In the CustomerManagement – Microsoft Visual Studio window, click **Customer.ascx.cs**.
2. Import the namespace used for accessing a SQL Server 2008 database, **System.Data.SqlClient**.
   - In the Customer.ascx.cs window, insert the following code at the top of the code file:

```
using System.Data.SqlClient;
```

3. Import the namespace used with the disconnected ADO.NET layer, **System.Data**.
   - In the Customer.ascx.cs window, insert the following code at the top of the code file:

```
using System.Data;
```

4. Import the namespace for reading the connection string from the web.config file, **System.Configuration**.
   - In the Customer.ascx.cs window, insert the following code at the top of the code file:

```
using System.Configuration;
```

5. Locate the **CustomerInsertButton_Click** event handler.

```
protected void CustomerInsertButton_Click(object sender, EventArgs
e)
{
    // Did page validation succeed?
    if (!Page.IsValid)
        return;

    // Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name;
    // Add the user credit limit
    currentCustomer.CreditLimit = 50000;
}
```

- In the Customer.ascx.cs window, locate the **CustomerInsertButton_Click** event handler.

```
protected void CustomerInsertButton_Click(object sender,
EventArgs e)
{
    // Did page validation succeed?
    if (!Page.IsValid)
        return;

    // Add the current user name
    currentCustomer.CreatedBy = Context.User.Identity.Name;
    // Add the user credit limit
    currentCustomer.CreditLimit = 50000;
}
```

6. Remove the assignment of the value **50000** to the **CreditLimit** property.

```
// Add the user credit limit
currentCustomer.CreditLimit = 50000;
```

- In the **CustomerInsertButton_Click** event handler, delete the following code.

```
// Add the user credit limit
currentCustomer.CreditLimit = 50000;
```

7. Append the following code to the **CustomerInsertButton_Click** event handler, by using a code snippet named **ADO.NET Insert Customer**. The code snippet has been supplied by the senior developer, and is placed in the My Code Snippets folder.

```
// Create and instantiate connection
using (SqlConnection customerManagementConnection = new
SqlConnection())
{
    // Initialize connection string from web.config
    customerManagementConnection.ConnectionString =

ConfigurationManager.ConnectionStrings["CustomerManagementConnecti
onString"].ConnectionString;

    // Open connection
    customerManagementConnection.Open();
    // Declare and instantiate data adapter
    SqlDataAdapter customerManagementDataAdapter = new
SqlDataAdapter();

    // Declare and instantiate command objects
    SqlCommand selectCommand = new SqlCommand("SELECT * FROM
Customers",
        customerManagementConnection);
    SqlCommand insertCommand = new SqlCommand(
        "INSERT INTO Customers (FirstName, LastName, Address,
ZipCode, City, State, CountryID, Phone, EmailAddress, " +
        "Url, CreditLimit, CreatedDate, CreatedBy)
VALUES(@FirstName, @LastName, @Address, @ZipCode, @City, @State, "
+
        "@CountryID, @Phone, @EmailAddress, @EmailAddress,
@CreditLimit, @NewsSubscriber, @CreatedDate, @CreatedBy)",
        customerManagementConnection);

    // Assign command objects
    customerManagementDataAdapter.SelectCommand = selectCommand;
    customerManagementDataAdapter.InsertCommand = insertCommand;
```

(*Code continued on the following page.*)

```
    // Declare and instantiate parameter objects
    SqlParameter insertFirstNameParameter = new
SqlParameter("@FirstName", SqlDbType.NVarChar, 50, "FirstName");
    SqlParameter insertLastNameParameter = new
SqlParameter("@LastName", SqlDbType.NVarChar, 30, "LastName");
    SqlParameter insertAddressParameter = new
SqlParameter("@Address", SqlDbType.NVarChar, 50, "Address");
    SqlParameter insertZipCodeParameter = new
SqlParameter("@ZipCode", SqlDbType.NVarChar, 10, "ZipCode");
    SqlParameter insertCityParameter = new SqlParameter("@City",
SqlDbType.NVarChar, 30, "City");
    SqlParameter insertStateParameter = new SqlParameter("@State",
SqlDbType.NVarChar, 30, "State");
    SqlParameter insertCountryIDParameter = new
SqlParameter("@CountryID", SqlDbType.UniqueIdentifier, 0,
"CountryID");
    SqlParameter insertPhoneParameter = new SqlParameter("@Phone",
SqlDbType.VarChar, 30, "Phone");
    SqlParameter insertEmailAddressParameter = new
SqlParameter("@EmailAddress", SqlDbType.NVarChar, 50,
"EmailAddress");
    SqlParameter insertWebAddressParameter = new
SqlParameter("@WebAddress", SqlDbType.NVarChar, 80, "Url");
    SqlParameter insertCreditLimitParameter = new
SqlParameter("@CreditLimit", SqlDbType.Int, 0, "CreditLimit");
    SqlParameter insertNewsSubscriberParameter = new
SqlParameter("@NewsSubscriber", SqlDbType.Bit, 0,
"NewsSubscriber");
    SqlParameter insertCreatedDateParameter = new
SqlParameter("@CreatedDate", SqlDbType.SmallDateTime, 0,
"CreatedDate");
    SqlParameter insertCreatedByParameter = new
SqlParameter("@CreatedBy", SqlDbType.VarChar, 15, "CreatedBy");


    // Assign parameters to command object
    insertCommand.Parameters.Add(insertFirstNameParameter);
    insertCommand.Parameters.Add(insertLastNameParameter);
    insertCommand.Parameters.Add(insertAddressParameter);
    insertCommand.Parameters.Add(insertZipCodeParameter);
    insertCommand.Parameters.Add(insertCityParameter);
    insertCommand.Parameters.Add(insertStateParameter);
```

(*Code continued on the following page.*)

```
    insertCommand.Parameters.Add(insertCountryIDParameter);
    insertCommand.Parameters.Add(insertPhoneParameter);
    insertCommand.Parameters.Add(insertEmailAddressParameter);
    insertCommand.Parameters.Add(insertWebAddressParameter);
    insertCommand.Parameters.Add(insertCreditLimitParameter);
    insertCommand.Parameters.Add(insertNewsSubscriberParameter);
    insertCommand.Parameters.Add(insertCreatedDateParameter);
    insertCommand.Parameters.Add(insertCreatedByParameter);

    // Declare and instantiate dataset
    DataSet customerManagementDataSet = new
DataSet("CustomerManagementDataSet");
    // Apply the full schema from the data source

customerManagementDataAdapter.FillSchema(customerManagementDataSet
, SchemaType.Source, "Customers");
    customerManagementDataAdapter.MissingSchemaAction =
MissingSchemaAction.AddWithKey;
    customerManagementDataAdapter.MissingMappingAction =
MissingMappingAction.Passthrough;
    // Populate Customers DataTable
    customerManagementDataAdapter.Fill(customerManagementDataSet,
"Customers");

    // Create new row locally
    DataRow newCustomerDataRow =
customerManagementDataSet.Tables["Customers"].NewRow();
    newCustomerDataRow["ID"] = Guid.NewGuid();
    newCustomerDataRow["FirstName"] = currentCustomer.FirstName;
    newCustomerDataRow["LastName"] = currentCustomer.LastName;
    newCustomerDataRow["Address"] = currentCustomer.Address;
    newCustomerDataRow["ZipCode"] = currentCustomer.ZipCode;
    newCustomerDataRow["City"] = currentCustomer.City;
    newCustomerDataRow["State"] = currentCustomer.State;
    newCustomerDataRow["CountryID"] = currentCustomer.CountryID;
    newCustomerDataRow["Phone"] = currentCustomer.Phone;
    newCustomerDataRow["EmailAddress"] =
currentCustomer.EmailAddress;
    newCustomerDataRow["Url"] = currentCustomer.EmailAddress;
    newCustomerDataRow["CreditLimit"] =
currentCustomer.CreditLimit;
    newCustomerDataRow["NewsSubscriber"] =
currentCustomer.NewsSubscriber;
    newCustomerDataRow["CreatedDate"] =
currentCustomer.CreatedDate;
    newCustomerDataRow["CreatedBy"] = currentCustomer.CreatedBy;
```

(*Code continued on the following page.*)

```
    // Insert new row locally

customerManagementDataSet.Tables["Customers"].Rows.Add(newCustomer
DataRow);

    // Update data source
    if
(customerManagementDataAdapter.Update(customerManagementDataSet,
"Customers") == 1)
    {
        // Instantiate new Customer object
        currentCustomer = new
CustomerManagementEntities.Customer();
        // Reload page to refresh with "blank" input controls
        Response.Redirect("~/InsertCustomer.aspx");
    }
}
```

a. In the CustomerManagement – Microsoft Visual Studio window, right-click the blank line following **currentCustomer.CreatedBy = Context.User.Identity.Name;**, and then click **Insert Snippet**.

b. In the context menu, double-click **My Code Snippets**, and then double-click **ADO.NET Insert Customer**.

```
// Create and instantiate connection
using (SqlConnection customerManagementConnection = new
SqlConnection())
{
    // Initialize connection string from web.config
    customerManagementConnection.ConnectionString =

ConfigurationManager.ConnectionStrings["CustomerManagementConn
ectionString"].ConnectionString;

    // Open connection
    customerManagementConnection.Open();
    // Declare and instantiate data adapter
    SqlDataAdapter customerManagementDataAdapter = new
SqlDataAdapter();
```

(*Code continued on the following page.*)

```
    // Declare and instantiate command objects
    SqlCommand selectCommand = new SqlCommand("SELECT * FROM
Customers",
        customerManagementConnection);
    SqlCommand insertCommand = new SqlCommand(
        "INSERT INTO Customers (FirstName, LastName, Address,
ZipCode, City, State, CountryID, Phone, EmailAddress, " +
        "Url, NewsSubscriber, CreditLimit, NewsSubscriber,
CreatedDate, CreatedBy) VALUES(@FirstName, @LastName,
@Address, @ZipCode, @City, @State, " +
        "@CountryID, @Phone, @EmailAddress, @WebAddress,
@CreditLimit, @NewsSubscriber, @CreatedDate, @CreatedBy)",
        customerManagementConnection);

    // Assign command objects
    customerManagementDataAdapter.SelectCommand =
selectCommand;
    customerManagementDataAdapter.InsertCommand =
insertCommand;

    // Declare and instantiate parameter objects
    SqlParameter insertFirstNameParameter = new
SqlParameter("@FirstName", SqlDbType.NVarChar, 50,
"FirstName");
    SqlParameter insertLastNameParameter = new
SqlParameter("@LastName", SqlDbType.NVarChar, 30, "LastName");
    SqlParameter insertAddressParameter = new
SqlParameter("@Address", SqlDbType.NVarChar, 50, "Address");
    SqlParameter insertZipCodeParameter = new
SqlParameter("@ZipCode", SqlDbType.NVarChar, 10, "ZipCode");
    SqlParameter insertCityParameter = new
SqlParameter("@City", SqlDbType.NVarChar, 30, "City");
    SqlParameter insertStateParameter = new
SqlParameter("@State", SqlDbType.NVarChar, 30, "State");
    SqlParameter insertCountryIDParameter = new
SqlParameter("@CountryID", SqlDbType.UniqueIdentifier, 0,
"CountryID");
    SqlParameter insertPhoneParameter = new
SqlParameter("@Phone", SqlDbType.VarChar, 30, "Phone");
    SqlParameter insertEmailAddressParameter = new
SqlParameter("@EmailAddress", SqlDbType.NVarChar, 50,
"EmailAddress");
```

(*Code continued on the following page.*)

```
    SqlParameter insertWebAddressParameter = new
SqlParameter("@WebAddress", SqlDbType.NVarChar, 80, "Url");
    SqlParameter insertCreditLimitParameter = new
SqlParameter("@CreditLimit", SqlDbType.Int, 0, "CreditLimit");
    SqlParameter insertNewsSubscriberParameter = new
SqlParameter("@NewsSubscriber", SqlDbType.Bit, 0,
"NewsSubscriber");
    SqlParameter insertCreatedDateParameter = new
SqlParameter("@CreatedDate", SqlDbType.SmallDateTime, 0,
"CreatedDate");
    SqlParameter insertCreatedByParameter = new
SqlParameter("@CreatedBy", SqlDbType.VarChar, 15,
"CreatedBy");


    // Assign parameters to command object
    insertCommand.Parameters.Add(insertFirstNameParameter);
    insertCommand.Parameters.Add(insertLastNameParameter);
    insertCommand.Parameters.Add(insertAddressParameter);
    insertCommand.Parameters.Add(insertZipCodeParameter);
    insertCommand.Parameters.Add(insertCityParameter);
    insertCommand.Parameters.Add(insertStateParameter);
    insertCommand.Parameters.Add(insertCountryIDParameter);
    insertCommand.Parameters.Add(insertPhoneParameter);
    insertCommand.Parameters.Add(insertEmailAddressParameter);
    insertCommand.Parameters.Add(insertWebAddressParameter);
    insertCommand.Parameters.Add(insertCreditLimitParameter);

insertCommand.Parameters.Add(insertNewsSubscriberParameter);
    insertCommand.Parameters.Add(insertCreatedDateParameter);
    insertCommand.Parameters.Add(insertCreatedByParameter);

    // Declare and instantiate dataset
    DataSet customerManagementDataSet = new
DataSet("CustomerManagementDataSet");
    // Apply the full schema from the data source

customerManagementDataAdapter.FillSchema(customerManagementDat
aSet, SchemaType.Source, "Customers");
    customerManagementDataAdapter.MissingSchemaAction =
MissingSchemaAction.AddWithKey;
    customerManagementDataAdapter.MissingMappingAction =
MissingMappingAction.Passthrough;
```

(*Code continued on the following page.*)

```
    // Populate Customers DataTable

customerManagementDataAdapter.Fill(customerManagementDataSet,
"Customers");

    // Create new row locally
    DataRow newCustomerDataRow =
customerManagementDataSet.Tables["Customers"].NewRow();
    newCustomerDataRow["ID"] = Guid.NewGuid();
    newCustomerDataRow["FirstName"] =
currentCustomer.FirstName;
    newCustomerDataRow["LastName"] = currentCustomer.LastName;
    newCustomerDataRow["Address"] = currentCustomer.Address;
    newCustomerDataRow["ZipCode"] = currentCustomer.ZipCode;
    newCustomerDataRow["City"] = currentCustomer.City;
    newCustomerDataRow["State"] = currentCustomer.State;
    newCustomerDataRow["CountryID"] =
currentCustomer.CountryID;
    newCustomerDataRow["Phone"] = currentCustomer.Phone;
    newCustomerDataRow["EmailAddress"] =
currentCustomer.EmailAddress;
    newCustomerDataRow["Url"] = currentCustomer.EmailAddress;
    newCustomerDataRow["CreditLimit"] =
currentCustomer.CreditLimit;
    newCustomerDataRow["NewsSubscriber"] =
currentCustomer.NewsSubscriber;
    newCustomerDataRow["CreatedDate"] =
currentCustomer.CreatedDate;
    newCustomerDataRow["CreatedBy"] =
currentCustomer.CreatedBy;

    // Insert new row locally

customerManagementDataSet.Tables["Customers"].Rows.Add(newCust
omerDataRow);

    // Update data source
    if
(customerManagementDataAdapter.Update(customerManagementDataSe
t, "Customers") == 1)
    {
        // Instantiate new Customer object
        currentCustomer = new
CustomerManagementEntities.Customer();
        // Reload page to refresh with "blank" input controls
        Response.Redirect("~/InsertCustomer.aspx");
    }
}
```

8. Save the Customer code-behind file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Customer.ascx.cs**.

▶ Task 6: Update sitemap to enable view all customers

1. Open the **web.sitemap** file.
   - In Solution Explorer, double-click **web.sitemap**.
2. Append the new **siteMapNode** element to the **Customers siteMapNode**.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers.aspx" />
```

   - Append the following markup to the **siteMapNode** element with a **title** attribute value of **Customers**.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers.aspx" />
```

3. Save and close the **web.sitemap** file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**, or press CTRL+S.
   b. In the web.sitemap window, click the **Close** button.

▶ Task 7: Create a customer and verify the Data Source

1. Build the **CustomerManagement** solution.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Solution**.

> **Note:** Notice the Validation Complete message that displays in the **Build** pane of the Output window.

2. Run the **CustomerManagement** Web application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

3. Open the **InsertCustomer** Web Form.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **New**.
4. Create a new customer by using the following information, and then click the **Insert** button:
   - First Name: **Kim**
   - Last Name: **Abercrombie**
   - Address: **9876 Maine Road**
   - Zip Code: **M24NG**
   - City: **Manchester**
   - Country: **Great Britain**
   - Phone: **0161-123 555**
   - Email Address: **kim@litwareinc.com**
   - Web Address: **http://www.litwareinc.com**
   - Credit Limit: **50000**
   - News Subscriber: **Yes**

> **Note:** If an AutoComplete message box displays, click **No**.

5. Check that the new customer has been added to the data source by using the new **Customers** Web Form.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **All**.

> **Note:** Verify that the new customer has indeed been added in the data source.

6. Close Internet Explorer.
    a. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
    b. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 8: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 9

Lab Answer Key: Managing Data Access Tasks by Using LINQ (Visual Basic)

Contents:

Lab: Managing Data Access Tasks by Using LINQ

(Visual Basic)

Exercise 1: Loading Data by Using the XmlDataSource Control

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M9\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M9\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Create the ImportCountries Web Form

1. Add a Web Form named **ImportCountries**, that is based on the **Site.master** master page.
   a. In Solution Explorer, right-click **D:\Labfiles\Starter\M9\VB \CustomerManagement**, and then click **Add New Item**.
   b. In the **Add New Item – D:\Labfiles\Starter\M9\VB \CustomerManagement** dialog box, in the middle pane, click **Web Form**.

c. In the **Name** box, type **ImportCountries.aspx**, ensure that the **Place code in separate file** check box is selected, select the **Select master page** check box, and then click **Add**.

d. In the **Select a Master Page** dialog box, in the **Contents of folder** box, click **Site.master**, and then click **OK**.

2. Open the **ImportCountries** Web Form in the Design view.

- In the ImportCountries.aspx window, click **Design**.

▶ Task 3: Add an XmlDataSource control to the Web Form

1. Add an existing XML file, **D:\Labfiles\Starter\M9\Countries.xml**, to the Web site.

a. In Solution Explorer, right-click **D:\Labfiles\Starter\M9\VB \CustomerManagement**, and then click **Add Existing Item**.

b. In the **Add Existing Item** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M9\Countries.xml**, and then click **Add**.

2. Add an **XmlDataSource** control to the **Content** control of the **ImportCountries** Web Form.

a. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

b. In Toolbox, under **Data**, double-click **XmlDataSource** control.

3. Rename the **XmlDataSource** control that you just added to **CountriesXmlDataSource**.

- In the Properties window of the **XmlDataSource** control, change the value of the **ID** property from **XmlDataSource1** to **CountriesXmlDataSource**, and then press ENTER.

4. Save the **ImportCountries** Web Form.

- On the **File** menu, click **Save ImportCountries.aspx**, or press the CTRL+S keys.

▶ Task 4: Configure the XmlDataSource control

1. Open the **Configure Data Source** control by using the **Show Smart Tag** option.
   - In the ImportCountries.aspx window, right-click **CountriesXmlDataSource** control, click **Show Smart Tag**, and then click **Configure Data Source**.
2. Use the **Countries.xml** data file in the **XmlDataSource** control.
   - In the **Data file** box, type **~/Countries.xml**, and then click **OK**.
3. Save the **ImportCountries** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx**, or press CTRL+S.

▶ Task 5: Add a GridView control to the Web Form

1. Add a **GridView** control to the **Content** control of the **ImportCountries** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**, and then double-click **GridView** under **Data**.
2. Rename the **GridView** control as **CountriesGridView**.
   - In the Properties window of the **GridView** control, change the value of the **ID** attribute from **GridView1** to **CountriesGridView**, and then press ENTER.
3. Set the width of the **GridView** control to **100%**.
   - In the Properties window of the **GridView** control, set the value of the **Width** property to **100%**, and then press ENTER.
4. Bind the **CountriesGridView** control to the **CountriesXmlDataSource** control by using the Smart Tag.
   a. In the ImportCountries.aspx window, right-click **CountriesGridView**, and then click **Show Smart Tag**.
   b. In the **GridView Tasks** dialog box, in the **Choose Data Source** list, click **CountriesXmlDataSource**.

5. Save the **ImportCountries** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx**, or press CTRL+S.
6. Build the Web Form and fix any errors.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.

> **Note:** Notice that the Build success message appears in the **Build** pane of the Output window.

7. View the **ImportCountries** Web Form in the browser.
   - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View in Browser**.

> **Note:** Notice that the **CountriesGridView** control displays the content of the XML data file.

8. Close the Windows® Internet Explorer® browser.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Displaying Data by Using LINQ to XML

▶ Task 1: Add a Button control to the Web Form

1. Open the **ImportCountries** Web Form in the Source view.
   - In the ImportCountries.aspx window, click **Source**.
2. Add a **div** element with a class attribute value of **importCountriesHeader** to the **Content** control, above the **GridView** control.
   - Place the cursor before the opening tag of the **GridView** control,

     ```
     <asp:GridView ID="CountriesGridView" runat="server"
     AutoGenerateColumns="False"
     ```

     and then type the following markup.

     ```
     <div class="importCountriesHeader">

     </div>
     ```
3. Add a **Button** control to the **div** element.
   a. In the ImportCountries.aspx window, place the cursor between the opening and closing **div** tags with a **class** attribute value of **importCountriesHeader**.
   b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
   c. In Toolbox, expand the **Standard** group, and then double-click the **Button** control.
4. Change the value for the **ID** attribute from **Button1** to **FilterButton**.

   ```
   ID="FilterButton"
   ```

   - In the ImportCountries.aspx window, change the value of the **Button** control **ID** attribute from **Button1** to **FilterButton**.

     ```
     ID="FilterButton"
     ```

5. Change the **Text** property of the **Button** control from **Button** to **Filter Countries**.

```
Text="Filter Countries"
```

   - In the ImportCountries.aspx window, change the value of the **Button** control **Text** property from **Button** to **Filter Countries**.

```
Text="Filter Countries"
```

6. Open the **ImportCountries** Web Form in Design view, and then select the **FilterButton** control.

   - In the ImportCountries.aspx window, click **Design**, and then click the **FilterButton** control.

7. Add the default code for the **Click** event of the **FilterButton** control.

   - In the Properties window, click the **Events** button, and then double-click the text area of the **Click** event.

8. Save the **ImportCountries** code file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx.vb**, or press CTRL+S.

9. View the ImportCountries.aspx in Source view.

   a. In the CustomerManagement – Microsoft Visual Studio window, click **ImportCountries.aspx**.

   b. In the ImportCountries.aspx window, click **Source**.

10. Add a **div** element with a **class** attribute value of **importResult** to the **Content** control, between the **GridView** control and the **div** element with a **class** attribute value of **importCountriesHeader**.

    - Place the cursor before the opening tag of the **GridView** control,

```
<asp:GridView ID="CountriesGridView" runat="server"
AutoGenerateColumns="False"
```

      and then type the following markup.

```
<div class="importResult">

</div>
```

11. Add a **Label** control to the **div** element.
    a. In the ImportCountries.aspx window, place the cursor between the opening and closing **div** tags with a **class** attribute value of **importResult**.
    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
    c. In Toolbox, expand the **Standard** group, and then double-click the **Label** control.
12. Change the value for the **ID** attribute from **Label1** to **ImportResultLabel**.

```
ID="ImportResultLabel"
```

- In the ImportCountries.aspx window, change the value of the **ID** attribute of the **Label** control from **Label1** to **ImportResultLabel**.

```
ID="ImportResultLabel"
```

13. Remove the **Text** attribute and value.

```
Text="Label"
```

- In the ImportCountries.aspx window, delete the **Text** attribute of the **Label** and associated value.

```
Text="Label"
```

14. Disable view state for the **Label** control.

```
EnableViewState="false"
```

- In the ImportCountries.aspx window, add the **EnableViewState** attribute with a value of **false**.

```
EnableViewState="false"
```

15. Format and save the **ImportCountries** Web Form.
    a. In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press **CTRL+K**, and then press CTRL+D.
    b. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx**, or press CTRL+S.

▶ Task 2: Update sitemap to enable country import

1. Open the **web.sitemap** file.
   - In Solution Explorer, double-click **web.sitemap**.
2. Append new siteMapNode element to the **Countries siteMapNode**.

```
<siteMapNode title="Import" description="Import countries"
url="~/ImportCountries.aspx" />
```

   - Add the following markup to the **siteMapNode** element with a **title** attribute value of **Countries**.

```
<siteMapNode title="Import" description="Import countries"
url="~/ImportCountries.aspx" />
```

3. Save and close the **web.sitemap** file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**, or press CTRL+S.
   b. In the Web.sitemap window, click the **Close** button.

▶ Task 3: Load the XML data file manually

1. Create a private method named **loadCountries**, to load the XML data to the **ImportCountries** class.

> **Note:** The **loadCountries** method returns an **XElement**, and takes a string parameter named **fileName**.

```
Private Function loadCountries(ByVal fileName As String) As
XElement
End Function
```

   a. In the CustomerManagement – Microsoft Visual Studio window, click **ImportCountries.aspx.vb**.

b. In the ImportCountries.aspx.vb window, type the following code in the **ImportCountries** class.

```
Private Function loadCountries(ByVal fileName As String) As
XElement
End Function
```

2. Import the **System.Xml.Linq** namespace.

```
Imports System.Xml.Linq
```

- In the ImportCountries.aspx.vb window, type the following code at the top of the code file.

```
Imports System.Xml.Linq
```

3. Add the code to the **loadCountries** method to load the content of the file by using the **XElement.Load** method, and then return to the caller of the method.

- In the ImportCountries.aspx.vb window, add the following code to the **loadCountries** method.

```
return XElement.Load(Server.MapPath(fileName))
```

4. In the **Click** event handler for the **FilterButton** control, call the **loadCountries** method, passing Countries.xml for the filename, and save the returned value in an **XElement** variable named **countries**.

```
Dim countries As XElement = loadCountries("Countries.xml")
```

- In the ImportCountries.aspx.vb window, add the following code to the **Click** event handler of the **FilterButton** control.

```
Dim countries As XElement = loadCountries("Countries.xml")
```

5. Create a shared private member variable of type **IEnumerable(Of XElement)** named **countriesWithPhNoFormat**, for holding the filtered countries. Initialize to **Nothing**, in the **ImportCountries** class.

```
Private Shared countriesWithPhNoFormat As IEnumerable(Of XElement)
= Nothing
```

- In the ImportCountries.aspx.vb window, add the following code to the **ImportCountries** class.

```
Private Shared countriesWithPhNoFormat As IEnumerable(Of
XElement) = Nothing
```

6. Create a new private method named **filterCountries**, for filtering the XML data in the **ImportCountries** class.

> **Note:** The **filterCountries** method returns an **IEnumerable(XElement)**, and takes an **XElement** parameter named **countries**.

```
Private Function filterCountries(ByVal countries As XElement) As
IEnumerable(Of XElement)
End Function
```

- In the ImportCountries.aspx.vb window, add the following code to the **ImportCountries** class.

```
Private Function filterCountries(ByVal countries As XElement)
As IEnumerable(Of XElement)
End Function
```

7. Add a LINQ query to the **filterCountries** method that searches the child elements of the root node in the passed countries variable, and returns the result to the caller of the method.

```
Return _
    From c In countries.Elements()
    Select c
```

- In the ImportCountries.aspx.vb window, type the following code in the **filterCountries** method.

```
Return _
    From c In countries.Elements()
    Select c
```

8. Add filtering to the LINQ query, selecting the **Country** elements for which a value for the **PhoneNoFormat** attribute has been specified.

```
Where Not String.IsNullOrEmpty(c.Attribute("PhoneNoFormat"))
```

   - In the ImportCountries.aspx.vb window, add the following code to the LINQ query, between the **From** and **Select** clauses in the **filterCountries** method.

```
Where Not String.IsNullOrEmpty(c.Attribute("PhoneNoFormat"))
```

9. Call the **filterCountries** method, pass the local countries variable, and then save the returned value in the private member variable named **countriesWithPhNoFormat**, in the **Click** event handler of the **FilterButton** control.

```
countriesWithPhNoFormat = filterCountries(countries)
```

   - In the ImportCountries.aspx.vb window, append the following code to the **Click** event handler of the **FilterButton** control.

```
countriesWithPhNoFormat = filterCountries(countries)
```

10. Save the **ImportCountries** code file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx.vb**, or press CTRL+S.

▶ Task 4: Show the filtered countries by using the GridView control

1. Create a private method named **buildXmlString** for building an XML string from the filtered XML data in the **ImportCountries** class.

> **Note:** The **buildXmlString** method returns a string and takes an **IEnumerable(XElement)** parameter named **countriesWithPhNoFormat**.

```
Private Function buildXmlString(ByVal countriesWithPhNoFormat As
IEnumerable(Of XElement)) As String
End Function
```

- In the ImportCountries.aspx.vb window, type the following code in the **ImportCountries** class.

```
Private Function buildXmlString(ByVal countriesWithPhNoFormat
As IEnumerable(Of XElement)) As String
End Function
```

2. Return the value of the passed **countriesWithPhNoFormat**, wrapped in a root node named **Countries**, in the **buildXmlString** method.

```
return "<Countries>" & countriesWithPhNoFormat & "</Countries>"
```

- In the ImportCountries.aspx.vb window, type the following code in the **buildXmlString** method.

```
return "<Countries>" & countriesWithPhNoFormat &
"</Countries>"
```

3. Call the **IEnumerable(XElement).Select** method on the **countriesWithPhNoFormat** variable to convert each country into a string.

```
countriesWithPhNoFormat.Select()
```

- In the ImportCountries.aspx.vb window, modify the **countriesWithPhNoFormat** variable in the return statement as **countriesWithPhNoFormat.Select()**.

```
countriesWithPhNoFormat.Select()
```

4. Convert all the values in the **countriesWithPhNoFormat** variable to strings by using the **ToString** method in an anonymous Lambda function, placed in the call to the **Select** method.

```
countriesWithPhNoFormat.Select(Function(x) x.ToString())
```

- In the ImportCountries.aspx.vb window, modify the **countriesWithPhNoFormat** variable in the **Return** statement as **countriesWithPhNoFormat.Select(Function(x) x.ToString())**.

```
countriesWithPhNoFormat.Select(Function(x) x.ToString())
```

5. Convert the current output from the **Select** method to an array by using the **ToArray** method.

```
countriesWithPhNoFormat.Select(Function(x) x.ToString()).ToArray()
```

- In the ImportCountries.aspx.vb window, modify the **countriesWithPhNoFormat** variable in the **Return** statement as **countriesWithPhNoFormat.Select(Function(x) x.ToString()).ToArray()**.

```
countriesWithPhNoFormat.Select(Function(x)
x.ToString()).ToArray()
```

6. Join the current output from the **ToArray** method by using the **String.Join** method, specifying an empty string separator.

```
String.Join("", countriesWithPhNoFormat.Select(Function(x)
x.ToString()).ToArray())
```

- In the ImportCountries.aspx.vb window, modify the **countriesWithPhNoFormat** variable in the **Return** statement as **String.Join("", countriesWithPhNoFormat.Select(Function(x) x.ToString()).ToArray())**.

```
String.Join("", countriesWithPhNoFormat.Select(Function(x)
x.ToString()).ToArray())
```

7. Assign the return value of the **buildXmlString** method to the **Data** property of the **CountriesXmlDataSource** control, passing the **countriesWithPhNoFormat** private member variable to the **buildXmlString** method.

```
CountriesXmlDataSource.Data =
buildXmlString(countriesWithPhNoFormat)
```

- In the ImportCountries.aspx.vb window, append the following code to the **Click** event handler of the **FilterButton** control.

```
CountriesXmlDataSource.Data =
buildXmlString(countriesWithPhNoFormat)
```

8. Reset the **DataFile** property of the **CountriesXmlDataSource** control, ensuring that the **Data** property is used when rendering.

```
CountriesXmlDataSource.DataFile = ""
```

   - In the ImportCountries.aspx.vb window, append the following code to the **Click** event handler of the **FilterButton** control.

```
CountriesXmlDataSource.DataFile = ""
```

9. Build the **ImportCountries** Web Form, and fix any errors.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.
10. View the **ImportCountries** Web Form in the browser.
    - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View in Browser**.

> **Note:** Notice that the **CountriesGridView** control displays all the countries.

11. Filter the countries from the **ImportCountries** Web Form.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Filter Countries** button.

> **Note:** Notice that the **CountriesGridView** control displays the three filtered countries. Note the names of the countries.

12. Close Internet Explorer.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 3: Saving Data by Using LINQ to Entities

▶ Task 1: Add a Button control to the Web Form

1. Open the **ImportCountries** Web Form in Source view.
   - In the CustomerManagement – Microsoft Visual Studio window, click **ImportCountries.aspx**.
2. Add a **Button** control next to the **FilterButton** control.
   a. In ImportCountries.aspx window, locate the **FilterButton** control element, place the cursor at the end of the following markup, and then press ENTER.

```
<asp:Button ID="FilterButton" runat="server" Text="Filter
Countries" />
```

   b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
   c. In Toolbox, under **Standard**, double-click the **Button** control.
3. Change the value for the **ID** property from **Button1** to **SaveButton**.

```
ID="SaveButton"
```

4. Change the value for the **Text** property from **Button** to **Save Countries**.

```
Text="Save Countries"
```

5. Open the **ImportCountries** Web Form in Design view, and double-click the **SaveButton** control.
   - In the ImportCountries.aspx window, click **Design**, and then double-click the **SaveButton** control.
6. Save the **ImportCountries** code file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx.vb**, or press CTRL+S.

▶ Task 2: Add an ADO.NET Entity Data Model project item to the Web site project

1. Add an **ADO.NET Entity Data Model** project item named **CustomerManagement.edmx**.
    a. In Solution Explorer, right-click **D:\Labfiles\Starter\M9\VB \CustomerManagement**, and then click **Add New Item**.
    b. In the **Add New Item - D:\Labfiles\Starter\M9\VB \CustomerManagement** dialog box, in the middle pane, click **ADO.NET Entity data Model**.
    c. In the **Name** box, type **CustomerManagement.edmx**, and then click **Add**.
    d. In the Microsoft Visual Studio message box, click **Yes**.
    e. In the Entity Data Model Wizard, on the **Choose Model Contents** page, click **Generate from database**, and then click **Next**.
    f. On the **Choose Your Data Connection** page, on the **Which data connection should your application use to connect to the database?** list, click **CustomerManagementConnectionString (Settings)**, in the box below the **Save entity connection settings in Web.config as** check box, type **Entities**, and then click **Next**.
    g. On the **Choose Your Database Objects** page, on the **Which database objects do you want to include in your model?** list, expand **Tables**, click **Countries (dbo)**, click **Customers (dbo)**, ensure the **Pluralize or singularize generated object names** check box is selected, and then click **Finish**.
2. Save the **ADO.NET Entity Data Model** item.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save App_Code/CustomerManagement.edmx**, or press CTRL+S.
3. Close the **ADO.NET Entity Data Model** item.
    - In the App_Code/CustomerManagement.edmx window, click the **Close** button.

▶ Task 3: Create an ObjectContext object

1. Create a private method named **saveCountries** in the **ImportCountries** class, for saving the filtered XML data.

> **Note:** The **saveCountries** method does not return a value, but it takes an **IEnumerable(XElement)** parameter named **countriesWithPhNoFormat**.

```
Private Sub saveCountries(ByVal countriesWithPhNoFormat As
IEnumerable(Of XElement))
End Sub
```

- In the ImportCountries.aspx.vb window, type the following code in the **ImportCountries** class.

```
Private Sub saveCountries(ByVal countriesWithPhNoFormat As
IEnumerable(Of XElement))
End Sub
```

2. Add code to create and instantiate an **ObjectContext** of type **CustomerManagementModel.Entities**, named **cmObjectContext**, retrieving the **Entities** connection string from the **web.config** file by using the **System.Web.Configuration.WebConfigurationManager** class.

```
Dim cmObjectContext As New CustomerManagementModel.Entities(
    System.Web.Configuration.WebConfigurationManager.
    ConnectionStrings("Entities").ConnectionString)
```

- In the ImportCountries.aspx.vb window, type the following code in the **saveCountries** method.

```
Dim cmObjectContext As New CustomerManagementModel.Entities(
    System.Web.Configuration.WebConfigurationManager.
    ConnectionStrings("Entities").ConnectionString)
```

▶ Task 4: Insert the filtered XML data in the local ObjectContext data

1. Add code to loop through the **Country** elements in the passed **countriesWithPhNoFormat** variable, by using a **For Each** construct.

```
' Loop through the filtered countries
For Each country As XElement In countriesWithPhNoFormat
Next
```

   - In the ImportCountries.aspx.vb window, append the following code to the **saveCountries** method.

```
' Loop through the filtered countries
For Each country As XElement In countriesWithPhNoFormat
Next
```

2. Add code to insert each country in the **Countries** collection of the **cmObjectContext** object by using the **AddObject** method, and by passing the values from the filtered XML.

```
' Loop through the filtered countries
For Each country As XElement In countriesWithPhNoFormat
    ' Add the new country to the Countries collection
    cmObjectContext.Countries.AddObject(New
CustomerManagementModel.Country With
    {
        .ID = New Guid(country.Attribute("ID").Value),
        .Name = country.Attribute("Name").Value,
        .PhoneNoFormat = country.Attribute("PhoneNoFormat").Value,
        .DialingCountryCode =
country.Attribute("DialingCountryCode").Value,
        .InternationalDialingCode =
country.Attribute("InternationalDialingCode").Value,
        .InternetTLD = country.Attribute("InternetTLD").Value
    })
Next
```

- In the ImportCountries.aspx.vb window, add the following code to the **For Each** construct of the **saveCountries** method.

```
' Add the new country to the Countries collection
cmObjectContext.Countries.AddObject(New
CustomerManagementModel.Country With
{
    .ID = New Guid(country.Attribute("ID").Value),
    .Name = country.Attribute("Name").Value,
    .PhoneNoFormat = country.Attribute("PhoneNoFormat").Value,
    .DialingCountryCode =
country.Attribute("DialingCountryCode").Value,
    .InternationalDialingCode =
country.Attribute("InternationalDialingCode").Value,
    .InternetTLD = country.Attribute("InternetTLD").Value
})
```

▶ Task 5: Submit the local ObjectContext data to the database

1. Add a **Try Catch Finally** construct for saving the local modifications to the database. The **Catch** code must catch all exceptions.

```
' Save to database
Try
Catch ex As Exception
Finally
End Try
```

- In the ImportCountries.aspx.vb window, append the following code to the **saveCountries** method.

```
' Save to database
Try
Catch ex As Exception
Finally
End Try
```

2. Add a call to submit the local modifications to the database by using the **SaveChanges** method of the **ObjectContext** object in the **Try** part.

```
' Save to database
Try
    cmObjectContext.SaveChanges()
Catch ex As Exception
Finally
End Try
```

   - In the ImportCountries.aspx.vb window, type the following code in the **Try** part of the **saveCountries** method.

```
cmObjectContext.SaveChanges()
```

3. Display successful import result message by setting the **Text** property of the **ImportResultLabel** control to **Rows successfully exported to SQL Server table** in the **Try** part.

```
' Display success message
ImportResultLabel.Text = "Rows successfully exported to SQL Server
table."
```

   - In the ImportCountries.aspx.vb window, type the following code in the **Try** part of the **saveCountries** method.

```
' Display success message
ImportResultLabel.Text = "Rows successfully exported to SQL
Server table."
```

4. Add code to display a custom error message by setting the **Text** property of the **ImportResultLabel** control to **An error occurred exporting to SQL Server.
** in the **Catch** part. Append the value of the **Message** property of the caught **Exception**, and two additional line breaks **

. The text color must be set to red, by setting the **ForeColor property of the **Label** control to the value **System.Drawing.Color.Red**.

```
Catch ex as Exception
    ' Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" & ex.Message & "<br/><br/>"
    ImportResultLabel.ForeColor = System.Drawing.Color.Red
```

- In the ImportCountries.aspx.vb window, type the following code in the **Catch** part of the **saveCountries** method.

```
' Display error
ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" & ex.Message & "<br/><br/>"
ImportResultLabel.ForeColor = System.Drawing.Color.Red
```

5. Add a more specific **Catch** statement to the **Try Catch Finally** construct for catching exceptions that are thrown when errors occur during the actual update, by catching exceptions of type **System.Data.UpdateException**. The **Catch** code must be added before the existing catch.

```
' Save to database
Try
    cmObjectContext.SaveChanges()
    ' Display success message
    ImportResultLabel.Text = "Rows successfully exported to SQL
Server table."
Catch ex As System.Data.UpdateException
Catch ex As Exception
    ' Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" & ex.Message & "<br/><br/>"
    ImportResultLabel.ForeColor = System.Drawing.Color.Red
Finally
End Try
```

- In the ImportCountries.aspx.vb window, add the following code to the **Try Catch Finally** construct.

```
Catch ex As System.Data.UpdateException
```

6. Add code to display a custom error message by setting the **Text** property of the **ImportResultLabel** control to **An error occurred exporting to SQL Server.
One or more of the rows already exist in the table.

** in the new **Catch**. The text color must be set to red by setting the **ForeColor** property of the **Label** control to **System.Drawing.Color.Red**.

```
Catch ex As System.Data.UpdateException
    ' Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>One or more of the rows already exist in the
table.<br/><br/>"
    ImportResultLabel.ForeColor = System.Drawing.Color.Red
```

   - In the ImportCountries.aspx.vb window, type the following code in the new **Catch**.

```
' Display error
ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>One or more of the rows already exist in the
table.<br/><br/>"
ImportResultLabel.ForeColor = System.Drawing.Color.Red
```

7. Add code to dispose the **ObjectContext** object in the **Finally** part.

```
Finally
    cmObjectContext.Dispose()
```

   - In the ImportCountries.aspx.vb window, add the following code to the **finally** part of the **saveCountries** method.

```
cmObjectContext.Dispose()
```

8. Call the **saveCountries** method by passing the static **countriesWithPhNoFormat** private member variable from the **Click** event handler of the **SaveButton** control.

   - In the ImportCountries.aspx.vb window, append the following code to the **Click** event handler of the **SaveButton** control.

```
saveCountries(countriesWithPhNoFormat)
```

9. Build the **ImportCountries** Web Form, and fix any errors.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.

10. View the **ImportCountries** Web Form in the Solution Explorer browser.
    - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View in Browser**.

> **Note:** Notice that the **CountriesGridView** control displays the content of the XML data file.

11. Filter the countries from the **ImportCountries** Web Form.
    - In the Contoso Customer Management – Windows Internet Explorer window, click **Filter Countries**.
12. Save the imported countries in the database.
    a. In the Contoso Customer Management – Windows Internet Explorer window, click **Save Countries**.
    b. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

> **Note:** If you try to save the countries that already exist in the SQL database, you will receive an error message.

▶ Task 6: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3, In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 9

Lab Answer Key: Managing Data Access Tasks by Using LINQ (Visual C#)

Contents:

Lab: Managing Data Access Tasks by Using LINQ

(C#)

Exercise 1: Loading Data by Using the XmlDataSource Control

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the 10267A-GEN-DEV virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M9\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M9\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Create the ImportCountries Web Form

1. Add a Web Form named **ImportCountries**, which is based on the **Site.master** master page.
   a. In Solution Explorer, right-click **D:\Labfiles\Starter\M9\CS \CustomerManagement**, and then click **Add New Item**.
   b. In the **Add New Item – D:\Labfiles\Starter\M9\CS \CustomerManagement** dialog box, in the middle pane, click **Web Form**.

c. In the **Name** box, type **ImportCountries.aspx**, ensure that the **Place code in separate file** check box is selected, select the **Select master page** check box, and then click **Add**.

d. In the **Select a Master Page** dialog box, in the **Contents of folder** box, click **Site.master**, and then click **OK**.

2. Open the **ImportCountries** Web Form in the Design view.

   - In the ImportCountries.aspx window, click **Design**.

▶ Task 3: Add an XmlDataSource control to the Web Form

1. Add an existing XML file, **D:\Labfiles\Starter\M9\Countries.xml**, to the Web site.

   a. In Solution Explorer, right-click **D:\Labfiles\Starter\M9\CS\CustomerManagement**, and then click **Add Existing Item**.

   b. In the **Add Existing Item** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M9\Countries.xml**, and then click **Add**.

2. Add an **XmlDataSource** control to the **Content** control of the **ImportCountries** Web Form.

   a. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.

   b. In Toolbox, under **Data**, double-click **XmlDataSource** control.

3. Rename the **XmlDataSource** control that you just added to **CountriesXmlDataSource**.

   - In the Properties window of the **XmlDataSource** control, change the value of the **ID** property from **XmlDataSource1** to **CountriesXmlDataSource**, and then press ENTER.

4. Save the **ImportCountries** Web Form.

   - On the **File** menu, click **Save ImportCountries.aspx**, or press the CTRL+S keys.

▶ Task 4: Configure the XmlDataSource control

1. Open the **Configure Data Source** control by using the **Show Smart Tag** option.
   - In the ImportCountries.aspx window, right-click **CountriesXmlDataSource** control, click **Show Smart Tag**, and then click **Configure Data Source**.
2. Use the **Countries.xml** data file in the **XmlDataSource** control.
   - In the **Data file** box, type **~/Countries.xml**, and then click **OK**.
3. Save the **ImportCountries** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx**, or press CTRL+S.

▶ Task 5: Add a GridView control to the Web Form

1. Add a **GridView** control to the **Content** control of the **ImportCountries** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**, and then double-click **GridView** under **Data**.
2. Rename the **GridView** control as **CountriesGridView**.
   - In the Properties window of the **GridView** control, change the value of the **ID** attribute from **GridView1** to **CountriesGridView**, and then press ENTER.
3. Set the width of the **GridView** control to **100%**.
   - In the Properties window of the **GridView** control, set the value of the **Width** property to **100%**, and then press ENTER.
4. Bind the **CountriesGridView** control to the **CountriesXmlDataSource** control by using the Smart Tag.
   a. In the ImportCountries.aspx window, right-click **CountriesGridView**, and then click **Show Smart Tag**.
   b. In the **GridView Tasks** dialog box, in the **Choose Data Source** list, click **CountriesXmlDataSource**.

5. Save the **ImportCountries** Web Form.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx**, or press CTRL+S.
6. Build the Web Form and fix any errors.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.

> **Note:** Notice that the Build success message appears in the **Build** pane of the Output window.

7. View the ImportCountries Web Form in the browser.
    - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View in Browser**.

> **Note:** Notice that the **CountriesGridView** control displays the content of the XML data file.

8. Close Windows® Internet Explorer®.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Displaying Data by Using LINQ to XML

▶ Task 1: Add a Button control to the Web Form

1. Open the **ImportCountries** Web Form in the Source view.
   - In the ImportCountries.aspx window, click **Source**.
2. Add a **div** element with a class attribute value of **importCountriesHeader** to the **Content** control, above the **GridView** control.
   - Place the cursor before the opening tag of the **GridView** control,

     ```
     <asp:GridView ID="CountriesGridView" runat="server"
     AutoGenerateColumns="False"
     ```

     and then, type the following markup.

     ```
     <div class="importCountriesHeader">

     </div>
     ```
3. Add a **Button** control to the **div** element.
   a. In the ImportCountries.aspx window, place the cursor between the opening and closing **div** tags with a **class** attribute value of **importCountriesHeader**.
   b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
   c. In Toolbox, expand the **Standard** group, and then double-click the **Button** control.
4. Change the value for the **ID** attribute from **Button1** to **FilterButton**.

   ```
   ID="FilterButton"
   ```

   - In the ImportCountries.aspx window, change the value of the **Button** control **ID** attribute from **Button1** to **FilterButton**.

     ```
     ID="FilterButton"
     ```

5. Change the **Text** property of the **Button** control from **Button** to **Filter Countries**.

```
Text="Filter Countries"
```

   - In the ImportCountries.aspx window, change the value of the **Button** control **Text** property from **Button** to **Filter Countries**.

```
Text="Filter Countries"
```

6. Open the **ImportCountries** Web Form in Design view, and then select the **FilterButton** control.
   - In the ImportCountries.aspx window, click **Design**, and then click the **FilterButton** control.
7. Add the default code for the **Click** event of the **FilterButton** control.
   - In the Properties window, click the **Events** button, and then double-click the text area of the **Click** event.
8. Save the **ImportCountries** code file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx.cs**, or press CTRL+S.
9. View the ImportCountries.aspx in Source view.
   a. In the CustomerManagement – Microsoft Visual Studio window, click **ImportCountries.aspx**.
   b. In the ImportCountries.aspx window, click **Source**.
10. Add a **div** element with a class attribute value of **importResult** to the **Content** control, between the **GridView** control and the **div** element with a class attribute value of **importCountriesHeader**.
    - Place the cursor before the opening tag of the **GridView** control,

```
<asp:GridView ID="CountriesGridView" runat="server"
AutoGenerateColumns="False"
```

      and then type the following markup.

```
<div class="importResult">

</div>
```

11. Add a **Label** control to the **div** element.
    a. In the ImportCountries.aspx window, place the cursor between the opening and closing **div** tags with a **class** attribute value of **importResult**.
    b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
    c. In Toolbox, expand the **Standard** group, and then double-click the **Label** control.
12. Change the value for the **ID** attribute from **Label1** to **ImportResultLabel**.

```
ID="ImportResultLabel"
```

- In the ImportCountries.aspx window, change the value of the **ID** attribute of the **Label** control from **Label1** to **ImportResultLabel**.

```
ID="ImportResultLabel"
```

13. Remove the **Text** attribute and value.

```
Text="Label"
```

- In the ImportCountries.aspx window, delete the **Text** attribute of the **Label** and associated value.

```
Text="Label"
```

14. Disable view state for the **Label** control.

```
EnableViewState="false"
```

- In the ImportCountries.aspx window, add the **EnableViewState** attribute with a value of **false**.

```
EnableViewState="false"
```

15. Format and save the **ImportCountries** Web Form.
    a. In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, click **Format Document**, or press CTRL+K, and then press CTRL+D.
    b. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx**, or press CTRL+S.

▶ Task 2: Update sitemap to enable country import

1. Open the **web.sitemap** file.
   - In Solution Explorer, double-click **web.sitemap**.
2. Append new siteMapNode element to the **Countries siteMapNode**.

```
<siteMapNode title="Import" description="Import countries"
url="~/ImportCountries.aspx" />
```

   - Add the following markup to the **siteMapNode** element with a **title** attribute value of **Countries**.

```
<siteMapNode title="Import" description="Import countries"
url="~/ImportCountries.aspx" />
```

3. Save and close the **web.sitemap** file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**, or press CTRL+S.
   b. In the Web.sitemap window, click the **Close** button.

▶ Task 3: Load the XML data file manually

1. Create a private method named **loadCountries**, to load the XML data to the **ImportCountries** class.

> **Note:** The **loadCountries** method returns an **XElement**, and takes a string parameter named **fileName**.

```
private XElement loadCountries(string fileName)
{
}
```

   a. In the CustomerManagement – Microsoft Visual Studio window, click **ImportCountries.aspx.cs**.

b. In the ImportCountries.aspx.cs window, type the following code in the **ImportCountries** class.

```
private XElement loadCountries(string fileName)
{
}
```

2. Import the **System.Xml.Linq** namespace.

```
using System.Xml.Linq;
```

- In the ImportCountries.aspx.cs window, type the following code at the top of the code file.

```
using System.Xml.Linq;
```

3. Add the code to the **loadCountries** method to load the content of the file by using the **XElement.Load** method, and then return to the caller of the method.

- In the ImportCountries.aspx.cs window, add the following code to the **loadCountries** method.

```
return XElement.Load(Server.MapPath(fileName));
```

4. In the **Click** event handler for the **FilterButton** control, call the **loadCountries** method, passing **Countries.xml** for the filename, and save the returned value in an **XElement** variable named **countries**.

```
XElement countries = loadCountries("Countries.xml");
```

- In the ImportCountries.aspx.cs window, add the following code to the **Click** event handler of the **FilterButton** control.

```
XElement countries = loadCountries("Countries.xml");
```

5. Create a static private member variable of type **IEnumerable(XElement)** named **countriesWithPhNoFormat**, for holding the filtered countries. Initialize to **null**, in the **ImportCountries** class.

```
static private IEnumerable<XElement> countriesWithPhNoFormat =
null;
```

- In the ImportCountries.aspx.cs window, add the following code to the **ImportCountries** class.

```
static private IEnumerable<XElement> countriesWithPhNoFormat =
null;
```

6. Create a new private method named **filterCountries**, for filtering the XML data in the **ImportCountries** class.

> **Note:** The **filterCountries** method returns an **IEnumerable(XElement)**, and takes an **XElement** parameter named **countries**.

```
private IEnumerable<XElement> filterCountries(XElement countries)
{
}
```

- In the ImportCountries.aspx.cs window, add the following code to the **ImportCountries** class.

```
private IEnumerable<XElement> filterCountries(XElement
countries)
{
}
```

7. Add a LINQ query to the **filterCountries** method that searches the child elements of the root node in the passed countries variable, and returns the result to the caller of the method.

```
return
    from c in countries.Elements()
    select c;
```

- In the ImportCountries.aspx.cs window, type the following code in the **filterCountries** method.

```
return
    from c in countries.Elements()
    select c;
```

8. Add filtering to the LINQ query, selecting the **Country** elements for which a value for the **PhoneNoFormat** attribute has been specified.

```
where c.Attribute("PhoneNoFormat") != null &&
    (string) c.Attribute("PhoneNoFormat").Value != ""
```

   - In the ImportCountries.aspx.cs window, add the following code to the LINQ query, between the **from** and **select** clauses in the **filterCountries** method.

```
where c.Attribute("PhoneNoFormat") != null &&
    (string) c.Attribute("PhoneNoFormat").Value != ""
```

9. Call the **filterCountries** method, pass the local countries variable, and then save the returned value in the private member variable named **countriesWithPhNoFormat**, in the **Click** event handler of the **FilterButton** control.

```
countriesWithPhNoFormat = filterCountries(countries);
```

   - In the ImportCountries.aspx.cs window, append the following code to the **Click** event handler of the **FilterButton** control.

```
countriesWithPhNoFormat = filterCountries(countries);
```

10. Save the **ImportCountries** code file.

   - In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx.cs**, or press CTRL+S.

▶ Task 4: Show the filtered countries by using the GridView control

1. Create a private method named **buildXmlString** for building an XML string from the filtered XML data in the **ImportCountries** class.

> **Note:** The **buildXmlString** method returns a string and takes an **IEnumerable(XElement)** parameter named **countriesWithPhNoFormat**.

```
private string buildXmlString(IEnumerable<XElement>
countriesWithPhNoFormat)
{
}
```

- In the ImportCountries.aspx.cs window, type the following code in the **ImportCountries** class.

```
private string buildXmlString(IEnumerable<XElement>
countriesWithPhNoFormat)
{
}
```

2. Return the value of the passed **countriesWithPhNoFormat**, wrapped in a root node named **Countries**, in the **buildXmlString** method.

```
return "<Countries>" + countriesWithPhNoFormat + "</Countries>";
```

- In the ImportCountries.aspx.cs window, type the following code in the **buildXmlString** method:

```
return "<Countries>" + countriesWithPhNoFormat +
"</Countries>";
```

3. Call the **IEnumerable(XElement).Select** method on the **countriesWithPhNoFormat** variable to convert each country into a string.

```
countriesWithPhNoFormat.Select()
```

- In the ImportCountries.aspx.cs window, modify the **countriesWithPhNoFormat** variable in the return statement as **countriesWithPhNoFormat.Select()**.

```
countriesWithPhNoFormat.Select()
```

4. Convert all the values in the **countriesWithPhNoFormat** variable to strings by using the **ToString** method in an anonymous Lambda function, placed in the call to the **Select** method.

```
countriesWithPhNoFormat.Select(x => x.ToString())
```

   - In the ImportCountries.aspx.cs window, modify the **countriesWithPhNoFormat** variable in the **return** statement as **countriesWithPhNoFormat.Select(x => x.ToString())**.

```
countriesWithPhNoFormat.Select(x => x.ToString())
```

5. Convert the current output from the **Select** method to an array by using the **ToArray** method.

```
countriesWithPhNoFormat.Select(x => x.ToString()).ToArray()
```

   - In the ImportCountries.aspx.cs window, modify the **countriesWithPhNoFormat** variable in the **return** statement as **countriesWithPhNoFormat.Select(x => x.ToString()).ToArray()**.

6. Join the current output from the **ToArray** method by using the **String.Join** method, specifying an empty string separator.

```
string.Join("", countriesWithPhNoFormat.Select(x =>
x.ToString()).ToArray())
```

   - In the ImportCountries.aspx.cs window, modify the **countriesWithPhNoFormat** variable in the **return** statement as **string.Join("", countriesWithPhNoFormat.Select(x => x.ToString()).ToArray())**.

```
string.Join("", countriesWithPhNoFormat.Select(x =>
x.ToString()).ToArray())
```

7. Assign the return value of the **buildXmlString** method to the **Data** property of the **CountriesXmlDataSource** control, passing the **countriesWithPhNoFormat** private member variable to the **buildXmlString** method.

```
CountriesXmlDataSource.Data =
buildXmlString(countriesWithPhNoFormat);
```

- In the ImportCountries.aspx.cs window, append the following code to the **Click** event handler of the **FilterButton** control.

```
CountriesXmlDataSource.Data =
buildXmlString(countriesWithPhNoFormat);
```

8. Reset the **DataFile** property of the **CountriesXmlDataSource** control, ensuring that the **Data** property is used when rendering.

```
CountriesXmlDataSource.DataFile = "";
```

- In the ImportCountries.aspx.cs window, append the following code to the **Click** event handler of the **FilterButton** control.

```
CountriesXmlDataSource.DataFile = "";
```

9. Build the **ImportCountries** Web Form, and fix any errors.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.
10. View the **ImportCountries** Web Form in the browser.
    - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View in Browser**.

Note: Notice that the **CountriesGridView** control displays all the countries.

11. Filter the countries from the **ImportCountries** Web Form.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Filter Countries** button.

Note: Notice that the **CountriesGridView** control displays the three filtered countries. Note the names of the countries.

12. Close Internet Explorer.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 3: Saving Data by Using LINQ to Entities

▶ Task 1: Add a Button control to the Web Form

1. Open the **ImportCountries** Web Form in Source view.
   - In the CustomerManagement – Microsoft Visual Studio window, click **ImportCountries.aspx**.
2. Add a **Button** control next to the **FilterButton** control.
   a. In ImportCountries.aspx window, locate the **FilterButton** control element, place the cursor at the end of the following markup, and then press ENTER.

```
<asp:Button ID="FilterButton" runat="server" Text="Filter
Countries"
    onclick="FilterButton_Click" />
```

   b. In the CustomerManagement – Microsoft Visual Studio window, point to **Toolbox**.
   c. In Toolbox, under **Standard**, double-click the **Button** control.
3. Change the value for the **ID** property from **Button1** to **SaveButton**.

```
ID="SaveButton"
```

4. Change the value for the **Text** property from **Button** to **Save Countries**.

```
Text="Save Countries"
```

5. Open the **ImportCountries** Web Form in Design view, and double-click the **SaveButton** control.
   - In the ImportCountries.aspx window, click **Design**, and then double-click the **SaveButton** control.
6. Save the **ImportCountries** code file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx.cs**, or press CTRL+S.

▶ Task 2: Add an ADO.NET Entity Data Model project item to the Web site project

1. Add an **ADO.NET Entity Data Model** project item named **CustomerManagement.edmx**.
   a. In Solution Explorer, right-click **D:\Labfiles\Starter\M9\CS \CustomerManagement**, and then click **Add New Item**.
   b. In the **Add New Item - D:\Labfiles\Starter\M9\CS \CustomerManagement** dialog box, in the middle pane, click **ADO.NET Entity data Model**.
   c. In the **Name** box, type **CustomerManagement.edmx**, and then click **Add**.
   d. In the Microsoft Visual Studio message box, click **Yes**.
2. In the Entity Data Model Wizard, on the **Choose Model Contents** page, click **Generate from database**, and then click **Next**.
3. On the **Choose Your Data Connection** page, on the **Which data connection should your application use to connect to the database?** list, click **CustomerManagementConnectionString (Settings)**, in the box below the **Save entity connection settings in Web.config as** check box, type **Entities**, and then click **Next**.
4. On the **Choose Your Database Objects** page, on the **Which database objects do you want to include in your model?** list, expand **Tables**, click **Countries (dbo)**, click **Customers (dbo)**, ensure the **Pluralize or singularize generated object names** check box is selected, and then click **Finish**.
5. Save the **ADO.NET Entity Data Model** item.
   - In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save App_Code/CustomerManagement.edmx**, or press CTRL+S.
6. Close the **ADO.NET Entity Data Model** item.
   - In the App_Code/CustomerManagement.edmx window, click the **Close** button.

▶ Task 3: Create an ObjectContext object

1. Create a private method named **saveCountries** in the **ImportCountries** class, for saving the filtered XML data.

> **Note:** The **saveCountries** method does not return a value, but it takes an **IEnumerable(XElement)** parameter named **countriesWithPhNoFormat**.

```
private void saveCountries(IEnumerable<XElement>
countriesWithPhNoFormat)
{
}
```

- In the ImportCountries.aspx.cs window, type the following code in the **ImportCountries** class.

```
private void saveCountries(IEnumerable<XElement>
countriesWithPhNoFormat)
{
}
```

2. Add code to create and instantiate an **ObjectContext** of type **CustomerManagementModel.Entities**, named **cmObjectContext**, retrieving the **Entities** connection string from the **web.config** file by using the **System.Web.Configuration.WebConfigurationManager** class.

```
CustomerManagementModel.Entities cmObjectContext =
    new CustomerManagementModel.Entities(
        System.Web.Configuration.WebConfigurationManager.
        ConnectionStrings["Entities"].ConnectionString);
```

- In the ImportCountries.aspx.cs window, type the following code in the **saveCountries** method.

```
CustomerManagementModel.Entities cmObjectContext =
    new CustomerManagementModel.Entities(
        System.Web.Configuration.WebConfigurationManager.
        ConnectionStrings["Entities"].ConnectionString);
```

▶ Task 4: Insert the filtered XML data in the local ObjectContext data

1. Add code to loop through the **Country** elements in the passed **countriesWithPhNoFormat** variable, by using a **foreach** construct.

```
// Loop through the filtered countries
foreach (XElement country in countriesWithPhNoFormat)
{
}
```

   - In the ImportCountries.aspx.cs window, append the following code to the **saveCountries** method.

```
// Loop through the filtered countries
foreach (XElement country in countriesWithPhNoFormat)
{
}
```

2. Add code to insert each country in the **Countries** collection of the **cmObjectContext** object by using the **AddObject** method, and by passing the values from the filtered XML.

```
// Add the new country to the Countries collection
cmObjectContext.Countries.AddObject(new
CustomerManagementModel.Country
{
    ID = new Guid(country.Attribute("ID").Value),
    Name = country.Attribute("Name").Value,
    PhoneNoFormat = country.Attribute("PhoneNoFormat").Value,
    DialingCountryCode =
country.Attribute("DialingCountryCode").Value,
    InternationalDialingCode =
country.Attribute("InternationalDialingCode").Value,
    InternetTLD = country.Attribute("InternetTLD").Value
});
```

- In the ImportCountries.aspx.cs window, add the following code to the **foreach** construct of the **saveCountries** method.

```
// Add the new country to the Countries collection
cmObjectContext.Countries.AddObject(new
CustomerManagementModel.Country
{
    ID = new Guid(country.Attribute("ID").Value),
    Name = country.Attribute("Name").Value,
    PhoneNoFormat = country.Attribute("PhoneNoFormat").Value,
    DialingCountryCode =
country.Attribute("DialingCountryCode").Value,
    InternationalDialingCode =
country.Attribute("InternationalDialingCode").Value,
    InternetTLD = country.Attribute("InternetTLD").Value
});
```

▶ Task 5: Submit the local ObjectContext data to the database

1. Add a **try catch finally** construct for saving the local modifications to the database. The **catch** code must catch all exceptions.

```
// Save to database
try
{
}
catch (Exception ex)
{
}
finally
{
}
```

- In the ImportCountries.aspx.cs window, append the following code to the **saveCountries** method.

```
// Save to database
try
{
}
catch (Exception ex)
{
}
finally
{
}
```

2. Add a call to submit the local modifications to the database by using the **SaveChanges** method of the **ObjectContext** object in the **try** part.

```
// Save to database
try
{
    cmObjectContext.SaveChanges();
}
catch (Exception ex)
{
}
finally
{
}
```

- In the ImportCountries.aspx.cs window, type the following code in the **try** part of the **saveCountries** method.

```
cmObjectContext.SaveChanges();
```

3. Display successful import result message, by setting the **Text** property of the **ImportResultLabel** control to **Rows successfully exported to SQL Server table** in the **try** part.

```
// Display success message
ImportResultLabel.Text = "Rows successfully exported to SQL Server
table.";
```

- In the ImportCountries.aspx.cs window, type the following code in the **try** part of the **saveCountries** method.

```
// Display success message
ImportResultLabel.Text = "Rows successfully exported to SQL
Server table.";
```

4. Add code to display a custom error message by setting the **Text** property of the **ImportResultLabel** control to **An error occurred exporting to SQL Server.
** in the **catch** part. Append the value of the **Message** property of the caught **Exception**, and two additional line breaks **

. The text color must be set to red, by setting the **ForeColor property of the **Label** control to the value **System.Drawing.Color.Red**.

```
catch (Exception ex)
{
    // Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" + ex.Message + "<br/><br/>";
    ImportResultLabel.ForeColor = System.Drawing.Color.Red;
}
```

- In the ImportCountries.aspx.cs window, type the following code in the **catch** part of the **saveCountries** method.

```
// Display error
ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" + ex.Message + "<br/><br/>";
ImportResultLabel.ForeColor = System.Drawing.Color.Red;
```

5. Add a more specific **catch** to the **try catch finally** construct for catching exceptions that are thrown when errors occur during the actual update, by catching exceptions of type **System.Data.UpdateException**. The **catch** code must be added before the existing catch.

```
// Save to database
try
{
    cmObjectContext.SaveChanges();

    // Display success message
    ImportResultLabel.Text = "Rows successfully exported to SQL
Server table.";
}
catch (System.Data.UpdateException)
{
}
catch (Exception ex)
{
    // Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>" + ex.Message + "<br/><br/>";
    ImportResultLabel.ForeColor = System.Drawing.Color.Red;
}
finally
{
}
```

- In the ImportCountries.aspx.cs window, add the following code to the **try catch finally** construct.

```
catch (System.Data.UpdateException)
{
}
```

6. Add code to display a custom error message by setting the **Text** property of the **ImportResultLabel** control to **An error occurred exporting to SQL Server.
One or more of the rows already exist in the table.

** in the new **catch**. The text color must be set to red, by setting the **ForeColor** property of the **Label** control to **System.Drawing.Color.Red**.

```
catch (System.Data.UpdateException)
{
    // Display error
    ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>One or more of the rows already exist in the
table.<br/><br/>";
    ImportResultLabel.ForeColor = System.Drawing.Color.Red;
}
```

   - In the ImportCountries.aspx.cs window, type the following code in the new catch.

```
// Display error
ImportResultLabel.Text = "An error occurred exporting to SQL
Server.<br/>One or more of the rows already exist in the
table.<br/><br/>";
ImportResultLabel.ForeColor = System.Drawing.Color.Red;
```

7. Add code to dispose the **ObjectContext** object in the **finally** part.

```
finally
{
    cmObjectContext.Dispose();
}
```

   - In the ImportCountries.aspx.cs window, add the following code to the **finally** part of the **saveCountries** method.

```
cmObjectContext.Dispose();
```

8. Call the **saveCountries** method by passing the static **countriesWithPhNoFormat** private member variable from the **Click** event handler of the **SaveButton** control.

   - In the ImportCountries.aspx.cs window, append the following code to the **Click** event handler of the **SaveButton** control.

```
saveCountries(countriesWithPhNoFormat);
```

9. Build the ImportCountries Web Form, and fix any errors.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Page**.
10. View the **ImportCountries** Web Form in the Solution Explorer browser.
    - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View in Browser**.

> **Note:** Notice that the **CountriesGridView** control displays the content of the XML data file.

11. Filter the countries from the **ImportCountries** Web Form.
    - In the Contoso Customer Management – Windows Internet Explorer window, click **Filter Countries**.
12. Save the imported countries in the database.
    a. In the Contoso Customer Management – Windows Internet Explorer window, click **Save Countries**.
    b. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

> **Note:** If you try to save the countries that already exist in the SQL database, you will receive an error message.

▶ Task 6: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 10

Lab Answer Key: Managing Data by Using Microsoft® ASP.NET Dynamic Data (Visual Basic)

Contents:

Lab: Managing Data by Using ASP.NET Dynamic Data

(Visual Basic)

Exercise 1: Adding Dynamic Data to an Existing Web Site

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the CustomerManagement solution from the D:\Labfiles\Starter \M10\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M10\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Create a sample Dynamic Data Web site

1. Open a second instance of Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.

2. Create a new ASP.NET Dynamic Data Web site in the D:\Labfiles\Starter \M10\VB\SampleDDWebSite folder.
    a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **New Web Site**.
    b. In the **New Web Site** dialog box, in the left pane, ensure that **Visual Basic** is selected, in the middle pane, click **ASP.NET Dynamic Data Entities Web Site**, in the **Web location** list, click **File System**, in the adjacent text box, type **D:\Labfiles\Starter\M10\VB\SampleDDWebSite**, and then click **OK**.
3. Close the second instance of Visual Studio 2010.
    - In the SampleDDWebSite – Microsoft Visual Studio window, click the **Close** button, or click the ALT+F4 keys.

▶ Task 3: Add the dynamic data default files and folders

1. Add the **DynamicData** folder and default dynamic data content from the SampleDDWebSite Web site to the **CustomerManagement** Web site by copying the DynamicData folder from the path, **D:\Labfiles\Starter\M10\VB \SampleDDWebSite\DynamicData**. Use Windows Explorer for this purpose.
    a. On the **Start** menu, click **Run**.
    b. In the **Run** dialog box, in the **Open** box, type **D:\Labfiles\Starter\M10 \VB\SampleDDWebSite**, and then click **OK**.
    c. In the Windows® Explorer window, right-click **DynamicData**, and then click **Copy**.
    d. Right-click **CustomerManagement**, and then click **Paste**.
    e. Close the Windows Explorer window.
2. In Solution Explorer, refresh the Web site content to ensure the addition of the **DynamicData** folder.
    - In Solution Explorer, right-click **D:\Labfiles\Starter\M10\VB \CustomerManagement**, and then click **Refresh Folder**.

▶ Task 4: Add the ScriptManager control to the master page

1. Open the **Site.master** master page in Source view.
   - In Solution Explorer, right-click **Site.master**, and then click **View Markup**.
2. Add a **ScriptManager** control before the **div** element with a value of **content** for the class attribute, by adding an **asp:ScriptManager** element with an **ID** attribute value of **MainScriptManager**.

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

   - In the Site.master window, type the following markup before the **div** element with a value of **content** for the class attribute.

     ```
     <asp:ScriptManager ID="MainScriptManager" runat="server" />
     ```
3. Save the **Site.master** master page.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**, or press CTRL+S.
4. Close the **Site.master** master page.
   - In the Site.master window, click the **Close** button, or press CTRL+F4.

▶ Task 5: Update the dynamic data page templates to use an existing master page

1. Open the **Find and Replace** dialog box.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, point to **Find and Replace**, and then click **Replace in Files**, or press CTRL+SHIFT+H.
2. Replace the **ContentPlaceHolder1** name that is referenced in the Dynamic Data Page Templates, with the name of the **ContentPlaceHolder** control that is used in the existing master page, **MainContentPlaceHolder**.
   a. In the **Find and Replace** dialog box, in the **Find what** box, type **ContentPlaceHolder1**, in the **Replace with** box, type **MainContentPlaceHolder**, and then click **Replace All**.
   b. In the **Replace All** dialog box, select the **Replace All will open all files with changes for editing** check box, and then click **Yes**.

c. In the **Microsoft Visual Studio** dialog box, click **OK**, and in the **Find and Replace** dialog box, click the **Close** button.

3. In all of the opened and modified Dynamic Data page templates—**Details.aspx**, **Edit.aspx**, **Insert.aspx**, **List.aspx**, and **ListDetails.aspx**—remove the **Content** control with an **ID** attribute value of **headContent**.

   - In the Dynamic Data page template, locate and select the **Content** control with an **ID** attribute value of **headContent**.

     ```
     <asp:Content ID="headContent" ContentPlaceHolderID="head"
     Runat="Server">
     </asp:Content>
     ```

   - On the **Edit** menu, click **Delete**, or press the DELETE key, to delete the selected markup.

4. Save and close all modified files.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**, or press CTRL+SHIFT+S.

   b. Click the **Close** button for all open Web Forms.

▶ Task 6: Copy dynamic data styles

1. Copy the content of the **D:\Labfiles\Starter\M10\VB\SampleDDWebSite\Site.css** stylesheet.

   a. On the **Start** menu, click **Run**.

   b. In the **Run** dialog box, in the **Open** box, type **D:\Labfiles\Starter\M10\VB\SampleDDWebSite**, and then click **OK**.

   c. In the Windows Explorer window, double-click **Site.css**.

   d. In the Site.css window, select all by pressing CTRL+A, right-click the selection, and then click **Copy**.

2. Close the Windows Explorer window.

3. Add the copied content to the existing Styles\Site.css stylesheet.
   a. In Solution Explorer, expand **Styles**, and double-click **Site.css**.
   b. In the Styles/Site.css window, press CTRL+END, press ENTER, and then press SHIFT+INSERT.
4. Save and close the Site.css file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Styles/Site.css**, or press CTRL +S.
   b. In the **Styles/Site.css** window, click the **Close** button.

Exercise 2: Registering Entity Framework with Dynamic Data

▶ Task 1: Add a Global Application class file

- Add the Global Application class project item to the **CustomerManagement** Web site.

  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M10\VB \CustomerManagement**, and then click **Add New Item**.

  b. In the **Add New Item - D:\Labfiles\Starter\M10\VB \CustomerManagement** dialog box, in the middle pane, click **Global Application Class**, and then click **Add**.

▶ Task 2: Import the required namespaces

- Add an **Import** directive to the Global Application Class with the **Namespace** attribute set to **System.Web.Routing**.

```
<%@ Import Namespace="System.Web.Routing" %>
```

- In the Global.asax window, type the following markup after the **Application** directive.

```
<%@ Import Namespace="System.Web.Routing" %>
```

▶ Task 3: Get Entity Model namespace

1. Open the CustomerManagement Entity Model, **CustomerManagement.edmx**, in the ADO.NET Entity Data Model Designer (Entity Designer).

   - In Solution Explorer, expand **App_Code**, and double-click **CustomerManagement.edmx**.

2. Notice the namespace that is used for the code that is generated for the entity model.

   a. In the Entity Designer, click the visual design surface.

   b. In the Properties window, notice that the name in the **Namespace** box is **CustomerManagementModel**.

3. Close the CustomerManagement Entity Model.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Close**, or press CTRL+F4.

▶ Task 4: Register the ObjectContext with the metadata model

1. Create a public **Shared** procedure named **Register**, which takes a single parameter named **routes**, of type **RouteCollection**.

> **Note:** The **Register** procedure does not return a value.

```
Public Shared Sub Register(ByVal routes As RouteCollection)
End Sub
```

   - In the Global.asax window, type the following code after the opening **script** tag.

```
Public Shared Sub Register(ByVal routes As RouteCollection)
End Sub
```

2. In the Global.asax file, in the **Application_Start** method, call the **Register** procedure by passing **RouteTable.Routes** as the only parameter.

```
Register(RouteTable.Routes)
```

   - Add the following code to the **Application_Start** method.

```
Register(RouteTable.Routes)
```

3. Add and initialize a new private shared variable of type **MetaModel**, named **defaultModel**. Initialize by using the default and parameterless constructor.

```
Private Shared defaultModel As New MetaModel()
```

   - In the Global.asax window, type the following code after the opening **script** tag.

```
Private Shared defaultModel As New MetaModel()
```

4. Add a new public shared property of type **MetaModel**, named **DefaultMetaModel**. Make the property read-only, and have it return the private and shared variable **defaultModel**.

```
Public Shared ReadOnly Property DefaultMetaModel As MetaModel
    Get
        Return defaultModel
    End Get
End Property
```

   - In the Global.asax window, type the following code after the declaration of the private shared **defaultModel** variable.

```
Public Shared ReadOnly Property DefaultMetaModel As MetaModel
    Get
        Return defaultModel
    End Get
End Property
```

5. Register the **CustomerManagementModel.Entities ObjectContext** in the **Register** procedure with scaffolding enabled for all tables, by using the **RegisterContext** method of the **MetaModel** class.

```
DefaultMetaModel.RegisterContext(GetType(CustomerManagementModel.E
ntities), New ContextConfiguration() With {

    .ScaffoldAllTables = True
})
```

   - Add the following code to the **Register** procedure.

```
DefaultMetaModel.RegisterContext(GetType(CustomerManagementMod
el.Entities), New ContextConfiguration() With {

    .ScaffoldAllTables = True
})
```

▶ Task 5: Add a generic route to the routing table

1. In the **Register** procedure, add a generic route of type **DynamicDataRoute** for the **List**, **Details**, **Edit**, and **Insert** actions, in that order. Use the DefaultMetaModel as the meta model to redirect to the page template of the same name, which is prefixed in the URL with the table name. Add the route by using the **Add** method of the passed **routes** parameter.

```
routes.Add(New DynamicDataRoute("{table}/{action}.aspx") With {
    .Constraints = New RouteValueDictionary(New With {.Action =
"List|Details|Edit|Insert"}),
    .Model = DefaultMetaModel
})
```

   - Append the following code to the **Register** procedure.

```
routes.Add(New DynamicDataRoute("{table}/{action}.aspx") With
{
    .Constraints = New RouteValueDictionary(New With {.Action
= "List|Details|Edit|Insert"}),
    .Model = DefaultMetaModel
})
```

2. Save the Global Application Class project item.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Global.asax**, or press CTRL+S.
3. Close the Global Application Class project item.
   - In the Global.asax window, click the **Close** button, or press CTRL+F4.

Exercise 3: Map, Clean, and Test the Solution

▶ Task 1: Map navigation to Dynamic Data page templates

1. Open the web.sitemap file.
   - In Solution Explorer, right-click **web.sitemap**, and then click **Open**.
2. Map the **New** menu command in the **Customers** menu to the Insert page template, by modifying the **url** attribute for the **New** siteMapNode within the **Customers** siteMapNode, to a value of **~/Customers/Insert.aspx**.
   - In the web.sitemap window, locate the **New** siteMapNode within the **Customers** siteMapNode, and then modify the value of the **url** attribute.

```
<siteMapNode title="New" description="Create New Customer"
url="~/Customers/Insert.aspx" />
```

3. Map the **All** menu command in the **Customers** menu to the List page template, by modifying the **url** attribute for the **All** siteMapNode within the **Customers** siteMapNode, to a value of **~/Customers/List.aspx**.
   - In the web.sitemap window, locate the **All** siteMapNode within the **Customers** siteMapNode, and then modify the value of the **url** attribute.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers/List.aspx" />
```

4. Add a **New** menu command to the **Countries** menu to make it use the Insert page template, by adding a new siteMapNode within the **Countries** siteMapNode.
   - In the web.sitemap window, locate the **Countries** siteMapNode, and then add the following markup, above the existing Import siteMapNode.

```
<siteMapNode title="New" description="Create New Country"
url="~/Countries/Insert.aspx" />
```

5. Add an **All** menu command to the **Countries** menu to make it use the List page template, by adding a new siteMapNode within the **Countries** siteMapNode.
   - In the web.sitemap window, locate the **Countries** siteMapNode, and then add the following markup, above the existing **Import** siteMapNode.

```
<siteMapNode title="All" description="View All Countries"
url="~/Countries/List.aspx" />
```

6. Save and close the web.sitemap file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**, or press CTRL+S.
   b. In the web.sitemap window, click the **Close** button, or press CTRL+F4.

▶ Task 2: Remove superfluous solution and project items

1. Remove the **CustomerManagementEntities** project from the CustomerManagement solution.
   a. In Solution Explorer, right-click **CustomerManagementEntities**, and then click **Remove**.
   b. In the Microsoft Visual Studio message box, click **OK**.
2. Delete the **Customers** Web Form in the CustomerManagement Web site.
   a. In Solution Explorer, right-click **Customers.aspx**, and then click **Delete**.
   b. In the Microsoft Visual Studio message box, click **OK**.
3. Delete the **InsertCustomer** Web Form in the CustomerManagement Web site.
   a. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **Delete**.
   b. In the Microsoft Visual Studio message box, click **OK**.
4. Delete the **Customer** user control in the CustomerManagement Web site.
   a. In Solution Explorer, right-click **Customer.ascx**, and then click **Delete**.
   b. In the Microsoft Visual Studio message box, click **OK**.

▶ Task 3: Test the Dynamic Data functionality

1. Run the **CustomerManagement** application.
   a. In Solution Explorer, under D:\Labfiles\Starter\M10\VB \CustomerManagement, click **Default.aspx**.
   b. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.
2. Show all customers, by clicking **All**, on the **Customers** menu.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **All**.
3. Close all open windows.
   a. In the Customers – Windows Internet Explorer window, click the **Close** button.
   b. In the Windows Explorer window, click the **Close** button.

▶ Task 4: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 10

Lab Answer Key: Managing Data by Using Microsoft® ASP.NET Dynamic Data (Visual C#)

Contents:

Lab: Managing Data by Using ASP.NET Dynamic Data

(C#)

Exercise 1: Adding dynamic data to an Existing Web Site

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the CustomerManagement solution from the D:\Labfiles\Starter \M10\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M10\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Create a sample dynamic data Web site

1. Open a second instance of Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. Create a new ASP.NET Dynamic Data Web site in the D:\Labfiles\Starter \M10\CS\SampleDDWebSite folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **New Web Site**.

b. In the **New Web Site** dialog box, in the left pane, ensure that **Visual C#** is selected, in the middle pane, click **ASP.NET Dynamic Data Entities Web Site**, in the **Web location** list, click **File System**, in the adjacent text box, type **D:\Labfiles\Starter\M10\CS\SampleDDWebSite**, and then click **OK**.

3. Close the second instance of Visual Studio 2010.

   - In the SampleDDWebSite – Microsoft Visual Studio window, click the **Close** button, or press the ALT+F4 keys.

▶ Task 3: Add the dynamic data default files and folders

1. Add the **DynamicData** folder and default dynamic data content from the SampleDDWebSite Web site to the **CustomerManagement** Web site by copying the DynamicData folder from the path, **D:\Labfiles\Starter\M10\CS\SampleDDWebSite\DynamicData**. Use Windows Explorer for this purpose.

   a. On the **Start** menu, click **Run**.

   b. In the **Run** dialog box, in the **Open** box, type **D:\Labfiles\Starter\M10\CS\SampleDDWebSite**, and then click **OK**.

   c. In the Windows® Explorer window, right-click **DynamicData**, and then click **Copy**.

   d. Right-click **CustomerManagement**, and then click **Paste**.

   e. Close the Windows Explorer window.

2. In Solution Explorer, refresh the Web site content to ensure the addition of the **DynamicData** folder.

   - In Solution Explorer, right-click **D:\Labfiles\Starter\M10\CS\CustomerManagement**, and then click **Refresh Folder**.

▶ Task 4: Add the ScriptManager control to the master page

1. Open the **Site.master** master page in Source view.
   - In Solution Explorer, right-click **Site.master**, and then click **View Markup**.
2. Add a **ScriptManager** control before the **div** element with a value of **content** for the class attribute, by adding an **asp:ScriptManager** element with an **ID** attribute value of **MainScriptManager**.

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

   - In the Site.master window, type the following markup before the **div** element with a value of **content** for the class attribute.

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

3. Save the **Site.master** master page.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**, or press CTRL+S.
4. Close the **Site.master** master page.
   - In the Site.master window, click the **Close** button, or press CTRL+F4.

▶ Task 5: Update the dynamic data page templates to use an existing master page

1. Open the **Find and Replace** dialog box.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Edit** menu, point to **Find and Replace**, and then click **Replace in Files**, or press the CTRL+SHIFT+H keys.
2. Replace the **ContentPlaceHolder1** name that is referenced in the Dynamic Data Page Templates, with the name of the **ContentPlaceHolder** control that is used in the existing master page, **MainContentPlaceHolder**.
   a. In the Find and Replace dialog box, in the **Find wha**t box, type **ContentPlaceHolder1**, in the **Replace with** box, type **MainContentPlaceHolder**, and then click **Replace All**.
   b. In the **Replace All** dialog box, select the **Replace All will open all files with changes for editing** check box, and then click **Yes**.

c. In the **Microsoft Visual Studio** dialog box, click **OK**, and in the **Find and Replace** dialog box, click the **Close** button.

3. In all of the opened and modified Dynamic Data page templates—**Details.aspx**, **Edit.aspx**, **Insert.aspx**, **List.aspx**, and **ListDetails.aspx**—remove the **Content** control with an **ID** attribute value of **headContent**.

   a. In the Dynamic Data page template, locate and select the **Content** control with an **ID** attribute value of **headContent**.

   ```
   <asp:Content ID="headContent" ContentPlaceHolderID="head"
   Runat="Server">
   </asp:Content>
   ```

   b. On the **Edit** menu, click **Delete**, or press the DELETE key, to delete the selected markup.

4. Save and close all modified files.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save All**, or press CTRL+SHIFT+S.

   b. Click the **Close** button for all open Web Forms.

▶ Task 6: Copy dynamic data styles

1. Copy the content of the **D:\Labfiles\Starter\M10\CS\SampleDDWebSite\Site.css** stylesheet.

   a. On the **Start** menu, click **Run**.

   b. In the **Run** dialog box, in the **Open** box, type **D:\Labfiles\Starter\M10\CS\SampleDDWebSite**, and then click **OK**.

   c. In the Windows Explorer window, double-click **Site.css**.

   d. In the Site.css window, select all by pressing CTRL+A, right-click the selection, and then click **Copy**.

2. Close the Windows Explorer window.

3. Add the copied content to the existing Styles\Site.css stylesheet.
   a. In Solution Explorer, expand **Styles**, and double-click **Site.css**.
   b. In the Styles/Site.css window, press CTRL+END, press ENTER, and then press SHIFT+INSERT.
4. Save and close the Site.css file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Styles/Site.css**, or press CTRL +S.
   b. In the Styles/Site.css window, click the **Close** button.

Exercise 2: Registering Entity Framework with Dynamic Data

▶ Task 1: Add a Global Application class file

- Add the Global Application Class project item to the **CustomerManagement** Web site.

  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M10\CS \CustomerManagement**, and then click **Add New Item**.

  b. In the **Add New Item - D:\Labfiles\Starter\M10\CS \CustomerManagement** dialog box, in the middle pane, click **Global Application Class**, and then click **Add**.

▶ Task 2: Import the required namespace

- Add an **Import** directive to the Global Application Class with the **Namespace** attribute set to **System.Web.Routing**.

```
<%@ Import Namespace="System.Web.Routing" %>
```

- In the Global.asax window, type the following markup after the **Application** directive.

```
<%@ Import Namespace="System.Web.Routing" %>
```

▶ Task 3: Get Entity Model namespace

1. Open the CustomerManagement Entity Model **CustomerManagement.edmx** that is located in the App_Code folder, in the ADO.NET Entity Data Model Designer (Entity Designer).

   - In Solution Explorer, expand **App_Code**, and double-click **CustomerManagement.edmx**.

2. Notice the namespace that is used for the code that is generated for the entity model.

   a. In the Entity Designer, click the visual design surface.

   b. In the Properties window, notice that the name in the **Namespace** box is **CustomerManagementModel**.

3. Close the CustomerManagement Entity Model.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Close**, or press CTRL+F4.

▶ Task 4: Register the ObjectContext with the metadata model

1. Create a public **static** procedure named **Register**, which takes a single parameter named **routes**, of type **RouteCollection**.

> **Note:** The **Register** procedure does not return a value.

```
public static void Register(RouteCollection routes)
{
}
```

   - In the Global.asax window, type the following code after the opening **script** tag.

```
public static void Register(RouteCollection routes)
{
}
```

2. In the **Global.asax** file, in the **Application_Start** method, call the **Register** procedure by passing **RouteTable.Routes** as the only parameter.

```
Register(RouteTable.Routes);
```

   - Add the following code to the **Application_Start** method.

```
Register(RouteTable.Routes);
```

3. Add and initialize a new private static variable of type **MetaModel**, named **defaultModel**. Initialize by using the default and parameterless constructor.

```
private static MetaModel defaultModel = new MetaModel();
```

   - In the Global.asax window, type the following code after the opening **script** tag.

```
private static MetaModel defaultModel = new MetaModel();
```

4. Add a new public static property of type **MetaModel**, named **DefaultMetaModel**. Make the property read-only, and have it return the private and static variable **defaultModel**.

```
public static MetaModel DefaultMetaModel
{
    get
    {
        return defaultModel;
    }
}
```

   - In the Global.asax window, type the following code after the declaration of the private static **defaultModel** variable.

```
public static MetaModel DefaultMetaModel
{
    get
    {
        return defaultModel;
    }
}
```

5. Register the **CustomerManagementModel.Entities ObjectContext** in the **Register** procedure with scaffolding enabled for all tables, by using the **RegisterContext** method of the **MetaModel** class.

```
DefaultMetaModel.RegisterContext(typeof(CustomerManagementModel.En
tities), new ContextConfiguration()
{
    ScaffoldAllTables = true
});
```

   - Add the following code to the **Register** procedure.

```
DefaultMetaModel.RegisterContext(typeof(CustomerManagementMode
l.Entities), new ContextConfiguration()
{
    ScaffoldAllTables = true
});
```

▶ Task 5: Add a generic route to the routing table

1. In the **Register** procedure, add a generic route of type **DynamicDataRoute** for the **List**, **Details**, **Edit**, and **Insert** actions, in that order. Use the DefaultMetaModel as the meta model to redirect to the page template of the same name, which is prefixed in the URL with the table name. Add the route by using the **Add** method of the passed **routes** parameter.

```
routes.Add(new DynamicDataRoute("{table}/{action}.aspx") {
    Constraints = new RouteValueDictionary(new { action =
"List|Details|Edit|Insert" }),
    Model = DefaultMetaModel
});
```

   - Append the following code to the **Register** procedure.

```
routes.Add(new DynamicDataRoute("{table}/{action}.aspx") {
    Constraints = new RouteValueDictionary(new { action =
"List|Details|Edit|Insert" }),
    Model = DefaultMetaModel
});
```

2. Save the Global Application Class project item.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Global.asax**, or press CTRL+S.

3. Close the Global Application Class project item.

   - In the Global.asax window, click the **Close** button, or press CTRL+F4.

Exercise 3: Map, Clean, and Test the Solution

▶ Task 1: Map navigation to Dynamic Data page templates

1. Open the web.sitemap file.
   - In Solution Explorer, right-click **web.sitemap**, and then click **Open**.
2. Map the **New** menu command in the **Customers** menu to the Insert page template, by modifying the **url** attribute for the **New** siteMapNode within the **Customers** siteMapNode, to a value of **~/Customers/Insert.aspx**.
   - In the web.sitemap window, locate the **New** siteMapNode within the **Customers** siteMapNode, and then modify the value of the **url** attribute.

```
<siteMapNode title="New" description="Create New Customer"
url="~/Customers/Insert.aspx" />
```

3. Map the **All** menu command in the **Customers** menu to the List page template, by modifying the **url** attribute for the **All** siteMapNode within the **Customers** siteMapNode, to a value of **~/Customers/List.aspx**.
   - In the web.sitemap window, locate the **All** siteMapNode within the **Customers** siteMapNode, and then modify the value of the **url** attribute.

```
<siteMapNode title="All" description="View All Customers"
url="~/Customers/List.aspx" />
```

4. Add a **New** menu command to the **Countries** menu to make it use the Insert page template, by adding a new siteMapNode within the **Countries** siteMapNode.
   - In the web.sitemap window, locate the **Countries** siteMapNode, and then add the following markup, above the existing Import siteMapNode.

```
<siteMapNode title="New" description="Create New Country"
url="~/Countries/Insert.aspx" />
```

5. Add an **All** menu command to the **Countries** menu to make it use the List page template, by adding a new siteMapNode within the **Countries** siteMapNode.

- In the web.sitemap window, locate the **Countries** siteMapNode, and then add the following markup, above the existing Import siteMapNode.

```
<siteMapNode title="All" description="View All Countries"
url="~/Countries/List.aspx" />
```

6. Save and close the web.sitemap file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**, or press CTRL+S.
   b. In the web.sitemap window, click the **Close** button, or press CTRL+F4.

▶ Task 2: Remove superfluous solution and project items

1. Remove the CustomerManagementEntities project from the CustomerManagement solution.
   a. In Solution Explorer, right-click **CustomerManagementEntities**, and then click **Remove**.
   b. In the Microsoft Visual Studio message box, click **OK**.
2. Delete the **Customers** Web Form in the CustomerManagement Web site.
   a. In Solution Explorer, right-click **Customers.aspx**, and then click **Delete**.
   b. In the Microsoft Visual Studio message box, click **OK**.
3. Delete the **InsertCustomer** Web Form in the CustomerManagement Web site.
   a. In Solution Explorer, right-click **InsertCustomer.aspx**, and then click **Delete**.
   b. In the Microsoft Visual Studio message box, click **OK**.
4. Delete the **Customer** user control in the CustomerManagement Web site.
   a. In Solution Explorer, right-click **Customer.ascx**, and then click **Delete**.
   b. In the Microsoft Visual Studio message box, click **OK**.

▶ Task 3: Test the Dynamic Data functionality

1. Run the **CustomerManagement** application.
   a. In Solution Explorer, under D:\Labfiles\Starter\M10\CS \CustomerManagement, click **Default.aspx**.
   b. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.
2. Show all customers, by clicking **All**, on the **Customers** menu.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **All**.
3. Close all open windows.
   a. In the Customers – Windows Internet Explorer window, click the **Close** button.
   b. In the Windows Explorer window, click the **Close** button.

▶ Task 4: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 11

Lab Answer Key: Creating a Microsoft® ASP.NET Ajax-enabled Web Forms Application (Visual Basic)

Contents:

Lab: Creating a Microsoft® ASP.NET Ajax-Enabled Web Forms Application

(Visual Basic)

Exercise 1: Creating a Modal About Box

▶ Task 1: Open an existing Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M11\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M11\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Add a new Web Form

1. Add a new Web Form named **About.aspx**, based on the master page **Site.master**.
   a. In Solution Explorer, right-click **D:\Labfiles\Starter\M11\VB \CustomerManagement**, and then click **Add New Item**.
   b. In **the Add New Item - D:\Labfiles\Starter\M11\VB \CustomerManagement** dialog box, in the middle pane, click **Web Form**.

c. In the **Name** box, type **About.aspx**, select the **Select master page** check box, and then click **Add**.

d. In the **Select a Master Page** dialog box, in the **Contents of folder** box, click **Site.master**, and then click **OK**.

2. Change the **ID** property of the **Content** control to **AboutContent**.

- In the About.aspx window, locate the **Content** control, and change the value of the **ID** property from **Content1** to **AboutContent**.

```
<asp:Content ID="AboutContent"
ContentPlaceHolderID="MainContentPlaceHolder" Runat="Server">
```

▶ Task 3: Add a reference to the Ajax Control Toolkit assembly

- Add a reference to the **C:\Program Files\ASP.NET Ajax Library \WebForms\Debug\AjaxControlToolkit.dll** assembly.

a. In Solution Explorer, right-click **D:\Labfiles\Starter\M11\VB \CustomerManagement**, and then click Add **Reference**.

b. On the **Browse** tab of the **Add Reference** dialog box, in the **File name** box, type **C:\Program Files\ASP.NET Ajax Library\WebForms\Debug \AjaxControlToolkit.dll**, and then click **OK**.

▶ Task 4: Register the Ajax Toolkit assembly, namespace, and TagPrefix

1. Add a **Register** directive to the **web.config** file by using the following attribute values:

   - assembly: **AjaxControlToolkit**
   - namespace: **AjaxControlToolkit**
   - tagPrefix: **ajaxToolKit**

a. In Solution Explorer, double-click **web.config**.

b. In the web.config window, type the following markup after the closing **compilation** tag.

```
<pages>
    <controls>
        <add assembly="AjaxControlToolkit"
namespace="AjaxControlToolkit" tagPrefix="ajaxToolKit" />
    </controls>
</pages>
```

2. Save and close the web.config file.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**, or press the CTRL+S keys.

   b. In the web.config window, click the **Close** button, or press CTRL+F4.

▶ Task 5: Change the ScriptManager control in the master page

1. Open the **Site.master** master page in Source view.

   - In Solution Explorer, right-click **Site.master**, and then click **View Markup**.

2. Locate the **ScriptManager** control.

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

3. Change the **ScriptManager** control to a **ToolkitScriptManager**.

```
<ajaxToolKit:ToolkitScriptManager ID="MainScriptManager"
runat="server" />
```

   - In the Site.master window, modify the existing **ScriptManager** control to look like the following code.

```
<ajaxToolKit:ToolkitScriptManager ID="MainScriptManager"
runat="server" />
```

4. Save the **Site.master** master page.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**, or press CTRL+S.

5. Close the Site.master master page.

   - In the Site.master window, click the **Close** button, or press CTRL+F4.

▶ Task 6: Add the ScriptManagerProxy control

- Add a **ScriptManagerProxy** control named **AboutScriptManagerProxy**, to the **AboutContent** control.

```
<asp:ScriptManagerProxy runat="server"
ID="AboutScriptManagerProxy" />
```

- In the About.aspx window, type the following markup within the **AboutContent** control.

```
<asp:ScriptManagerProxy runat="server"
ID="AboutScriptManagerProxy" />
```

▶ Task 7: Add the HTML and UI controls

1. Append a **Panel** control named **AboutPanel**, to the **Content** control. The panel can be made invisible by using an inline style of **display: none**.

- In the About.aspx window, append the following markup within the **AboutContent** control.

```
<asp:Panel ID="AboutPanel" runat="server" Style="display:
none">
</asp:Panel>
```

2. Add an **UpdatePanel** control named **AboutUpdatePanel**, with an empty **ContentTemplate** element, to the **AboutPanel** control.

- In the About.aspx window, add the following markup within the **AboutPanel** control.

```
<asp:UpdatePanel ID="AboutUpdatePanel" runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

3. Add a **Panel** control named **AboutPopupPanel**, with a cascading style sheet (CSS) class of **aboutPopupPanel**, to the **ContentTemplate** element in the **AboutUpdatePanel** control.

   - In the About.aspx window, add the following markup to the **ContentTemplate** element of the **AboutUpdatePanel** control.

```
<asp:Panel runat="server" ID="AboutPopupPanel"
CssClass="aboutPopupPanel">
</asp:Panel>
```

4. Add the following HTML tags and text within the **AboutPopupPanel** control.

```
<br />
 <b>About Contoso Customer Management</b> <br />
<br />
 Copyright © 2010 by Contoso <br />
<br />
```

   - In the About.aspx window, add the following markup within the **AboutPopupPanel** control.

```
<br />
 <b>About Contoso Customer Management</b> <br />
<br />
 Copyright © 2010 by Contoso <br />
<br />
```

5. Append a **Button** control named **ShowModalButton**, with a CSS class of **displayNone** to the **AboutPopupPanel** control.

   - In the About.aspx window, append the following markup within the **AboutPopupPanel** control.

```
<asp:Button ID="ShowModalButton" runat="server"
CssClass="displayNone" />
```

6. Append a **ModalPopupExtender** control named **AboutModalPopupExtender**, with the following attribute values to the **AboutPopupPanel** control:

    - TargetControlID: **ShowModalButton**
    - PopupControlID: **AboutPanel**
    - CancelControlID: **CloseButton**
    - BackgroundCssClass: **modalPopup**

    - In the About.aspx window, append the following markup within the **AboutPopupPanel** control.

```
<ajaxToolKit:ModalPopupExtender ID="AboutModalPopupExtender"
runat="server" TargetControlID="ShowModalButton"
    PopupControlID="AboutPanel" CancelControlID="CloseButton"
BackgroundCssClass="modalPopup" />
```

7. Append a **Button** control named **CloseButton**, which does not trigger validation, set the **OnClick** event to **CloseButton_Click**, set the **Text** property to **Close**, and add two HTML line break elements to the **AboutPopupPanel** control.

    - In the About.aspx window, append the following markup within the **AboutPopupPanel** control.

```
<asp:Button ID="CloseButton" runat="server"
CausesValidation="False" OnClick="CloseButton_Click"
Text="Close" />
<br />
<br />
```

8. Add the following styles to the Styles/Site.css stylesheet.

```
div.aboutPopupPanel
{
    background-color: White;
    text-align: center;
    border: 2px solid Black;
}
.modalPopup
{
    position: relative;
    background-color: Gray;
    -ms-filter: alpha(opacity=70);
    z-index: 1;
}
.displayNone
{
    display: none;
}
```

   a. In Solution Explorer, expand **Styles**, and double-click **Site.css**.

   b. In the Styles/Site.css window, press the CTRL+END keys, press ENTER, and then type the following markup.

```
div.aboutPopupPanel
{
    background-color: White;
    text-align: center;
    border: 2px solid Black;
}
.modalPopup
{
    position: relative;
    background-color: Gray;
    -ms-filter: alpha(opacity=70);
    z-index: 1;
}
.displayNone
{
    display: none;
}
```

9. Save and close the Site.css file.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Styles/Site.css**, or press CTRL+S.

   b. In the Styles/Site.css window, click the **Close** button.

▶ Task 8: Add the event handler and code

1. View the code-behind file of the About.aspx Web Form.
   - In the About.aspx window, right-click anywhere, and then click **View Code**.
2. Show the **AboutModalPopupExtender** control in the **Page.Load** event.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Show about box
    AboutModalPopupExtender.Show()
End Sub
```

   - In the About.aspx.vb code window, add the following code to the **About** class.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Show about box
    AboutModalPopupExtender.Show()
End Sub
```

3. Hide the **AboutModalPopupExtender** control in the **CloseButton_Click** event handler.

```
Protected Sub CloseButton_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles CloseButton.Click
    ' Close about box
    AboutModalPopupExtender.Hide()
End Sub
```

   - In the About.aspx.vb code window, add the following code to the **About** class.

```
Protected Sub CloseButton_Click(ByVal sender As Object, ByVal
e As System.EventArgs) Handles CloseButton.Click
    ' Close about box
    AboutModalPopupExtender.Hide()
End Sub
```

▶ Task 9: Map navigation to the About Web Form

1. Open the web.sitemap file.
   - In Solution Explorer, right-click **web.sitemap**, and then click **Open**.
2. Add an **About** menu command to the **Help** menu, making it use the **About** Web Form, by adding a new siteMapNode within the **Help** siteMapNode.
   - In the web.sitemap window, locate the **Help** siteMapNode, and then add the following markup.

```
<siteMapNode title="About" description="" url="~/About.aspx"
/>
```

3. Save and close the web.sitemap file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**, or press CTRL+S.
   b. In the web.sitemap window, click the **Close** button, or press CTRL+F4.

▶ Task 10: Test the modal About box

1. Build the solution.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Solution**, or press CTRL+SHIFT+B.
2. Run the CustomerManagement Web application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.
3. Close the **About** box.
   - In the **About Contoso Customer Management** message box, click the **Close** button.
4. Show the **About** box, using the menu.
   - In the Contoso Customer Management – Windows Internet Explorer window, in the **Help** menu, click **About**.
5. Close the **About** box.
   - In the **About Contoso Customer Management** message box, click the **Close** button.

6. Close the CustomerManagement Web application.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button, or press CTRL+F4.
7. Close the About.aspx.vb code file.
   - In the About.aspx.vb window, click the **Close** button, or press CTRL+F4.
8. Close the **About** Web Form.
   - In the About.aspx window, click the **Close** button, or press CTRL+F4.

Exercise 2: Customizing Dynamic Data Field Templates with Ajax Server Controls

▶ Task 1: Add the CalendarExtender control to the DateTime_Edit field template

1. Open the **DateTime_Edit.ascx** user control Dynamic Data field template in the Source view.
   - In Solution Explorer, expand **DynamicData**, expand **FieldTemplates**, right-click **DateTime_Edit.ascx**, and then click **View Markup**.
2. Add a **CalendarExtender** control named **DateTimeEditCalendarExtender**, to the user control by using the following attribute values:
   - TargetControlID: **TextBox1**
   - PopupPosition: **Right**
   - In the DateTime_Edit.ascx window, type the following markup after the **TextBox** control.

```
<ajaxToolKit:CalendarExtender
ID="DateTimeEditCalendarExtender" runat="server"
    TargetControlID="TextBox1" PopupPosition="Right" />
```

3. Save and close the **DateTime_Edit.ascx** user control.
   a. In the **CustomerManagement** – Microsoft Visual Studio window, on the **File** menu, click **Save DynamicData/FieldTemplates /DateTime_Edit.ascx**, or press CTRL+S.
   b. In the DynamicData/FieldTemplates/DateTime_Edit.ascx window, click the **Close** button, or press CTRL+F4.

▶ Task 2: Test the DateTime_Edit field template

1. Build the solution.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Solution**.
2. Run the CustomerManagement Web application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

3. Test the **CalendarExtender** control by adding a new entry to the Customers table, click the **ModifiedDate** box, and then select a date from the calendar.
   - In the Customers – Windows Internet Explorer window, on the **Customers** menu, click **New**, click the **ModifiedDate** box, and then select a date from the calendar.
4. Cancel the new customer entry.
   - On the CUSTOMER MANAGEMENT page, click **Cancel**.
5. Close the CustomerManagement Web application.
   - In the Customers – Windows Internet Explorer window, click the **Close** button.

Exercise 3: Adding the Country Import Progress Indicator

▶ Task 1: Add the ScriptManagerProxy control

1. Open the **ImportCountries** Web Form in the Source view.
   - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View Markup**.
2. Rename the **Content** control from **Content1** to **ImportCountriesContent**.
   - In the ImportCountries.aspx window, rename the **Content** control from **Content1** to **ImportCountriesContent**.
3. Add a **ScriptManagerProxy** control named **ImportCountriesScriptManagerProxy**, to the **Content** control.

```
<asp:ScriptManagerProxy runat="server"
ID="ImportCountriesScriptManagerProxy" />
```

   - In the ImportCountries.aspx window, type the following markup,

```
<asp:ScriptManagerProxy runat="server"
ID="ImportCountriesScriptManagerProxy" />
```

     at the top of the **ImportCountriesContent** control.

```
<asp:Content ID="ImportCountriesContent"
ContentPlaceHolderID="MainContentPlaceHolder" runat="Server">
    ...
</asp:Content>
```

▶ Task 2: Add the HTML and UI controls

1. Add an **UpdatePanel** control named **ImportCountriesUpdatePanel**, to the **ImportCountriesContent** control, with an empty **ContentTemplate** element, after the **ImportCountriesScriptManagerProxy** control.

```
<asp:UpdatePanel ID="ImportCountriesUpdatePanel" runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

- In the ImportCountries.aspx window, type the following markup,

```
<asp:UpdatePanel ID="ImportCountriesUpdatePanel"
runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

after the **ImportCountriesScriptManagerProxy** control.

```
<asp:ScriptManagerProxy runat="server"
ID="ImportCountriesScriptManagerProxy" />
```

2. Move the existing content of the **ImportCountriesContent** control to the **ContentTemplate** element.

   a. In the ImportCountries.aspx window, select the following markup,

```
<asp:XmlDataSource ID="CountriesXmlDataSource" runat="server"
DataFile="~/Countries.xml">
</asp:XmlDataSource>
<div class="importCountriesHeader">
    <asp:Button ID="FilterButton" runat="server" Text="Filter
Countries" />
    <asp:Button ID="SaveButton" runat="server" Text="Save
Countries" />
</div>
<div class="importResult">
    <asp:Label ID="ImportResultLabel" runat="server"
EnableViewState="false"></asp:Label>
</div>
<asp:GridView ID="CountriesGridView" runat="server"
AutoGenerateColumns="False"
DataSourceID="CountriesXmlDataSource"
        Width="100%">
    <Columns>
        <asp:BoundField DataField="ID" HeaderText="ID"
SortExpression="ID" />
        <asp:BoundField DataField="Name" HeaderText="Name"
SortExpression="Name" />
        <asp:BoundField DataField="PhoneNoFormat"
HeaderText="PhoneNoFormat" SortExpression="PhoneNoFormat" />
        <asp:BoundField DataField="DialingCountryCode"
HeaderText="DialingCountryCode"
SortExpression="DialingCountryCode" />
        <asp:BoundField DataField="InternationalDialingCode"
HeaderText="InternationalDialingCode"
            SortExpression="InternationalDialingCode" />
        <asp:BoundField DataField="InternetTLD"
HeaderText="InternetTLD" SortExpression="InternetTLD" />
    </Columns>
</asp:GridView>
```

   right-click the selected text, and then click **Cut**.

   b. Place the cursor within the opening and closing tags of the **ContentTemplate** element, right-click, and then click **Paste**.

3. Format the document markup.

   - To format the markup, press CTRL+K, and then press CTRL+D.

```
<asp:UpdatePanel ID="ImportCountriesUpdatePanel"
runat="server">
    <ContentTemplate>
        <asp:XmlDataSource ID="CountriesXmlDataSource"
runat="server" DataFile="~/Countries.xml">
        </asp:XmlDataSource>
        <div class="importCountriesHeader">
            <asp:Button ID="FilterButton" runat="server"
Text="Filter Countries" />
            <asp:Button ID="SaveButton" runat="server"
Text="Save Countries" />
        </div>
        <div class="importResult">
            <asp:Label ID="ImportResultLabel" runat="server"
EnableViewState="false"></asp:Label>
        </div>
        <asp:GridView ID="CountriesGridView" runat="server"
AutoGenerateColumns="False"
DataSourceID="CountriesXmlDataSource"
                Width="100%">
            <Columns>
                <asp:BoundField DataField="ID" HeaderText="ID"
SortExpression="ID" />
                <asp:BoundField DataField="Name"
HeaderText="Name" SortExpression="Name" />
                <asp:BoundField DataField="PhoneNoFormat"
HeaderText="PhoneNoFormat" SortExpression="PhoneNoFormat" />
                <asp:BoundField DataField="DialingCountryCode"
HeaderText="DialingCountryCode"
SortExpression="DialingCountryCode" />
                <asp:BoundField
DataField="InternationalDialingCode"
HeaderText="InternationalDialingCode"

SortExpression="InternationalDialingCode" />
                <asp:BoundField DataField="InternetTLD"
HeaderText="InternetTLD" SortExpression="InternetTLD" />
            </Columns>
        </asp:GridView>
    </ContentTemplate>
</asp:UpdatePanel>
```

4. Add an **UpdateProgress** control named **ImportCountriesUpdateProgress**, to the **ImportCountriesContent** control, after the **ImportCountriesScriptManagerProxy** control, by using the following attribute values:
    - AssociatedUpdatePanelID: **ImportCountriesUpdatePanel**
    - DisplayAfter: **0**
    - In the ImportCountries.aspx window, type the following markup in the **ImportCountriesContent** control, after the **ImportCountriesScriptManagerProxy** control.

```
<asp:UpdateProgress ID="ImportCountriesUpdateProgress"
runat="server"
AssociatedUpdatePanelID="ImportCountriesUpdatePanel"
DisplayAfter="0">
</asp:UpdateProgress>
```

5. Add an opening and a closing **ProgressTemplate** tag with the text, "**Processing...**" to the **ImportCountriesUpdateProgress** control.
    - In the ImportCountries.aspx window, type the following markup within the **ImportCountriesUpdateProgress** control.

```
<ProgressTemplate>
    Processing...
</ProgressTemplate>
```

6. Save and close the **ImportCountries.aspx** Web Form.
    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx**.
    b. In the ImportCountries.aspx window, click the **Close** button.

▶ Task 3: Test the update progress

1. Build the solution.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Solution**.
2. Debug the application.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**, or press F5.
   b. In the **Debugging Not Enabled** dialog box, click **OK**.
3. Filter the countries.
   a. In the Contoso Customer Management – Windows Internet Explorer window, click the **Filter Countries** button.

> **Note:** Notice that the text, "Processing...", displays in the upper-left corner of the Web site for a short period of time.

   b. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
   c. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 4: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 11

Lab Answer Key: Creating a Microsoft® ASP.NET Ajax-enabled Web Forms Application (Visual C#)

Contents:

Lab: Creating a Microsoft® ASP.NET Ajax-Enabled Web Forms Application

(C#)

Exercise 1: Creating a Modal About Box

▶ Task 1: Open an existing Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M11\CS folder.
   - In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   - In the **Open Project dialog** box, in the **File name** box, type **D:\Labfiles\Starter\M11\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Add a new Web Form

1. Add a new Web Form named **About.aspx**, based on the master page, **Site.master**.
   - In Solution Explorer, right-click **D:\Labfiles\Starter\M11\CS\CustomerManagement**, and then click **Add New Item**.
   - In the **Add New Item - D:\Labfiles\Starter\M11\CS\CustomerManagement** dialog box, in the middle pane, click **Web Form**.

- In the **Name** box, type **About.aspx**, select the **Select master page** check box, and then click **Add**.
- In the **Select a Master Page** dialog box, in the **Contents of folder** box, click **Site.master**, and then click **OK**.

2. Change the **ID** property of the **Content** control to **AboutContent**.
   - In the **About.aspx** window, locate the **Content** control, and change the value of the **ID** property from **Content1** to **AboutContent**.

```
<asp:Content ID="AboutContent"
ContentPlaceHolderID="MainContentPlaceHolder" Runat="Server">
```

▶ Task 3: Add a reference to the Ajax Control Toolkit assembly

- Add a reference to the **C:\Program Files\ASP.NET Ajax Library \WebForms\Debug\AjaxControlToolkit.dll** assembly.
  a. In Solution Explorer, right-click **D:\Labfiles\Starter\M11\CS \CustomerManagement**, and then click **Add Reference**.
  b. On the **Browse** tab of the **Add Reference** dialog box, in the **File name** box, type **C:\Program Files\ASP.NET Ajax Library\WebForms \Debug\AjaxControlToolkit.dll**, and then click **OK**.

▶ Task 4: Register the Ajax Toolkit assembly, namespace, and TagPrefix

1. Add a **Register** directive to the **web.config** file by using the following attribute values:
   - assembly: **AjaxControlToolkit**
   - namespace: **AjaxControlToolkit**
   - tagPrefix: **ajaxToolKit**

   a. In Solution Explorer, double-click **web.config**.

b. In the web.config window, type the following markup after the closing **compilation** tag.

```
<pages>
    <controls>
        <add assembly="AjaxControlToolkit"
namespace="AjaxControlToolkit" tagPrefix="ajaxToolKit" />
    </controls>
</pages>
```

2. Save and close the web.config file.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**, or press the CTRL+S keys.

   b. In the web.config window, click the **Close** button, or press CTRL+F4.

▶ Task 5: Change the ScriptManager control in the master page

1. Open the **Site.master** master page in Source view.

   - In Solution Explorer, right-click **Site.master**, and then click **View Markup**.

2. Locate the **ScriptManager** control.

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

3. Change the **ScriptManager** control to a **ToolkitScriptManager**.

```
<ajaxToolKit:ToolkitScriptManager ID="MainScriptManager"
runat="server" />
```

   - In the Site.master window, modify the existing **ScriptManager** control to look like the following code.

```
<ajaxToolKit:ToolkitScriptManager ID="MainScriptManager"
runat="server" />
```

4. Save the **Site.master** master page.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**, or press CTRL+S.

5. Close the Site.master master page.

   - In the Site.master window, click the **Close** button, or press CTRL+F4.

▶ Task 6: Add the ScriptManagerProxy control

- Add a **ScriptManagerProxy** control named **AboutScriptManagerProxy**, to the **AboutContent** control.

```
<asp:ScriptManagerProxy runat="server"
ID="AboutScriptManagerProxy" />
```

- In the About.aspx window, type the following markup within the **AboutContent** control.

```
<asp:ScriptManagerProxy runat="server"
ID="AboutScriptManagerProxy" />
```

▶ Task 7: Add the HTML and UI controls

1. Append a **Panel** control named **AboutPanel**, to the **Content** control. The panel can be made invisible by using an inline style of **display: none**.

- In the About.aspx window, append the following markup within the **AboutContent** control.

```
<asp:Panel ID="AboutPanel" runat="server" Style="display: none">
</asp:Panel>
```

2. Add an **UpdatePanel** control named **AboutUpdatePanel**, with an empty **ContentTemplate** element, to the **AboutPanel** control.

- In the About.aspx window, add the following markup within the **AboutPanel** control.

```
<asp:UpdatePanel ID="AboutUpdatePanel" runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

3. Add a **Panel** control named **AboutPopupPanel**, with a cascading style sheet (CSS) class of **aboutPopupPanel**, to the **ContentTemplate** element in the **AboutUpdatePanel** control.

   - In the About.aspx window, add the following markup to the **ContentTemplate** element of the **AboutUpdatePanel** control.

```
<asp:Panel runat="server" ID="AboutPopupPanel"
CssClass="aboutPopupPanel">
</asp:Panel>
```

4. Add the following HTML tags and text within the **AboutPopupPanel** control.

```
<br />
 <b>About Contoso Customer Management</b> <br />
<br />
 Copyright © 2010 by Contoso <br />
<br />
```

   - In the About.aspx window, add the following markup within the **AboutPopupPanel** control.

```
<br />
 <b>About Contoso Customer Management</b> <br />
<br />
 Copyright © 2010 by Contoso <br />
<br />
```

5. Append a **Button** control named **ShowModalButton**, with a CSS class of **displayNone** to the **AboutPopupPanel** control.

   - In the About.aspx window, append the following markup within the **AboutPopupPanel** control.

```
<asp:Button ID="ShowModalButton" runat="server"
CssClass="displayNone" />
```

6. Append a **ModalPopupExtender** control named **AboutModalPopupExtender**, with the following attribute values to the **AboutPopupPanel** control:

   - TargetControlID: **ShowModalButton**
   - PopupControlID: **AboutPanel**
   - CancelControlID: **CloseButton**
   - BackgroundCssClass: **modalPopup**

- In the About.aspx window, append the following markup within the **AboutPopupPanel** control.

```
<ajaxToolKit:ModalPopupExtender ID="AboutModalPopupExtender"
runat="server" TargetControlID="ShowModalButton"
    PopupControlID="AboutPanel" CancelControlID="CloseButton"
BackgroundCssClass="modalPopup" />
```

7. Append a **Button** control named **CloseButton**, which does not trigger validation, set the **OnClick** event to **CloseButton_Click**, set the **Text** property to **Close**, and add two HTML line break elements to the **AboutPopupPanel** control.

   - In the About.aspx window, append the following markup within the **AboutPopupPanel** control.

```
<asp:Button ID="CloseButton" runat="server"
CausesValidation="False" OnClick="CloseButton_Click"
Text="Close" />
<br />
<br />
```

8. Add the following styles to the Styles/Site.css stylesheet.

```
div.aboutPopupPanel
{
    background-color: White;
    text-align: center;
    border: 2px solid Black;
}
.modalPopup
{
    position: relative;
    background-color: Gray;
    -ms-filter: alpha(opacity=70);
    z-index: 1;
}
.displayNone
{
    display: none;
}
```

   a. In Solution Explorer, expand **Styles**, and double-click **Site.css**.

b. In the Styles/Site.css window, press the CTRL+END keys, press ENTER, and then type the following markup.

```
div.aboutPopupPanel
{
    background-color: White;
    text-align: center;
    border: 2px solid Black;
}
.modalPopup
{
    position: relative;
    background-color: Gray;
    -ms-filter: alpha(opacity=70);
    z-index: 1;
}
.displayNone
{
    display: none;
}
```

9. Save and close the Site.css file.

   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Styles/Site.css**, or press CTRL +S.

   b. In the **Styles/Site.css** window, click the **Close** button.

▶ Task 8: Add the event handler and code

1. View the code-behind file of the About.aspx Web Form.

   - In the About.aspx window, right-click anywhere, and then click **View Code**.

2. Show the **AboutModalPopupExtender** control in the **Page.Load** event.

```
// Show about box
AboutModalPopupExtender.Show();
```

   - In the About.aspx.cs code window, add the following code to the **Page_Load** event handler.

```
// Show about box
AboutModalPopupExtender.Show();
```

3. Hide the **AboutModalPopupExtender** control in the **CloseButton_Click** event handler.

```
protected void CloseButton_Click(object sender, EventArgs e)
{
    // Close about box
    AboutModalPopupExtender.Hide();
}
```

- In the About.aspx.cs code window, add the following code to the **About** class.

```
protected void CloseButton_Click(object sender, EventArgs e)
{
    // Close about box
    AboutModalPopupExtender.Hide();
}
```

▶ Task 9: Map navigation to the About Web Form

1. Open the web.sitemap file.
   - In Solution Explorer, right-click **web.sitemap**, and then click **Open**.
2. Add an **About** menu command to the **Help** menu, making it use the **About** Web Form, by adding a new siteMapNode within the **Help** siteMapNode.
   - In the web.sitemap window, locate the **Help** siteMapNode, and then add the following markup.

```
<siteMapNode title="About" description="" url="~/About.aspx"
/>
```

3. Save and close the web.sitemap file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**, or press CTRL+S.
   b. In the web.sitemap window, click the **Close** button, or press CTRL+F4.

▶ Task 10: Test the modal About box

1. Build the solution.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Solution**, or press CTRL+SHIFT+B.
2. Run the CustomerManagement Web application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.
3. Close the **About** box.
   - In the **About Contoso Customer Management** message box, click the **Close** button.
4. Show the **About** box using the menu.
   - In the Contoso Customer Management – Windows Internet Explorer window, in the **Help** menu, click **About**.
5. Close the **About** box.
   - In the **About Contoso Customer Management** message box, click the **Close** button.
6. Close the CustomerManagement Web application.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button, or press CTRL+F4.
7. Close the About.aspx.cs code file.
   - In the About.aspx.cs window, click the **Close** button, or press CTRL+F4.
8. Close the **About** Web Form.
   - In the About.aspx window, click the **Close** button, or press CTRL+F4.

Exercise 2: Customizing Dynamic Data Field Templates with Ajax Server Controls

▶ Task 1: Add the CalendarExtender control to the DateTime_Edit field template

1. Open the **DateTime_Edit.ascx** user control Dynamic Data field template in the Source view.
   - In Solution Explorer, expand **DynamicData**, expand **FieldTemplates**, right-click **DateTime_Edit.ascx**, and then click **View Markup**.
2. Add a **CalendarExtender** control named **DateTimeEditCalendarExtender**, to the user control by using the following attribute values:
   - TargetControlID: **TextBox1**
   - PopupPosition: **Right**
   - In the DateTime_Edit.ascx window, type the following markup after the **TextBox** control.

```
<ajaxToolKit:CalendarExtender
ID="DateTimeEditCalendarExtender" runat="server"
    TargetControlID="TextBox1" PopupPosition="Right" />
```

3. Save and close the **DateTime_Edit.ascx** user control.
   a. In the **CustomerManagement** – Microsoft Visual Studio window, on the **File** menu, click **Save DynamicData/FieldTemplates /DateTime_Edit.ascx**, or press CTRL+S.
   b. In the DynamicData/FieldTemplates/DateTime_Edit.ascx window, click the **Close** button, or press CTRL+F4.

▶ Task 2: Test the DateTime_Edit field template

1. Build the solution.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Solution**.
2. Run the CustomerManagement Web application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

3. Test the **CalendarExtender** control by adding a new entry to the Customers table, click the **ModifiedDate** box, and then select a date from the calendar.
   - In the Customers – Windows Internet Explorer window, on the **Customers** menu, click **New**, click the **ModifiedDate** box, and then select a date from the calendar.
4. Cancel the new customer entry.
   - On the CUSTOMER MANAGEMENT page, click **Cancel**.
5. Close the CustomerManagement Web application.
   - In the Customers – Windows Internet Explorer window, click the **Close** button.

Exercise 3: Adding the Country Import Progress Indicator

▶ Task 1: Add the ScriptManagerProxy control

1. Open the **ImportCountries** Web Form in the Source view.
   - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View Markup**.
2. Rename the **Content** control from **Content1** to **ImportCountriesContent**.
   - In the ImportCountries.aspx window, rename the **Content** control from **Content1** to **ImportCountriesContent**.
3. Add a **ScriptManagerProxy** control named **ImportCountriesScriptManagerProxy**, to the **Content** control.

```
<asp:ScriptManagerProxy runat="server"
ID="ImportCountriesScriptManagerProxy" />
```

   - In the ImportCountries.aspx window, type the following markup,

```
<asp:ScriptManagerProxy runat="server"
ID="ImportCountriesScriptManagerProxy" />
```

   - at the top of the **ImportCountriesContent** control.

```
<asp:Content ID="ImportCountriesContent"
ContentPlaceHolderID="MainContentPlaceHolder" runat="Server">
    ...
</asp:Content>
```

▶ Task 2: Add the HTML and UI controls

1. Add an **UpdatePanel** control named **ImportCountriesUpdatePanel**, to the **ImportCountriesContent** control, with an empty **ContentTemplate** element, after the **ImportCountriesScriptManagerProxy** control.

```
<asp:UpdatePanel ID="ImportCountriesUpdatePanel" runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

- In the ImportCountries.aspx window, type the following markup,

```
<asp:UpdatePanel ID="ImportCountriesUpdatePanel"
runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

- after the **ImportCountriesScriptManagerProxy** control.

```
<asp:ScriptManagerProxy runat="server"
ID="ImportCountriesScriptManagerProxy" />
```

2. Move the existing content of the **ImportCountriesContent** control to the **ContentTemplate** element.

   a. In the ImportCountries.aspx window, select the following markup,

```
<asp:XmlDataSource ID="CountriesXmlDataSource" runat="server"
DataFile="~/Countries.xml">
</asp:XmlDataSource>
<div class="importCountriesHeader">
    <asp:Button ID="FilterButton" runat="server" Text="Filter
Countries" OnClick="FilterButton_Click" />
    <asp:Button ID="SaveButton" runat="server" Text="Save
Countries" OnClick="SaveButton_Click" />
</div>
<div class="importResult">
    <asp:Label ID="ImportResultLabel" runat="server"
EnableViewState="false"></asp:Label>
</div>
<asp:GridView ID="CountriesGridView" runat="server"
DataSourceID="CountriesXmlDataSource"
    Width="100%">
</asp:GridView>
```

   - right-click the selected text, and then click **Cut**.

   b. Place the cursor within the opening and closing tags of the **ContentTemplate** element, right-click, and then click **Paste**.

3. Format the document markup.

   - To format the markup, press CTRL+K, and then press CTRL+D.

```
<asp:UpdatePanel ID="ImportCountriesUpdatePanel"
runat="server">
    <ContentTemplate>
        <asp:XmlDataSource ID="CountriesXmlDataSource"
runat="server" DataFile="~/Countries.xml">
        </asp:XmlDataSource>
        <div class="importCountriesHeader">
            <asp:Button ID="FilterButton" runat="server"
Text="Filter Countries" OnClick="FilterButton_Click" />
            <asp:Button ID="SaveButton" runat="server"
Text="Save Countries" OnClick="SaveButton_Click" />
        </div>
        <div class="importResult">
            <asp:Label ID="ImportResultLabel" runat="server"
EnableViewState="false"></asp:Label>
        </div>
        <asp:GridView ID="CountriesGridView" runat="server"
DataSourceID="CountriesXmlDataSource"
            Width="100%">
        </asp:GridView>
    </ContentTemplate>
</asp:UpdatePanel>
```

4. Add an **UpdateProgress** control named **ImportCountriesUpdateProgress**, to the **ImportCountriesContent** control, after the **ImportCountriesScriptManagerProxy** control, by using the following attribute values:

   - AssociatedUpdatePanelID: **ImportCountriesUpdatePanel**
   - DisplayAfter: **0**

   - In the ImportCountries.aspx window, type the following markup in the **ImportCountriesContent** control, after the **ImportCountriesScriptManagerProxy** control.

```
<asp:UpdateProgress ID="ImportCountriesUpdateProgress"
runat="server"
AssociatedUpdatePanelID="ImportCountriesUpdatePanel"
DisplayAfter="0">
</asp:UpdateProgress>
```

5. Add an opening and a closing **ProgressTemplate** tag with the text, "**Processing...**" to the **ImportCountriesUpdateProgress** control.
   - In the ImportCountries.aspx window, type the following markup within the **ImportCountriesUpdateProgress** control.

```
<ProgressTemplate>
    Processing...
</ProgressTemplate>
```

6. Save and close the **ImportCountries.aspx** Web Form.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save ImportCountries.aspx**.
   b. In the ImportCountries.aspx window, click the **Close** button.

▶ Task 3: Test the update progress

1. Build the solution.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Build** menu, click **Build Solution**.
2. Debug the application.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**, or press F5.
   b. In the **Debugging Not Enabled** dialog box, click OK.
3. Filter the countries.
   a. In the Contoso Customer Management – Windows Internet Explorer window, click the **Filter Countries** button.

> **Note:** Notice that the text, "Processing...", displays in the upper-left corner of the Web site for a short period of time.

   b. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
   c. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 4: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 12

Lab Answer Key: Consuming Microsoft® Windows Communication Foundation Services (Visual Basic)

Contents:

Lab: Consuming Microsoft® Windows Communication Foundation Services

(Visual Basic)

Exercise 1: Creating a WCF Service Reference Proxy

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the 10267A-GEN-DEV virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M12\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M12\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Discover the WCF services at a specific address by using a .svc file

1. Open the **Add Service Reference** dialog box.
   - In Solution Explorer, right-click the **D:\Labfiles\Starter\M12\VB\CustomerManagement** Web site, and then click **Add Service Reference**.
2. Browse the WCF services at **http://localhost:1112/Services/Customers.svc**.
   - In the **Add Service Reference** dialog box, in the **Address** box, type **http://localhost:1112/Services/Customers.svc**, and then click **Go**.
3. Open the Customers Microsoft Windows Communication Foundation (WCF) services.

- In the **Add Service Reference** dialog box, in the **Services** pane, expand **Customers**.

4. View the operations of the **Customers** WCF service.
   - In the Services pane, click **ICustomers**.

> **Note:** Notice the single method, **GetCountries**, in the Operations pane.

5. Specify the service reference name as **CustomersService**.
   - In the **Namespace** box, type **CustomersService**, and then click **OK**.

▶ Task 3: Examine the Web reference files

1. Open the **Reference.svcmap** file, and examine the Web reference files.
   - In Solution Explorer, expand **App_WebReferences**, expand **CustomersService**, right-click **Reference.svcmap**, and then click **Open**.

> **Note:** Notice that the **MetadataFile** elements and **ExtensionFile** elements point to the Customers.wsdl, Customers.xsd, Customers1.xsd, Customers.disco, and Customers2.xsd files, and configuration91.svcinfo and configuration.svcinfo files respectively.

2. Close the Reference.svcmap file.
   - In the App_WebReferences/CustomersService/Reference.svcmap window, click the **Close** button.

3. Open the **Customers.disco** file, and examine the Web reference files.
   - In Solution Explorer, expand **Reference.svcmap**, right-click **Customers.disco**, and then click **Open**.

> **Note:** Notice the discovery file contains the **contractRef** element that link to other documents, including the Web Services Description Language (WSDL) and documentation on the server.

4. Close the Customers.disco file.
   - In the App_WebReferences/CustomersService/Customers.disco window, click the **Close** button.
5. Open the **Customers.wsdl** file, and examine the Web reference files.
   - In Solution Explorer, right-click **Customers.wsdl**, and then click **Open**.

> **Note:** View the WSDL for the WCF service, its methods, and the bindings.

6. Close the **Customers.wsdl** file.
   - In the App_WebReferences/CustomersService/Customers.wsdl window, click the **Close** button.

Exercise 2: Calling a WCF Service Method from a Web Form

▶ Task 1: Create the Customers Web Form

1. Create a folder named **Services**.
   - In Solution Explorer, right-click the **D:\Labfiles\Starter\M12\VB \CustomerManagement** Web site, click **New Folder**, type **Services**, and then press ENTER.
2. Add a Web Form named **Customers**, to the **Services** folder. The Web Form should not be based on a master page.

> **Note:** The Customers Web Form should not be based on a master page.

   a. In Solution Explorer, right-click **Services**, and then click **Add New Item**.
   b. In the **Add New Item - D:\Labfiles\Starter\M12\VB CustomerManagement**dialog box, in the middle pane, click **Web Form**.
   c. In the **Name** box, type **Customers**, ensure that the **Place code in separate file** check box is selected and the **Select master page** check box is cleared, and then click **Add**.
3. Set the page title to **Countries by Contoso**.
   - In the Services/Customers.aspx window, place the cursor between the opening and closing tags of the **title** element, and then type **Countries by Contoso**.
4. Link the Styles/Site.css stylesheet to the **Customers** Web Form.
   - In the Services/Customers.aspx window, in the **head** element, after the **title** element, place the cursor between the opening and closing tags of the **head** element.
   - Add a link element with an **href** attribute value of **~/Styles/Site.css**, a **rel** attribute with a value of **stylesheet**, and a **type** attribute with a value of **text/css**.

```
<head runat="server">
    <title>Countries by Contoso</title>
    <link href="~/Styles/Site.css" rel="stylesheet"
type="text/css" />
</head>
```

▶ Task 2: Add controls for specifying countries to return

1. Add a **Label** control named **StartingLettersLabel**, to the **div** element of the **Customers** Web Form.
    a. In the Services/Customers.aspx window, click **Design**.
    b. In the Design view, click ** **, and then point to **Toolbox**.
    c. In Toolbox, expand **Standard**, and then double-click the **Label** control.
    d. In the Properties window, in the **ID** box, type **StartingLettersLabel**.
    e. In the **Text** box, type **Starting letters of countries to return:**.
2. Add a **TextBox** control named **StartingLettersTextBox**, to the **div** element of the **Customers** Web Form, for specifying the starting letters of the countries to return.
    a. In the Services/Customers.aspx window, click the **StartingLettersLabel** control, press the RIGHT ARROW key, and then press SPACEBAR twice.
    b. In Toolbox, expand **Standard**, and then double-click the **TextBox** control.
    c. In the Properties window, in the **ID** box, type **StartingLettersTextBox**.
3. Add a **RequiredFieldValidator** control named **StartingLettersRequiredFieldValidator**, for requiring input in the **StartingLettersTextBox** control.
    a. In the Services/Customers.aspx window, click the **StartingLettersTextBox** control, and then press the RIGHT ARROW key.
    b. In Toolbox, expand **Validation**, and then double-click the **RequiredFieldValidator** control.
    c. In the Properties window, in the **ID** box, type **StartingLettersRequiredFieldValidator**.
    d. In the **ErrorMessage** box, type **The text box must contain at least one character**.
    e. In the **ControlToValidate** list, click **StartingLettersTextBox**.
    f. In the **Display** list, click **Dynamic**.

4. Add a **Button** control named **GetCountriesButton**, to the **div** element of the **Customers** Web Form. Create the **Click** event handler.

    a. In the Services/Customers.aspx window, click the **StartingLettersRequiredFieldValidator** control, press the RIGHT ARROW key, and then press SPACEBAR twice.

    b. In Toolbox, expand **Standard**, and then double-click the **Button** control.

    c. In the Properties window, in the **ID** box, type **GetCountriesButton**.

    d. In the **Text** box, type **Get Countries**.

    e. In the Properties window, click the **Events** button, and then double-click the **Click** box.

▶ Task 3: Add a GridView control

1. Add a **GridView** control named **CustomersGridView**, to the **div** element of the **Customers** Web Form, using a style of **DDGridView**.

    a. In the CustomerManagement − Microsoft Visual Studio window, click **Services/Customers.aspx**.

    b. In the Services/Customers.aspx window, click the **Get Countries** button, press the RIGHT ARROW key, and then press ENTER.

    c. In the Toolbox, expand **Data**, and then double-click the **GridView** control.

    d. In the Properties window of the **GridView** control, in the **ID** box, type **CustomersGridView**.

    e. In the **CssClass** list, click **DDGridView**.

2. Save the **Customers** Web Form.

    - In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Services/Customers.aspx**.

3. Close the **Customers** Web Form.

    - In the Services/Customers.aspx window, click the **Close** button.

▶ Task 4: Call the WCF service

1. Import the **CustomersService** namespace to the **Customers** Web Form.
   - In the Services/Customers.aspx.vb window, add the **CustomersService** namespace import at the top of the file.

```
Imports CustomersService
```

2. In the **GetCountriesButton.Click** event, declare and instantiate an instance of the **CustomersClient** class, named **customersService**.
   - In the Services/Customers.aspx.vb window, type the following code in the **GetCountriesButton_Click** event handler.

```
Dim customersService As New CustomersClient()
```

3. Call the **GetCountries** WCF service method, passing the content of the **Text** property of the **StartingLettersTextBox TextBox** control, and assign the return value to the **DataSource** property of the **CustomersGridView** control.
   - In the Services/Customers.aspx.vb window, append the following code to the **GetCountriesButton_Click** event handler.

```
CustomersGridView.DataSource =
customersService.GetCountries(StartingLettersTextBox.Text)
```

4. Implement data binding for the **CustomersGridView** control.
   - In the Services/Customers.aspx.vb window, append the following code to the **GetCountriesButton_Click** event handler.

```
CustomersGridView.DataBind()
```

5. Save the code-behind file of the **Customers** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Services/Customers.aspx.vb**.

▶ Task 5: Test the Customers Web Form

1. View the **Customers** Web Form in the browser.
   - In Solution Explorer, expand **Services**, right-click **Customers.aspx**, and then click **View in Browser**.
2. Get all countries beginning with the letter **a**.
   - In the text box, type **a**, and then click the **Get Countries** button.
3. Close the Windows® Internet Explorer® browser.
   - In the Countries by Contoso – Windows Internet Explorer window, click the **Close** button.
4. Close the **Customers** Web Form.
   - In the Services/Customers.aspx window, click the **Close** button.

Exercise 3: Implementing WCF Data Services

▶ Task 1: Add a WCF Data Service

1. Add a new WCF Data Service in the **Services** folder named **CustomersWcfDS**, by using the **Add New Item** dialog box and the **WCF Data Service** template.
   a. In Solution Explorer, right-click **Services**, and then click **Add New Item**.
   b. In the **Add New Item – D:\Labfiles\Starter\M12\VB \CustomerManagement** dialog box, in the middle pane, click **WCF Data Service**.
   c. In the **Name** box, type **CustomersWcfDS.svc**, and then click **Add**.
2. Add the existing EDM namespace and class name to the declaration of the generic **CustomerManagementModel.Entities**.

```
Public Class CustomersWcfDS
    Inherits DataService(Of CustomerManagementModel.Entities)
```

   - In the Services/App_Code/CustomersWcfDS.vb window, modify the following class definition code from,

```
Public Class CustomersWcfDS
    ' TODO: replace [[class name]] with your data class name
    Inherits DataService(Of [[class name]])
```

     to:

```
Public Class CustomersWcfDS
    Inherits DataService(Of CustomerManagementModel.Entities)
```

3. Allow read access to the **Countries** and **Customers** entities of the EDM by using the **IDataServiceConfiguration.SetEntitySetAccessRule** method in the Shared **InitializeService** method of the **DataService** class.

```
config.SetEntitySetAccessRule("Countries",
EntitySetRights.AllRead)
config.SetEntitySetAccessRule("Customers",
EntitySetRights.AllRead)
```

- In the Services/App_Code/CustomersWcfDS.vb window, replace the following code of the class definition,

```
' TODO: set rules to indicate which entity sets and service
operations are visible, updatable, etc.
' Examples:
' config.SetEntitySetAccessRule("MyEntityset",
EntitySetRights.AllRead)
' config.SetServiceOperationAccessRule("MyServiceOperation",
ServiceOperationRights.All)
```

  with:

```
config.SetEntitySetAccessRule("Countries",
EntitySetRights.AllRead)
config.SetEntitySetAccessRule("Customers",
EntitySetRights.AllRead)
```

4. Save and close the **CustomersWcfDS** code-behind file.
   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Services/App_Code/CustomersWcfDS.vb**.
   b. In the Services/App_Code/CustomersWcfDS.vb window, click the **Close** button.
5. Move the **CustomersWcfDS** code-behind file from the **Services\App_Code** folder to the **App_Code** folder in the root folder.
   - In Solution Explorer, in the **Services** folder, under **App_Code**, drag the **CustomersWcfDS.vb** file to the **App_Code** folder in the root folder.
6. Delete the **App_Code** folder from the **Services** folder.
   a. In Solution Explorer, under **Services**, right-click **App_Code**, and then click **Delete**.
   b. In the Microsoft Visual Studio message box, click **OK**.

▶ Task 2: Test the Data Service

1. Run the application.
   a. In Solution Explorer, click **Default.aspx**.
   b. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.
2. View the entities available with the Customers Data Service by browsing to the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc**.
   - In the Address bar of Internet Explorer, type **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc**, and then press ENTER.

> **Note:** Notice that the XML that was returned from the service contains both the **Countries** and **Customers** entities.

3. View the Countries entity by browsing to the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries**.
   - In the Address bar of Internet Explorer, type **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries**, and then press ENTER.

> **Note:** Notice that the XML that was returned from the service contains all of the Countries from the data model.

4. View the Customers entity by browsing to the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Customers**.
   - In the Address bar of Internet Explorer, type **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Customers**, and then press ENTER.

> **Note:** Notice the XML that was returned from the service contains all of the Customers from the data model.

5. View the information for the country, **India**, by browsing to the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=Name eq 'India'**.
   - In the Address bar of Internet Explorer, type **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=Name eq 'India'**, and then press ENTER.

> **Note:** Notice the XML that was returned from the service contains a single country, India.

6. View the countries for which the **InternetTLD** starts with less than B by using the URL, **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=InternetTLD lt 'B'**.
   - In the Address bar of Internet Explorer, type **http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=InternetTLD lt 'B'**, and then press ENTER.

> **Note:** Notice the XML returned from the service contains all countries for which the **InternetTLD** starts with less than B.

7. Close Internet Explorer.
   - In the http://localhost:1111/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=InternetTLD – Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 12

Lab Answer Key: Consuming Microsoft® Windows Communication Foundation Services (Visual C#)

Contents:

Lab: Consuming Microsoft® Windows Communication Foundation Services

(C#)

Exercise 1: Creating a WCF Service Reference Proxy

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M12\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M12\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Discover the WCF services at a specific address by using a .svc file

1. Open the **Add Service Reference** dialog box.
   - In Solution Explorer, right-click the **D:\Labfiles\Starter\M12\CS\CustomerManagement** Web site, and then click **Add Service Reference**.
2. Browse the WCF services at **http://localhost:1112/Services/Customers.svc**.
   - In the **Add Service Reference** dialog box, in the **Address** box, type **http://localhost:1112/Services/Customers.svc**, and then click **Go**.
3. Open the Customers Microsoft Windows Communication Foundation (WCF) services.

- In the **Add Service Reference** dialog box, in the **Services** pane, expand **Customers**.

4. View the operations of the **Customers** WCF service.
   - In the Services pane, click **ICustomers**.

> **Note:** Notice the single method, **GetCountries**, in the **Operations** pane.

5. Specify the service reference name as **CustomersService**.
   - In the **Namespace** box, type **CustomersService**, and then click **OK**.

▶ Task 3: Examine the Web reference files

1. Open the **Reference.svcmap** file, and examine the Web reference files.
   - In Solution Explorer, expand **App_WebReferences**, expand **CustomersService**, right-click **Reference.svcmap**, and then click **Open**.

> **Note:** Notice that the MetadataFile elements and ExtensionFile elements point to the Customers.wsdl, Customers.xsd, Customers1.xsd, Customers.disco, and Customers2.xsd files, and configuration91.svcinfo and configuration.svcinfo files respectively.

2. Close the Reference.svcmap file.
   - In the App_WebReferences/CustomersService/Reference.svcmap window, click the **Close** button.

3. Open the **Customers.disco** file, and examine the Web reference files.
   - In Solution Explorer, expand **Reference.svcmap**, right-click **Customers.disco**, and then click **Open**.

> **Note:** Notice the discovery file contains the contractRef element that link to other documents, including the Web Services Description Language (WSDL) and documentation on the server.

4. Close the Customers.disco file.
   - In the App_WebReferences/CustomersService/Customers.disco window, click the **Close** button.

5. Open the **Customers.wsdl** file, and examine the Web reference files.
   - In Solution Explorer, right-click **Customers.wsdl**, and then click **Open**.

> **Note:** View the WSDL for the WCF service, its methods, and the bindings.

6. Close the Customers.wsdl file.
   - In the App_WebReferences/CustomersService/Customers.wsdl window, click the **Close** button.

Exercise 2: Calling a WCF Service Method from a Web Form

▶ Task 1: Create the Customers Web Form

1. Create a folder named **Services**.
   - In Solution Explorer, right-click the **D:\Labfiles\Starter\M12\CS\CustomerManagement** Web site, click **New Folder**, type **Services**, and then press ENTER.
2. Add a Web Form named **Customers**, to the **Services** folder. The Web Form should not be based on a master page.

> **Note:** The **Customers** Web Form should not be based on a master page.

   a. In Solution Explorer, right-click **Services**, and then click **Add New Item**.
   b. In the **Add New Item - D:\Labfiles\Starter\M12\CS\CustomerManagement**dialog box, in the middle pane, click **Web Form**.
   c. In the **Name** box, type **Customers**, ensure that the **Place code in separate file** check box is selected and the **Select master page** check box is cleared, and then click **Add**.
3. Set the page title to **Countries by Contoso**.
   - In the Services/Customers.aspx window, place the cursor between the opening and closing tags of the **title** element, and then type **Countries by Contoso**.
4. Link the Styles/Site.css stylesheet to the **Customers** Web Form.
   - In the Services/Customers.aspx window, in the **head** element, after the **title** element, place the cursor between the opening and closing tags of the **head** element.

- Add a link element with an **href** attribute value of **~/Styles/Site.css**, a **rel** attribute with a value of **stylesheet**, and a **type** attribute with a value of **text/css**.

```
<head runat="server">
    <title>Countries by Contoso</title>
    <link href="~/Styles/Site.css" rel="stylesheet"
type="text/css" />
</head>
```

▶ Task 2: Add controls for specifying countries to return

1. Add a **Label** control named **StartingLettersLabel**, to the **div** element of the **Customers** Web Form.
   a. In the Services/Customers.aspx window, click **Design**.
   b. In the Design view, click ** **, and then point to **Toolbox**.
   c. In Toolbox, expand **Standard**, and then double-click the **Label** control.
   d. In the Properties window, in the **ID** box, type **StartingLettersLabel**.
   e. In the **Text** box, type **Starting letters of countries to return:**.
2. Add a **TextBox** control named **StartingLettersTextBox**, to the **div** element of the **Customers** Web Form, for specifying the starting letters of the countries to return.
   a. In the Services/Customers.aspx window, click the **StartingLettersLabel** control, press the RIGHT ARROW key, and then press SPACEBAR twice.
   b. In Toolbox, expand **Standard**, and then double-click the **TextBox** control.
   c. In the Properties window, in the **ID** box, type **StartingLettersTextBox**.
3. Add a **RequiredFieldValidator** control named **StartingLettersRequiredFieldValidator**, for requiring input in the **StartingLettersTextBox** control.
   a. In the Services/Customers.aspx window, click the **StartingLettersTextBox** control, and then press the RIGHT ARROW key.
   b. In Toolbox, expand **Validation**, and then double-click the **RequiredFieldValidator** control.
   c. In the Properties window, in the **ID** box, type **StartingLettersRequiredFieldValidator**.

d. In the **ErrorMessage** box, type **The text box must contain at least one character.**

e. In the **ControlToValidate** list, click **StartingLettersTextBox**.

f. In the **Display** list, click **Dynamic**.

4. Add a **Button** control named **GetCountriesButton**, to the **div** element of the **Customers** Web Form. Create the **Click** event handler.

a. In the Services/Customers.aspx window, click the **StartingLettersRequiredFieldValidator** control, press the RIGHT ARROW key, and then press SPACEBAR twice.

b. In Toolbox, expand **Standard**, and then double-click the **Button** control.

c. In the Properties window, in the **ID** box, type **GetCountriesButton**.

d. In the **Text** box, type **Get Countries**.

e. In the Properties window, click the **Events** button, and then double-click the **Click** box.

▶ Task 3: Add a GridView control

1. Add a **GridView** control named **CustomersGridView**, to the **div** element of the **Customers** Web Form, using a style of **DDGridView**.

a. In the CustomerManagement – Microsoft Visual Studio window, click **Services/Customers.aspx**.

b. In the Services/Customers.aspx window, click the **Get Countries** button, press the RIGHT ARROW key, and then press ENTER.

c. In the Toolbox, expand **Data**, and then double-click the **GridView** control.

d. In the Properties window of the **GridView** control, in the **ID** box, type **CustomersGridView**.

e. In the **CssClass** list, click **DDGridView**.

2. Save the **Customers** Web Form.

- In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Services/Customers.aspx**.

3. Close the **Customers** Web Form.

- In the Services/Customers.aspx window, click the **Close** button.

▶ Task 4: Call the WCF service

1. Import the **CustomersService** namespace to the **Customers** Web Form.
   - In the Services/Customers.aspx.cs window, add the **CustomersService** namespace import above the import of the **System** namespace.

```
using CustomersService;
```

2. In the **GetCountriesButton.Click** event, declare and instantiate an instance of the **CustomersClient** class, named **customersService**.
   - In the Services/Customers.aspx.cs window, type the following code in the **GetCountriesButton_Click** event handler.

```
CustomersClient customersService = new CustomersClient();
```

3. Call the **GetCountries** WCF service method, passing the content of the **Text** property of the **StartingLettersTextBox TextBox** control, and assign the return value to the **DataSource** property of the **CustomersGridView** control.
   - In the Services/Customers.aspx.cs window, append the following code to the **GetCountriesButton_Click** event handler.

```
CustomersGridView.DataSource =
customersService.GetCountries(StartingLettersTextBox.Text);
```

4. Implement data binding for the **CustomersGridView** control.
   - In the Services/Customers.aspx.cs window, append the following code to the **GetCountriesButton_Click** event handler.

```
CustomersGridView.DataBind();
```

5. Save the code-behind file of the **Customers** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Services/Customers.aspx.cs**.

▶ Task 5: Test the Customers Web Form

1. View the **Customers** Web Form in the browser.
   - In Solution Explorer, expand **Services**, right-click **Customers.aspx**, and then click **View in Browser**.
2. Get all countries beginning with the letter **a**.
   - In the text box, type **a**, and then click the **Get Countries** button.
3. Close the Windows® Internet Explorer® browser.
   - In the Countries by Contoso – Windows Internet Explorer window, click the **Close** button.
4. Close the **Customers** Web Form.
   - In the Services/Customers.aspx window, click the **Close** button.

Exercise 3: Implementing WCF Data Services

▶ Task 1: Add a WCF Data Service

1. Add a new WCF Data Service in the **Services** folder named **CustomersWcfDS**, by using the **Add New Item** dialog box and the **WCF Data Service** template.
   a. In Solution Explorer, right-click **Services**, and then click **Add New Item**.
   b. In the **Add New Item – D:\Labfiles\Starter\M12\CS\CustomerManagement** dialog box, in the middle pane, click **WCF Data Service**.
   c. In the **Name** box, type **CustomersWcfDS.svc**, and then click **Add**.
2. Add the existing EDM namespace and class name to the declaration of the generic **Customers** class, **CustomerManagementModel.Entities**.

```
public class CustomersWcfDs :
DataService<CustomerManagementModel.Entities>
```

   - In the Services/App_Code/CustomersWcfDS.cs window, modify the following class definition code from,

```
public class CustomersWcfDS : DataService< /* TODO: put your
data source class name here */ >
```

     to:

```
public class CustomersWcfDS :
DataService<CustomerManagementModel.Entities>
```

3. Allow read access to the **Countries** and **Customers** entities of the EDM by using the **IDataServiceConfiguration.SetEntitySetAccessRule** method in the Shared **InitializeService** method of the **DataService** class.

```
config.SetEntitySetAccessRule("Countries",
EntitySetRights.AllRead);
config.SetEntitySetAccessRule("Customers",
EntitySetRights.AllRead);
```

- In the Services/App_Code/CustomersWcfDS.cs window, replace the following code of the class definition,

```
// TODO: set rules to indicate which entity sets and service
operations are visible, updatable, etc.
// Examples:
// config.SetEntitySetAccessRule("MyEntityset",
EntitySetRights.AllRead);
// config.SetServiceOperationAccessRule("MyServiceOperation",
ServiceOperationRights.All);
```

  with:

```
config.SetEntitySetAccessRule("Countries",
EntitySetRights.AllRead);
config.SetEntitySetAccessRule("Customers",
EntitySetRights.AllRead);
```

4. Save and close the **CustomersWcfDS** code-behind file.

   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Services/App_Code/CustomersWcfDS.cs**.

   b. In the Services/App_Code/CustomersWcfDS.cs window, click the **Close** button.

5. Move the **CustomersWcfDS** code-behind file from the **Services\App_Code** folder to the **App_Code** folder in the root folder.

   - In Solution Explorer, in the **Services** folder, under **App_Code**, drag the **CustomersWcfDS.cs** file to the **App_Code** folder in the root folder.

6. Delete the **App_Code** folder from the **Services** folder.

   a. In Solution Explorer, under **Services**, right-click **App_Code**, and then click **Delete**.

   b. In the Microsoft Visual Studio message box, click **OK**.

▶ Task 2: Test the Data Service

1. Run the application.
   a. In Solution Explorer, click **Default.aspx**.
   b. In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.
2. View the entities available with the Customers Data Service by browsing to the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc**.
   - In the Address bar of Internet Explorer, type **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc**, and then press ENTER.

> **Note:** Notice that the XML that was returned from the service contains both the **Countries** and **Customers** entities.

3. View the Countries entity by browsing to the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries**.
   - In the Address bar of Internet Explorer, type **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries**, and then press ENTER.

> **Note:** Notice that the XML that was returned from the service contains all of the Countries from the data model.

4. View the Customers entity by browsing to the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Customers**.
   - In the Address bar of Internet Explorer, type **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Customers**, and then press ENTER.

> **Note:** Notice the XML that was returned from the service contains all of the Customers from the data model.

5. View the information for the country, **India**, by browsing to the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=Name eq 'India'**.
   - In the Address bar of Internet Explorer, type **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=Name eq 'India'**, and then press ENTER.

> **Note:** Notice the XML that was returned from the service contains a single country, India.

6. View the countries for which the InternetTLD starts with less than B by using the URL, **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=InternetTLD lt 'B'**.
   - In the Address bar of Internet Explorer, type **http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=InternetTLD lt 'B'**, and then press ENTER.

> **Note:** Notice the XML that was returned from the service contains all countries for which the **InternetTLD** starts with less than B.

7. Close Internet Explorer.
   - In the http://localhost:1110/CustomerManagement/Services/CustomersWcfDS.svc/Countries?$filter=InternetTLD – Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 13

Lab Answer Key: Managing State in Web Applications (Visual Basic)

Contents:

Lab: Managing State in Web Applications

(Visual Basic)

Exercise 1: Examining the View State

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M13\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M13\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.
  - In Solution Explorer, right-click **Default.aspx**, and then click **Set As Start Page**.

▶ Task 3: Notice the View state size

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

2. View all customers.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **All**.

Note: Notice that the **GridView** control displays with the customers in the database.

3. View the page source by using the **View Source** option.
   - On the CUSTOMER MANAGEMENT page, right-click anywhere, and then click **View Source**.

Note: You can also view the page source by using the **View Source** option on the **Page** menu.

Note: The built-in viewer of the Developer Tools in Windows® Internet Explorer® 8 opens in the http://localhost:1111/CustomerManagement /Customers/List.aspx – Original Source window.

4. Locate the page **View** state in the hidden field named **__VIEWSTATE**, and view the space occupied by the **__VIEWSTATE** element.
   - In the http://localhost:1111/CustomerManagement/Customers /List.aspx – Original Source window, scroll down to the HTML input element of type hidden, named **__VIEWSTATE**.

Note: Notice that about 10–15 percent of the space of the http://localhost:1111/CustomerManagement/Customers/List.aspx – Original Source window is occupied by the **__VIEWSTATE** element.

5. Close the built-in viewer of the Internet Explorer 8 Developer Tools.
   - In the http://localhost:1111/CustomerManagement/Customers /List.aspx − Original Source window, click the **Close** button.
6. Close Internet Explorer.
   - In the Customers − Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Disable the View state

1. In Solution Explorer, navigate to **DynamicData\PageTemplates**, and then open the **List.aspx** Web Form in the Source view.
   - In Solution Explorer, expand **DynamicData**, expand **PageTemplates**, right-click **List.aspx**, and then click **View Markup**.
2. Disable the **View** state for the **List** Web Form by using the **EnableViewState** page attribute.
   - In the DynamicData/PageTemplates/List.aspx window, add the **EnableViewState** attribute, and in the **Page** directive, set the value to **false**.

```
<%@ Page Language="VB" MasterPageFile="~/Site.master"
CodeFile="List.aspx.vb" Inherits="List"
EnableViewState="false" %>
```

3. Save the **List** Web Form.
   - In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save DynamicData/PageTemplates/List.aspx**.

▶ Task 5: Notice the View state size

1. Run the application.
   - In the CustomerManagement − Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.
2. View all customers, and note that there is no change in the number of customers.
   - In the Contoso Customer Management − Windows Internet Explorer window, on the **Customers** menu, click **All**.

Note: Notice that the **GridView** control displays with the same number of customers.

3. View the page source by using the **View Source** option.
   - On the CUSTOMER MANAGEMENT page, right-click anywhere, and then click **View Source**.

Note: You can also view the page source by using the **View Source** option on the **Page** menu.

Note: The Developer Tools built-in viewer in Internet Explorer 8 opens in the http://localhost:1111/CustomerManagement/Customers/List.aspx – Original Source window.

4. Locate the page **View** state in the hidden field named **__VIEWSTATE**, and view the space occupied by the **__VIEWSTATE** element.
   - In the http://localhost:1111/CustomerManagement/Customers /List.aspx – Original Source window, scroll down to the HTML input element of type hidden, named **__VIEWSTATE**.

Note: Notice that about 6–8 percent of the space of the http://localhost:1111 /CustomerManagement/Customers/List.aspx – Original Source window is occupied by the **__VIEWSTATE** element.

5. Close the built-in viewer of the Internet Explorer 8 Developer Tools.
   - In the http://localhost:1111/CustomerManagement/Customers /List.aspx – Original Source window, click the **Close** button.
6. Close Internet Explorer.
   - In the Customers – Windows Internet Explorer window, click the **Close** button.

▶ Task 6: Enable the View state

1. Enable the **View** state for the **List** Web Form by removing the **EnableViewState** page attribute.

   - In the DynamicData/PageTemplates/List.aspx window, remove the **EnableViewState** attribute from the **Page** directive.

     ```
     <%@ Page Language="VB" MasterPageFile="~/Site.master"
     CodeFile="List.aspx.vb" Inherits="List" %>
     ```

2. Save and close the **List** Web Form.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save DynamicData/PageTemplates/List.aspx**.

   b. In the DynamicData/PageTemplates/List.aspx window, click the **Close** button.

Exercise 2: Caching the Countries

▶ Task 1: Cache a DataTable

1. In the **Services_Customers** class, create a private member variable named **countryDataTable**, of type **System.Data.DataTable**, to hold the countries retrieved from the database and initialize the variable to **Nothing**.

   a. In Solution Explorer, expand **Services**, right-click **Customers.aspx**, and then click **View Code**.

   b. In the Services/Customers.aspx.vb window, add the following code to the **Services_Customers** class, after the class signature.

   ```
   Private countryDataTable As System.Data.DataTable = Nothing
   ```

2. Assign the **Countries** cache item value to the **countryDataTable** private variable, casting it to the data type **System.Data.DataTable**, upon page load.

   - Type the following code in to the **Services_Customers** class.

   ```
   Protected Sub Page_Load(ByVal sender As Object, ByVal e As
   System.EventArgs) Handles Me.Load
       ' Retrieve DataTable from cache
       countryDataTable =
           CType(Cache("Countries"), System.Data.DataTable)
   End Sub
   ```

3. Check whether an item has been retrieved from the cache by comparing the **countryDataTable** variable with a null value, in an **If** construct, when the user clicks the **Get Countries** button.

   - Add the following code at the top of the **GetCountriesButton_Click** event handler.

   ```
   ' Does cached item exist?
   If countryDataTable Is Nothing Then

   End If
   ```

4. Assign the value of the **countryDataTable** local variable to the **DataSource** property of the **CustomersGridView** control.
   - After the **If** construct, type the following code.

```
' Set GridView DataSource
CustomersGridView.DataSource = countryDataTable
```

5. If an item is not retrieved from the cache, move the existing line of code from after the **If** construct that declares and instantiates the local **customersService** variable.
   a. Locate and select the following code,

```
Dim customersService As New CustomersClient()
```

   right-click the selection, and then click **Cut**.
   b. Place the cursor within the **If** construct, right-click, and then click **Paste**.
6. If an item is not retrieved from the cache, assign the result of the call to the **GetCountries** Windows Communication Foundation (WCF) service method, to the local **countryDataTable** variable.
   - In the **If** construct, append the following code after the line that declares and instantiates the local **customersService** variable.

```
' Retrieve DataTable from WCF Service
countryDataTable =
customersService.GetCountries(StartingLettersTextBox.Text)
```

7. Add a new variable named **Countries** to the cache object, and assign the value of the local **countryDataTable** variable.
   - In the **If** construct, append the following code after the line that assigns the result of the call to the **GetCountries** WCF service method.

```
' Save DataTable to cache
Cache("Countries") = countryDataTable
```

8. Delete the existing line of code that assigns the result of the call to the **GetCountries** WCF service method to the **DataSource** property of the **CustomersGridView** control.
   - In the Services/Customers.aspx.vb window, delete the following line of code from the **GetCountriesButton_Click** event handler.

```
CustomersGridView.DataSource =
customersService.GetCountries(StartingLettersTextBox.Text)
```

9. Save the changes to the Customers.aspx.vb file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Services/Customers.aspx.vb**.

▶ Task 2: Test the cache storage

1. Set a breakpoint on the line that retrieves the **Countries** item from the **Cache** object.
   - In the **GetCountriesButton_Click** event handler, place the cursor at the beginning of the following code, and then press the F9 key.

```
If countryDataTable Is Nothing Then
```

2. Run the application in the debug mode.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**, or press F5.
3. View the **Customers** Web Form by typing the URL **http://localhost:1111/CustomerManagement/Services/Customers.aspx** in the Address bar.
   - In the Address bar of the Contoso Customer Management – Windows Internet Explorer window, type **http://localhost:1111/CustomerManagement/Services/Customers.aspx**, and then press ENTER.
4. Retrieve the countries starting with the letter **b**.
   - In the Countries by Contoso – Windows Internet Explorer window, in the text box, type **b**, and then click **Get Countries**.

> **Note:** The Customers Web Form code now opens in the debugger, and the current line of code will now check whether you have retrieved an item from the cache.

5. Press F10 to go to the next statement.

> **Note:** You have not retrieved an item from the cache, so you need to create and store an item in the cache.

6. Press F5 to continue running the application.

> **Note:** Notice that the countries now display on the Web page.

7. Close Internet Explorer.
   - In the Countries by Contoso – Windows Internet Explorer window, click the **Close** button.
8. Run the application in the debug mode.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**, or press F5.
9. View the **Customers** Web Form in browser by using the URL **http://localhost:1111/CustomerManagement/Services/Customers.aspx**.
   - In the Address bar of the Contoso Customer Management – Windows Internet Explorer window, type **http://localhost:1111/CustomerManagement/Services/Customers.aspx**, and then press ENTER.
10. Retrieve the countries starting with the letter **c**.
    - In the Countries by Contoso – Windows Internet Explorer window, in the text box, type **c**, and then click **Get Countries**.

> **Note:** The Customers Web Form code now opens in the debugger, and the current line of code will now check whether you have retrieved an item from the cache.

11. Press F10 to go to the next statement.

> **Note:** You have retrieved an item from the cache. Now, you need to assign the retrieved item to the **CustomersGridView** data source property.

12. Press F5 to continue running the application.

> **Note:** Notice that the countries display on the Web page.

13. Close Internet Explorer.
    - In the Countries by Contoso – Windows Internet Explorer window, click the **Close** button.
14. Remove the breakpoint from the code that retrieves the **Countries** item from the **Cache** object.
    - In the **GetCountriesButton_Click** event handler, place the cursor at the beginning of the following line of code, and the press F9.

```
If countryDataTable Is Nothing Then
```

15. Close the Customers.aspx.vb file.
    - In the Services/Customers.aspx.vb window, click the **Close** button.

Exercise 3: Displaying a Visitors Counter on the Default Page

▶ Task 1: Initialize an application variable

1. Add a variable named **Visitors**, with a value of **0**, to the **Application** state.
   a. In Solution Explorer, right-click **Global.asax**, and then click **Open**.
   b. In the Global.asax window, append the following code to the **Application_Start** method, after the code that calls the **Register** method.

```
' Save application variable
Application("Visitors") = 0
```

2. Save the Global.asax file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Global.asax**, or press CTRL+S.

▶ Task 2: Add a visitors counter to the default page

1. Add a **div** element with a **class** attribute set to the value of **footer** to the **MainContent** control of the **Default** Web Form.
   a. In Solution Explorer, right-click **Default.aspx**, and then click **View Markup**.
   b. In the Default.aspx window, type the following markup within the **MainContent** control.

```
<div class="footer">
</div>
```

2. Add a **Literal** control named **VisitorLiteral**, to the **div** element.
   - In the Default.aspx window, type the following markup within the **div** element.

```
<asp:Literal id="VisitorLiteral" runat="server" />
```

3. Save and close the **Default** Web Form.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, click the **Close** button.

4. Add the following style to the Styles/Site.css stylesheet.

```
div.footer
{
    background: #dbddff;
    text-align: center;
    height: 15px; /* Height of the footer */
    width: 100%;
    position: fixed;
    bottom: 0;
}
```

   a. In Solution Explorer, expand **Styles**, and double-click **Site.css**.

   b. In the Styles/Site.css window, press the CTRL+END keys, press ENTER, and then type the following markup.

```
div.footer
{
    background: #dbddff;
    text-align: center;
    height: 15px; /* Height of the footer */
    width: 100%;
    position: fixed;
    bottom: 0;
}
```

5. Save and close the Site.css file.

   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Styles/Site.css**, or press CTRL+S.

   b. In the Styles/Site.css window, click the **Close** button.

6. In the **Page_Load** event handler of the **Default** Web Form, create a local variable named **numVisitors**, of type **Long**, to hold the number of visitors and initialize the variable with a value of **0**.

   a. In Solution Explorer, right-click **Default.aspx**, and then click **View Code**.

   b. In the Default.aspx.vb window, type the following code in the **_Default** class.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    Dim numVisitors As Long = 0
End Sub
```

7. In the **Page_Load** event handler, check if the **Visitors** Application variable exists, and if so, assign the value to **numVisitors** local variable, casting it to data type **long**.

   - In the Default.aspx.vb window, append the following code in the **Page_Load** event handler, after the declaration and initialization of the **numVisitors** variable.

```
' Check if Application variable exists
If Not Application("Visitors") Is Nothing Then
    numVisitors =
Long.Parse(Application("Visitors").ToString())
End If
```

8. In the **Page_Load** event handler, assign the text "**Number of visitors:** ", and the value of the **numVisitors** variable, to the **Text** property of the **VisitorLiteral** control.

> **Note:** The **Visitors** application variable must be converted by using the **Parse** method of the **Long** data type.

   - In the Default.aspx.vb window, append the following code in the **Page_Load** event handler, after the code that checks for the application variable existence.

```
VisitorLiteral.Text = "Number of visitors: " &
numVisitors.ToString()
```

9. Save and close the Default.aspx.vb file.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Default.aspx.vb**.

   b. In the Default.aspx.vb window, click the **Close** button.

▶ Task 3: Update the visitors counter on a new session

1. Each time a new user session is started, increment the **Visitors** application variable by **1**, by appending the following code to the **Session_Start** method in the Global.asax file.

   - In Global.asax window, add the following code in the **Session_Start** method.

```
' Increment Visitors counter
Application("Visitors") =
    Long.Parse(Application("Visitors").ToString()) + 1
```

> **Note:** The Visitors application variable must be converted by using the Parse method of the data type.

2. Save and close the Global.asax file.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Global.asax**.

   b. In the Global.asax window, click the **Close** button.

▶ Task 4: Test the visitors counter

1. View the **Default** Web Form in a browser.

> **Note:** Notice that the Number of visitors is set to **1**.

   - In Solution Explorer, right-click **Default.aspx**, and then click **View in Browser**.

2. Close Internet Explorer.

   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

3. View the **Default** Web Form in a browser.

> **Note:** Notice that the Number of visitors is set to **2**.

   - In Solution Explorer, right-click **Default.aspx**, and then click **View in Browser**.

4. Close Internet Explorer.
   a. In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
   b. In the CustomerManagement Microsoft Visual Studio window, click the **Close** button.

▶ Task 5: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 13

Lab Answer Key: Managing State in Web Applications (Visual C#)

Contents:

Lab: Managing State in Web Applications

(C#)

Exercise 1: Examining the View State

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M13\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M13\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.
  - In Solution Explorer, right-click **Default.aspx**, and then click **Set As Start Page**.

▶ Task 3: Notice the View state size

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

2. View all customers.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **All**.

Note: Notice that the **GridView** control displays with the customers in the database.

3. View the page source by using the **View Source** option.
   - On the CUSTOMER MANAGEMENT page, right-click anywhere, and then click **View Source**.

Note: You can also view the page source by using the **View Source** option on the **Page** menu.

Note: The built-in viewer of the Developer Tools in Windows® Internet Explorer® 8 opens in the http://localhost:1110/CustomerManagement /Customers/List.aspx – Original Source window.

4. Locate the page **View** state in the hidden field named **__VIEWSTATE**, and view the space occupied by the **__VIEWSTATE** element.
   - In the http://localhost:1110/CustomerManagement/Customers /List.aspx – Original Source window, scroll down to the HTML input element of type hidden, named **__VIEWSTATE**.

Note: Notice that about 10–15 percent of the space of the http://localhost:1110 /CustomerManagement/Customers/List.aspx – Original Source window is occupied by the **__VIEWSTATE** element.

5. Close the built-in viewer of the Internet Explorer 8 Developer Tools.
   - In the http://localhost:1110/CustomerManagement/Customers /List.aspx – Original Source window, click the **Close** button.

6. Close Internet Explorer.
   - In the Customers – Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Disable the View state

1. In Solution Explorer, navigate to **DynamicData\PageTemplates**, and then open the **List.aspx** Web Form in the Source view.
   - In Solution Explorer, expand **DynamicData**, expand **PageTemplates**, right-click **List.aspx**, and then click **View Markup**.
2. Disable the **View** state for the **List** Web Form by using the **EnableViewState** page attribute.
   - In the DynamicData/PageTemplates/List.aspx window, add the **EnableViewState** attribute and in the **Page** directive, set the value to **false**.

```
<%@ Page Language="C#" MasterPageFile="~/Site.master"
CodeFile="List.aspx.cs" Inherits="List"
EnableViewState="false" %>
```

3. Save the **List** Web Form.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save DynamicData/PageTemplates/List.aspx**.

▶ Task 5: Notice the View state size

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.
2. View all customers, and note that there is no change in the number of customers.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Customers** menu, click **All**.

Note: Notice that the **GridView** control displays with the same number of customers.

3. View the page source by using the **View Source** option.
   - On the CUSTOMER MANAGEMENT page, right-click anywhere, and then click **View Source**.

> **Note:** You can also view the page source by using the **View Source** option on the Page menu.

> **Note:** The Developer Tools built-in viewer in Internet Explorer 8 opens in the http://localhost:1110/CustomerManagement/Customers/List.aspx – Original Source window.

4. Locate the page **View** state in the hidden field named **__VIEWSTATE**, and view the space occupied by the **__VIEWSTATE** element.
   - In the http://localhost:1110/CustomerManagement/Customers /List.aspx – Original Source window, scroll down to the HTML input element of type hidden, named **__VIEWSTATE**.

> **Note:** Notice that about 6–8 percent of the space of the http://localhost:1110 /CustomerManagement/Customers/List.aspx – Original Source window is occupied by the **__VIEWSTATE** element.

5. Close the built-in viewer of the Internet Explorer 8 Developer Tools.
   - In the http://localhost:1110/CustomerManagement/Customers /List.aspx – Original Source window, click the **Close** button.
6. Close Internet Explorer.
   - In the Customers – Windows Internet Explorer window, click the **Close** button.

▶ Task 6: Enable the View state

1. Enable the **View** state for the **List** Web Form by removing the **EnableViewState** page attribute.

   - In the DynamicData/PageTemplates/List.aspx window, remove the **EnableViewState** attribute from the **Page** directive.

     ```
     <%@ Page Language="C#" MasterPageFile="~/Site.master"
     CodeFile="List.aspx.cs" Inherits="List" %>
     ```

2. Save and close the **List** Web Form.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save DynamicData/PageTemplates/List.aspx**.

   b. In the DynamicData/PageTemplates/List.aspx window, click the **Close** button.

Exercise 2: Caching the Countries

▶ Task 1: Cache a DataTable

1. In the **Services_Customers** class, create a private member variable named **countryDataTable**, of type **System.Data.DataTable**, to hold the countries retrieved from the database and initialize the variable to **null**.

   a. In Solution Explorer, expand **Services**, right-click **Customers.aspx**, and then click **View Code**.

   b. In the Services/Customers.aspx.cs window, add the following code to the **Services_Customers** class, after the class signature.

   ```
   private System.Data.DataTable countryDataTable = null;
   ```

2. Assign the **Countries** cache item value to the **countryDataTable** private variable, casting it to the data type **System.Data.DataTable**, upon page load.

   - Type the following code in to the **Page_Load** event handler.

   ```
   // Retrieve DataTable from cache
   countryDataTable =
       (System.Data.DataTable) Cache["Countries"];
   ```

3. Check whether an item has been retrieved from the cache by comparing the **countryDataTable** variable with a **null** value, in an **if** construct, when the user clicks the **Get Countries** button.

   - Add the following code at the top of the **GetCountriesButton_Click** event handler.

   ```
   // Does cached item exist?
   if (countryDataTable == null)
   {

   }
   ```

4. Assign the value of the **countryDataTable** local variable to the **DataSource** property of the **CustomersGridView** control.

   - After the **if** construct, type the following code.

   ```
   // Set GridView DataSource
   CustomersGridView.DataSource = countryDataTable;
   ```

5. If an item is not retrieved from the cache, move the existing line of code from after the **if** construct that declares and instantiates the local **customersService** variable.

   a. Locate and select the following code,

   ```
   CustomersClient customersService = new CustomersClient();
   ```

   right-click the selection, and then click **Cut**.

   b. Place the cursor within the **if** construct, right-click, and then click **Paste**.

6. If an item is not retrieved from the cache, assign the result of the call to the **GetCountries** Windows Communication Foundation (WCF) service method, to the local **countryDataTable** variable.

   - In the **if** construct, append the following code after the line that declares and instantiates the local **customersService** variable.

   ```
   // Retrieve DataTable from WCF Service
   countryDataTable =
   customersService.GetCountries(StartingLettersTextBox.Text);
   ```

7. Add a new variable named **Countries** to the cache object, and assign the value of the local **countryDataTable** variable.

   - In the **if** construct, append the following code after the line that assigns the result of the call to the **GetCountries** WCF service method.

   ```
   // Save DataTable to cache
   Cache["Countries"] = countryDataTable;
   ```

8. Delete the existing line of code that assigns the result of the call to the **GetCountries** WCF service method to the **DataSource** property of the **CustomersGridView** control.

   - In the Services/Customers.aspx.cs window, delete the following line of code from the **GetCountriesButton_Click** event handler.

   ```
   CustomersGridView.DataSource =
   customersService.GetCountries(StartingLettersTextBox.Text);
   ```

9. Save the changes to the Customers.aspx.cs file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Services/Customers.aspx.cs**.

▶ Task 2: Test the cache storage

1. Set a breakpoint on the line that retrieves the **Countries** item from the **Cache** object.
   - In the **GetCountriesButton_Click** event handler, place the cursor at the beginning of the following code, and then press the F9 key.

```
if (countryDataTable == null)
```

2. Run the application in debug mode.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**, or press F5.
3. View the **Customers** Web Form by typing the URL **http://localhost:1110/CustomerManagement/Services/Customers.aspx** in the Address bar.
   - In the Address bar of the Contoso Customer Management – Windows Internet Explorer window, type **http://localhost:1110/CustomerManagement/Services/Customers.aspx**, and then press ENTER.
4. Retrieve the countries starting with the letter **b**.
   - In the Countries by Contoso – Windows Internet Explorer window, in the text box, type **b**, and then click **Get Countries**.

> **Note:** The Customers Web Form code now opens in the debugger, and the current line of code will now check whether you have retrieved an item from the cache.

5. Press F10 to go to the next statement.

> **Note:** You have not retrieved an item from the cache, so you need to create and store an item in the cache.

6. Press F5 to continue running the application.

> **Note:** Notice that the countries now display on the Web page.

7. Close Internet Explorer.
    - In the Countries by Contoso – Windows Internet Explorer window, click the **Close** button.
8. Run the application in the debug mode.
    - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Debugging**, or press F5.
9. View the **Customers** Web Form in browser by using the URL **http://localhost:1110/CustomerManagement/Services/Customers.aspx**.
    - In the Address bar of the Contoso Customer Management – Windows Internet Explorer window, type **http://localhost:1110/CustomerManagement/Services/Customers.aspx**, and then press ENTER.
10. Retrieve the countries starting with the letter **c**.
    - In the Countries by Contoso – Windows Internet Explorer window, in the text box, type **c**, and then click **Get Countries**.

Note: The Customers Web Form code now opens in the debugger, and the current line of code will now check whether you have retrieved an item from the cache.

11. Press F10 to go to the next statement.

Note: You have retrieved an item from the cache. Now, you need to assign the retrieved item to the **CustomersGridView** data source property.

12. Press F5 to continue running the application.

Note: Notice that the countries display on the Web page.

13. Close Internet Explorer.
    - In the Countries by Contoso – Windows Internet Explorer window, click the **Close** button.
14. Remove the breakpoint from the code that retrieves the **Countries** item from the **Cache** object.
    - In the **GetCountriesButton_Click** event handler, place the cursor at the beginning of the following line of code, and the press F9.

```
if (countryDataTable == null)
```

15. Close the Customers.aspx.cs file.
    - In the Services/Customers.aspx.cs window, click the **Close** button.

Exercise 3: Displaying a Visitors Counter on the Default Page

▶ Task 1: Initialize an application variable

1. Add a variable named **Visitors**, with a value of **0**, to the **Application** state.
   a. In Solution Explorer, right-click **Global.asax**, and then click **Open**.
   b. In the Global.asax window, append the following code to the **Application_Start** method, after the code that calls the **Register** method.

```
// Save application variable
Application["Visitors"] = 0;
```

2. Save the Global.asax file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Global.asax**, or press CTRL+S.

▶ Task 2: Add a visitors counter to the default page

1. Add a **div** element with a **class** attribute set to the value of **footer** to the **MainContent** control of the **Default** Web Form.
   a. In Solution Explorer, right-click **Default.aspx**, and then click **View Markup**.
   b. In the Default.aspx window, type the following markup within the **MainContent** control.

```
<div class="footer">
</div>
```

2. Add a **Literal** control named **VisitorLiteral**, to the **div** element.
   - In the Default.aspx window, type the following markup within the **div** element.

```
<asp:Literal ID="VisitorLiteral" runat="server" />
```

3. Save and close the **Default** Web Form.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Default.aspx**.
   b. In the Default.aspx window, click the **Close** button.

4. Add the following style to the Styles/Site.css stylesheet.

```
div.footer
{
    background: #dbddff;
    text-align: center;
    height: 15px; /* Height of the footer */
    width: 100%;
    position: fixed;
    bottom: 0;
}
```

   a. In Solution Explorer, expand **Styles**, and double-click **Site.css**.

   b. In the Styles/Site.css window, press the CTRL+END keys, press ENTER, and then type the following markup.

```
div.footer
{
    background: #dbddff;
    text-align: center;
    height: 15px; /* Height of the footer */
    width: 100%;
    position: fixed;
    bottom: 0;
}
```

5. Save and close the Site.css file.

   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Styles/Site.css**, or press CTRL +S.

   b. In the Styles/Site.css window, click the **Close** button.

6. In the **Page_Load** event handler of the **Default** Web Form, create a local variable named **numVisitors**, of type **long**, to hold the number of visitors and initialize the variable with a value of **0**.

   a. In Solution Explorer, right-click **Default.aspx**, and then click **View Code**.

   b. In the Default.aspx.cs window, type the following code in the **Page_Load** event handler.

```
long numVisitors = 0;
```

7. In the **Page_Load** event handler, check if the **Visitors** Application variable exists, and if so, assign the value to **numVisitors** local variable, casting it to data type **long**.

   - In the Default.aspx.cs window, append the following code in the **Page_Load** event handler, after the declaration and initialization of the **numVisitors** variable.

```
// Check if Application variable exists
if (Application["Visitors"] != null)
{
    numVisitors = long.Parse(Application["Visitors"].ToString());
}
```

8. In the **Page_Load** event handler, assign the text "**Number of visitors:** ", and the value of the **numVisitors** variable, to the **Text** property of the **VisitorLiteral** control.

> **Note:** The **Visitors** application variable must be converted by using the **Parse** method of the **long** data type.

   - In the Default.aspx.cs window, append the following code in the **Page_Load** event handler, after the code that checks for the application variable existence.

```
VisitorLiteral.Text = "Number of visitors: " +
numVisitors.ToString();
```

9. Save and close the Default.aspx.cs file.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Default.aspx.cs**.

   b. In the Default.aspx.cs window, click the **Close** button.

▶ Task 3: Update the visitors counter on a new session

1. Each time a new user session is started, increment the **Visitors** application variable by 1, by appending the following code to the **Session_Start** method in the Global.asax file.

   - In Global.asax window, add the following code in the **Session_Start** method.

```
// Increment Visitors counter
Application["Visitors"] =
    long.Parse(Application["Visitors"].ToString()) + 1;
```

> **Note:** The **Visitors** application variable must be converted by using the **Parse** method of the data type.

2. Save and close the Global.asax file.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Global.asax**.

   b. In the Global.asax window, click the **Close** button.

▶ Task 4: Test the visitors counter

1. View the **Default** Web Form in a browser.

> **Note:** Notice that the Number of visitors is set to **1**.

   - In Solution Explorer, right-click **Default.aspx**, and then click **View in Browser**.

2. Close Internet Explorer.

   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

3. View the **Default** Web Form in a browser.

> **Note:** Notice that the Number of visitors is set to **2**.

   - In Solution Explorer, right-click **Default.aspx**, and then click **View in Browser**.

4. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
   - In the CustomerManagement Microsoft Visual Studio window, click the **Close** button.

▶ Task 5: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 14

Lab Answer Key: Configuring and Deploying a Microsoft® ASP.NET Web Application (Visual Basic)

Contents:

Lab: Configuring and Deploying a Microsoft® ASP.NET Web Application

(Visual Basic)

Exercise 1: Configuring the List View Page Size and Enabling the Save Countries Button

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010 as an Administrator.
    a. On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, right-click **Microsoft Visual Studio 2010**, and then click **Run as administrator**.
    b. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter\M14\VB folder.
    a. In the Start Page – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Open Project**.
    b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M14\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.
    - In Solution Explorer, right-click **Default.aspx**, and then click **Set As Start Page**.

▶ Task 3: View the default page size in the List view

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View all the countries.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **All**.

> **Note:** Notice that by default, 10 items display in the table. This is because the page size is set to **10**.

3. Close Windows® Internet Explorer®.
   - In the Countries – Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Set the default List view page size

1. Open the **web.config** file, and add the appSettings element after the opening **configuration** tag.
   a. In Solution Explorer, right-click **web.config**, and then click **Open**.
   b. In the web.config window, add the **appSettings** element after the opening **configuration** tag.

```
<appSettings>
</appSettings>
```

2. Add a self-closing **add** element with a **key** attribute value of **ListViewPagerSize**, and a **value** attribute value of **5**, to the **appSettings** element.

```
<add key="ListViewPagerSize" value="5" />
```

- In the web.config file, add a self-closing **add** element within the **appSettings** element, with a **key** attribute value of **ListViewPagerSize**, and a **value** attribute value of **5**, to the **appSettings** element.

```
<add key="ListViewPagerSize" value="5" />
```

3. Save and close the web.config file.

   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.

   b. In the web.config window, click the **Close** button.

4. Navigate to **DynamicData/PageTemplates**, and open the **List** Web Form in the Code view.

   - In Solution Explorer, expand **DynamicData**, expand **PageTemplates**, right-click **List.aspx**, and then click **View Code**.

5. In the **Page_Load** event handler of the partial **List** class, check whether the **ListViewPagerSize** key exists in the **appSettings** element of the web.config file. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **If** construct.

   - In the DynamicData/PageTemplates/List.aspx.vb window, in the **Page_Load** event handler of the partial **List** class, append the following code,

```
' Check if key exists in web.config
If Not
System.Web.Configuration.WebConfigurationManager.AppSettings("
ListViewPagerSize") Is Nothing Then
End If
```

   after the code:

```
' Disable various options if the table is readonly
If table.IsReadOnly Then
    GridView1.Columns(0).Visible = False
    InsertHyperLink.Visible = False
    GridView1.EnablePersistedSelection = False
End If
```

6. Retrieve the default page size value from the web.config file in the **Page_Load** event handler, in the **If** construct, by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in a variable named **pagerSize**, of type **Integer**.

   - In the **Page_Load** event handler of the partial **List** class, in the **If** construct, add the following code,

     ```
     ' Get pager size from configuration file
     Dim pagerSize As Integer = Integer.Parse(

     System.Web.Configuration.WebConfigurationManager.AppSettings("
     ListViewPagerSize"))
     ```

     after the code:

     ```
     ' Check if key exists in web.config
     If Not
     System.Web.Configuration.WebConfigurationManager.AppSettings("
     ListViewPagerSize") Is Nothing Then
     ```

7. Set the page size of the **GridView** control to the value of the **pagerSize** variable.

   - In the **Page_Load** event handler of the partial **List** class, in the **If** construct, append the following code,

     ```
     ' Set page size
     GridView1.PageSize = pagerSize
     ```

     after the code:

     ```
     ' Get pager size from configuration file
     Dim pagerSize As Integer = Integer.Parse(

     System.Web.Configuration.WebConfigurationManager.AppSettings("
     ListViewPagerSize"))
     ```

8. Save the changes, and close the List.aspx.vb file.

   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save DynamicData/PageTemplates /List.aspx.vb**.

   b. In the DynamicData/PageTemplates/List.aspx.vb window, click the **Close** button.

▶ Task 5: View the new page size in List view

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View all countries.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **All**.

> **Note:** Notice that only five items display in the table. This is because the page size is set to **5**.

3. Close Internet Explorer.
   - In the Countries – Windows Internet Explorer window, click the **Close** button.

▶ Task 6: View the Country Import Page

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View the **Import Countries** page.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **Import**.

> **Note:** Notice that it is possible to save the countries by using **the Save Countries** button.

3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 7: Set the default value for enabling saving of the filtered countries

1. Open the **web.config** file.
   - In Solution Explorer, under D:\Labfiles\Starter\M14\VB \CustomerManagement, right-click **web.config**, and then click **Open**.
2. In the web.config file, add a new self-closing **add** element within the **appSettings** element, with a **key** attribute value of **EnableSaveImportedCountries**, and a **value** attribute value of **true**.

   ```
   <add key="EnableSaveImportedCountries" value="true" />
   ```

   - In the web.config window, type the following markup,

     ```
     <add key="EnableSaveImportedCountries" value="true" />
     ```

     after the markup:

     ```
     <add key="ListViewPagerSize" value="5" />
     ```
3. Save and close the web.config file.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.
   b. In the web.config window, click the **Close** button.
4. Open the **ImportCountries** Web Form in Code view.
   - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View Code**.
5. Check whether the **EnableSaveImportedCountries** key exists in the **appSettings** element of the web.config file in the **Page_Load** event handler of the partial **ImportCountries** class. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **If** construct.

- In the ImportCountries.aspx.vb window, in the partial **ImportCountries** class, add the following code.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ' Check if key exists in web.config
    If Not
System.Web.Configuration.WebConfigurationManager.AppSettings("
EnableSaveImportedCountries") Is Nothing Then
    End If
End Sub
```

6. Retrieve the default value for enabling the user to save the imported countries from the **web.config** file in the **Page_Load** event handler, in the **If** construct, by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in a variable named **enableSaveCountries**, of type **Boolean**.

   - In the **Page_Load** event handler of the partial **ImportCountries** class, in the **If** construct, add the following code,

```
Dim enableSaveCountries As Boolean = CType(

System.Web.Configuration.WebConfigurationManager.AppSettings("
EnableSaveImportedCountries").ToLower() = "true", Boolean)
```

   after the code:

```
' Check if key exists in web.config
If Not
System.Web.Configuration.WebConfigurationManager.AppSettings("
EnableSaveImportedCountries") Is Nothing Then
```

7. Enable the **SaveButton** control, depending on the value of the **enableSaveCountries** local variable.

- In the **Page_Load** event handler of the partial **ImportCountries** class, in the **If** construct, append the following code,

```
' Enable/disable SaveButton
SaveButton.Enabled = enableSaveCountries
```

after the code:

```
Dim enableSaveCountries As Boolean = CType(

System.Web.Configuration.WebConfigurationManager.AppSettings("
EnableSaveImportedCountries").ToLower() = "true", Boolean)
```

8. Save the changes, and close the ImportCountries.aspx.vb file.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save ImportCountries.aspx.vb**.
   b. In the ImportCountries.aspx.vb window, click the **Close** button.

▶ Task 8: View the Country Import Page

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View the Import Countries page.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **Import**.

> **Note:** Notice that the **Save Countries** button is enabled.

3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 9: Set the default value for enabling saving of the filtered countries

1. Open the **web.config** file.
   - In Solution Explorer, under D:\Labfiles\Starter\M14\VB \CustomerManagement, right-click **web.config**, and then click **Open**.
2. In the web.config file, modify the self-closing **add** element in the **appSettings** element, with a **key** attribute value of **EnableSaveImportedCountries**, to a **value** attribute value of **false**.

```
<add key="EnableSaveImportedCountries" value="false" />
```

3. Save and close the web.config file.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.
   b. In the web.config window, click the **Close** button.

▶ Task 10: View the Country Import page

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View all countries.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **Import**.

> **Note:** Notice that the **Save Countries** button is disabled.

3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Configuring the Visitor Counter

▶ Task 1: Set the default visitor counter value

1. Open the **web.config** file.
   - In Solution Explorer, under D:\Labfiles\Starter\M14\VB \CustomerManagement, right-click **web.config**, and then click **Open**.
2. In the web.config file, append a self-closing **add** element to the **appSettings** element, with a **key** attribute value of **VisitorCounter**, and a **value** attribute value of **0**.

```
<add key="VisitorCounter" value="0" />
```

3. Save and close the web.config file.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.
   b. In the web.config window, click the **Close** button.

▶ Task 2: Set the visitor counter value in the Application_Start method

1. Open the **Global.asax** file
   - In Solution Explorer, right-click **Global.asax**, and then click **Open**.
2. In the **Application_Start** method, declare a local variable named **numVisitors**, of type **long**, and initialize the variable with a value of **0**.
   - In the Global.asax window, in the **Application_Start** method, type the following code,

```
Dim numVisitors As Long = 0
```

   after the code:

```
Register(RouteTable.Routes)
```

3. Check whether the **VisitorCounter** key exists in the **appSettings** element of the web.config file in the **Application_Start** method. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **If** construct.

```
' Check if key exists in web.config
If Not
System.Web.Configuration.WebConfigurationManager.AppSettings("Visi
torCounter") Is Nothing Then
End If
```

   - In the Global.asax window, in the **Application_Start** method, append the following code,

```
' Check if key exists in web.config
If Not
System.Web.Configuration.WebConfigurationManager.AppSettings("
VisitorCounter") Is Nothing Then
End If
```

     after the code:

```
Dim numVisitors As Long = 0
```

4. Retrieve the visitor counter value from the web.config file in the **Application_Start** method by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in the **numVisitors** variable.

```
' Get visitor counter from configuration file
numVisitors = Long.Parse(

System.Web.Configuration.WebConfigurationManager.AppSettings("Visi
torCounter"))
```

- In the Global.asax window, in the **If** construct of the **Application_Start** method, type the following code,

```
' Get visitor counter from configuration file
numVisitors = Long.Parse(

System.Web.Configuration.WebConfigurationManager.AppSettings("
VisitorCounter"))
```

after the code:

```
If Not
System.Web.Configuration.WebConfigurationManager.AppSettings("
VisitorCounter") Is Nothing Then
```

5. Save the visitor counter value to the **Application** state in the **Visitors** application variable, by modifying the existing assignment of the value to **numVisitors**.

```
' Save application variable
Application("Visitors") = numVisitors
```

- In the Global.asax window, in the **Application_Start** method, modify the following code,

```
' Save application variable
Application("Visitors") = 0
```

as:

```
' Save application variable
Application("Visitors") = numVisitors
```

6. Save the Global.asax file.

- In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Global.asax**.

▶ Task 3: Save the visitor counter value in the Application_End method

1. In the **Application_End** method, declare a local variable named **mainConfiguration**, of type **System.Configuration.Configuration**, and initialize the variable with a value returned by the **System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~/")** method.

```
' Get configuration instance
Dim mainConfiguration As System.Configuration.Configuration =

System.Web.Configuration.WebConfigurationManager.OpenWebConfigurat
ion("~/")
```

   - In the Global.asax window, in the **Application_End** method, type the following code.

```
' Get configuration instance
Dim mainConfiguration As System.Configuration.Configuration =

System.Web.Configuration.WebConfigurationManager.OpenWebConfig
uration("~/")
```

2. Check whether the **VisitorCounter** key exists in the **appSettings** element of the web.config file in the **Application_End** method by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **If Then**...**Else** construct.

```
' Check if key exists in web.config
If
System.Web.Configuration.WebConfigurationManager.AppSettings("Visi
torCounter") Is Nothing Then
Else
End If
```

- In the **Application_End** method in the Global.asax file, append the following code,

```
' Check if key exists in web.config
If
System.Web.Configuration.WebConfigurationManager.AppSettings("
VisitorCounter") Is Nothing Then
Else
End If
```

after the code:

```
' Get configuration instance
Dim mainConfiguration As System.Configuration.Configuration =

System.Web.Configuration.WebConfigurationManager.OpenWebConfig
uration("~/")
```

3. In the **Application_End** method, in the **If** construct, save a new **VisitorCounter** key and value in the web.config file. To do this, use the **Add** method of the **System.Web.Configuration.WebConfigurationManager.AppSettings** property. Set the value to the value saved in the **Visitors** application variable, if it exists. If the application variable does not exist, save the value of **0**.

```
' Save new key and value
mainConfiguration.AppSettings.Settings.Add(New
KeyValueConfigurationElement("VisitorCounter", _
    IIf(Application("Visitors") Is Nothing, "0",
Application("Visitors").ToString())))
```

- In the Global.asax window, in the **If** construct of the **Application_End** method, type the following code,

```
' Save new key and value
mainConfiguration.AppSettings.Settings.Add(New
KeyValueConfigurationElement("VisitorCounter", _
    IIf(Application("Visitors") Is Nothing, "0",
Application("Visitors").ToString())))
```

after the code:

```
' Check if key exists in web.config
If
System.Web.Configuration.WebConfigurationManager.AppSettings("
VisitorCounter") Is Nothing Then
```

4. In the **Application_End** method, in the **Else** construct, save a new **VisitorCounter** value in the web.config file. To do this, use the **System.Web.Configuration.WebConfigurationManager.AppSettings** property. If the **Visitors** application variable exists, set the **VisitorCounter** value to the value saved in the **Visitors** application variable. If the **Visitors** application variable does not exist, assign the value **0**, to the **VisitorCounter** value.

```
' Save new value
mainConfiguration.AppSettings.Settings("VisitorCounter").Value = _
    IIf(Application("Visitors") Is Nothing, "0",
Application("Visitors").ToString())
```

- In the Global.asax window, in the **else** construct of the **Application_End** method, type the following code,

```
' Save new value
mainConfiguration.AppSettings.Settings("VisitorCounter").Value
= _
    IIf(Application("Visitors") Is Nothing, "0",
Application("Visitors").ToString())
```

after the code:

```
Else
```

5. In the **Application_End** method, save the changes to the web.config file by using the **System.Configuration.Configuration.Save** method on the **mainConfiguration** object.

```
' Save to file
mainConfiguration.Save()
```

   - In the Global.asax window, in the **Application_End** method, append the following code,

```
' Save to file
mainConfiguration.Save()
```

     after the code:

```
    ' Save new value

mainConfiguration.AppSettings.Settings("VisitorCounter").Value
= _
        IIf(Application("Visitors") Is Nothing, "0",
Application("Visitors").ToString())
End If
```

6. Save and close the **Global.asax** file.

   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Global.asax**.

   b. In the Global.asax window, click the **Close** button.

Exercise 3: Deploying the Web Application

▶ Task 1: Build the application

- Build the application, and fix any compile-time errors.
  - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Build** menu, click **Build Web Site**, and fix any errors that are displayed.

▶ Task 2: Copy the Web site

1. Open the Copy Web Site tool.
   - In Solution Explorer, right-click the **D:\Labfiles\Starter\M14\VB \CustomerManagement** Web site, and then click **Copy Web Site**.
2. Connect to the **CM** application on the **Default Web Site** on the local Internet Information Services (IIS).
   a. In the Copy Web Site tool window, click **Connect**.
   b. In the **Open Web Site** dialog box, click **Local IIS**.
   c. In the **Select the Web site you want to open** box, expand **Default web Site**, click **CM**, and then click **Open**.
3. Copy source files to the destination site. Overwrite the existing files.
   a. In the Source Web site pane, select all files and folders, and then click the **Copy selected files from source to remote web site** button.
   b. In the **Confirm File Overwrite** dialog box, click **Yes**.

> **Note:** Notice that all files and folders from the Source Web site pane are copied to the Remote Web sites pane.

4. Close the Copy Web Site tool.
   - In the Copy Web Site tool window, click the **Close** button.

▶ Task 3: Test the deployed Web site

1. Open Internet Explorer, and enter the URL, **http://localhost:1112/CM**.
   a. On the **Start** menu, click I**nternet Explorer**.
   b. In the Internet Explorer Address bar, type **http://localhost:1112/CM**, and then press ENTER.
2. On the default page of the deployed Contoso Customer Management Web site, notice that the **Number of visitors** is set to **1**.
3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
4. Open Internet Explorer again, and browse to **http://localhost1112/CM**.
   a. On the **Start** menu, click **Internet Explorer**.
   b. In the Internet Explorer Address bar, type **http://localhost1112/CM**, and then press ENTER.
5. On the default page of the deployed Customer Management Web site, notice that the **Number of visitors** is now set to **2**.
6. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
7. Open IIS Manager as an Administrator.
   a. On the **Start** menu, click **Control Panel**.
   b. In the Control Panel window, click the arrow next to **Control Panel**, and then click **All Control Panel Items**.
   c. In the Control Panel window, double-click **Administrative Tools**, right-click **Internet Information Services (IIS) Manager**, and then click **Run as administrator**.
   d. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.
8. Stop IIS.
   - In the Internet Information Services (IIS) Manager window, right-click **10267A-GEN-DEV (10267A-GEN-DEV\student)**, and then click **Stop**.

9. Open the **web.config** file from the folder **C:\inetpub\wwwroot\CM**, by using Windows Explorer.
    a. On the **Start** menu, point to **All Programs**, click **Accessories**, and then click **Windows Explorer**.
    b. In Windows Explorer, navigate to **Computer\Local Disk (C:) \inetpub\wwwroot\CM**, and then double-click **web.config**.
10. In Visual Studio 2010, notice the **add** element in the **appSettings** element, with a **key** attribute value of **VisitorCounter** and a **value** attribute value of **2**.
11. Close the deployed **web.config** file.
    - In the web.config window, click the **Close** button.
12. Open the **web.config** file from the project Web site that is currently open in Visual Studio 2010.
    - In Solution Explorer of Visual Studio 2010, under D:\Labfiles\Starter \M14\VB\CustomerManagement, right-click **web.config**, and then click **Open**.
13. In Visual Studio 2010, modify the **value** attribute value to **100** for the **add** element in the **appSettings** element, with a **key** attribute value of **VisitorCounter**.
14. Save and close the **web.config** file.
    a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.
    b. In the web.config window, click the **Close** button.
15. Open the Copy Web Site tool.
    - In Solution Explorer, right-click the **D:\Labfiles\Starter\M14\VB \CustomerManagement** Web site, and then click **Copy Web Site**.
16. Connect to the local IIS, and copy the web.config source file to the destination site.
    a. In the Copy Web Site tool window, click **Connect**.
    b. In the **Open Web Site** dialog box, ensure that **CM** is selected, and then click **Open**.
    c. In the Source Web site pane, click **web.config**, and then click the **Copy selected files from source to remote web site** button.
    d. In the **Confirm File Overwrite** dialog box, click **Yes**.

17. Close the Copy Web Site tool.
    - In the Copy Web Site tool window, click the **Close** button.
18. Close Visual Studio 2010.
    - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, click the **Close** button.
19. Close Windows Explorer.
    a. In Windows Explorer, click the **Close** button.
    b. In the Control Panel window, click the **Close** button.
20. Start IIS.
    - In the Internet Information Services (IIS) Manager window, right-click **10267A-GEN-DEV (10267A-GEN-DEV\student)**, and then click **Start**.
21. Close IIS Manager.
    - In the Internet Information Services (IIS) Manager window, click the **Close** button.
22. Open Internet Explorer, and browse to **http://localhost:1112/CM**.
    a. On the **Start** menu, click **Internet Explorer**.
    b. In the **Internet Explorer Address** bar, type **http://localhost:1112/CM**, and then press ENTER.

    On the default page of the deployed Contoso Customer Management Web site, notice that the **Number of visitors** is set to **101**.
23. Close Internet Explorer.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 14

Lab Answer Key: Configuring and Deploying a Microsoft® ASP.NET Web Application (Visual C#)

Contents:

Lab: Configuring and Deploying a Microsoft® ASP.NET Web Application

(C#)

Exercise 1: Configuring the List View Page Size and Enabling the Save Countries Button

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010 as an Administrator.
   a. On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, right-click **Microsoft Visual Studio 2010**, and then click **Run as administrator**.
   b. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M14\CS folder.
   a. In the Start Page – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M14\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.
  - In Solution Explorer, right-click **Default.aspx**, and then click **Set As Start Page**.

▶ Task 3: View the default page size in the List view

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View all the countries.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **All**.

> **Note:** Notice that by default, 10 items display in the table. This is because the page size is set to **10**.

3. Close Windows® Internet Explorer®.
   - In the Countries – Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Set the default List view page size

1. Open the **web.config** file, and add the **appSettings** element after the opening **configuration** tag.
   a. In Solution Explorer, right-click **web.config**, and then click **Open**.
   b. In the web.config window, add the **appSettings** element after the opening **configuration** tag.

```
<appSettings>
</appSettings>
```

2. Add a self-closing **add** element with a **key** attribute value of **ListViewPagerSize**, and a **value** attribute value of **5**, to the **appSettings** element.

```
<add key="ListViewPagerSize" value="5" />
```

   - In the web.config file, add a self-closing **add** element within the **appSettings** element, with a **key** attribute value of **ListViewPagerSize**, and a **value** attribute value of **5**, to the **appSettings** element.

```
<add key="ListViewPagerSize" value="5" />
```

3. Save and close the web.config file.

   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.

   b. In the web.config window, click the **Close** button.

4. Navigate to **DynamicData/PageTemplates**, and open the **List** Web Form in the Code view.

   - In Solution Explorer, expand **DynamicData**, expand **PageTemplates**, right-click **List.aspx**, and then click **View Code**.

5. In the **Page_Load** event handler of the partial **List** class, check whether the **ListViewPagerSize** key exists in the **appSettings** element of the web.config file. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **if** construct.

   - In the DynamicData/PageTemplates/List.aspx.cs window, in the **Page_Load** event handler of the partial **List** class, append the following code,

     ```
     // Check if key exists in web.config
     if
     (System.Web.Configuration.WebConfigurationManager.AppSettings[
     "ListViewPagerSize"] != null)
     {
     }
     ```

     after the code:

     ```
     // Disable various options if the table is readonly
     if (table.IsReadOnly) {
         GridView1.Columns[0].Visible = false;
         InsertHyperLink.Visible = false;
         GridView1.EnablePersistedSelection = false;
     }
     ```

6. Retrieve the default page size value from the web.config file in the **Page_Load** event handler, in the **if** construct, by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in a variable named, **pagerSize**, of type, **int**.

   - In the **Page_Load** event handler of the partial **List** class, in the **if** construct, add the following code,

     ```
     // Get pager size from configuration file
     int pagerSize = int.Parse(

     System.Web.Configuration.WebConfigurationManager.AppSettings["
     ListViewPagerSize"]);
     ```

     after the code:

     ```
     // Check if key exists in web.config
     if
     (System.Web.Configuration.WebConfigurationManager.AppSettings[
     "ListViewPagerSize"] != null)
     {
     ```

7. Set the page size of the **GridView** control to the value of the **pagerSize** variable.

   - In the **Page_Load** event handler of the partial **List** class, in the **if** construct, append the following code,

     ```
     // Set page size
     GridView1.PageSize = pagerSize;
     ```

     after the code:

     ```
     // Get pager size from configuration file
     int pagerSize = int.Parse(

     System.Web.Configuration.WebConfigurationManager.AppSettings["
     ListViewPagerSize"]);
     ```

8. Save the changes, and close the List.aspx.cs file.

   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save DynamicData/PageTemplates /List.aspx.cs**.

   b. In the DynamicData/PageTemplates/List.aspx.cs window, click the **Close** button.

▶ Task 5: View the new page size in List view

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View all countries.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **All**.

> **Note:** Notice that only five items display in the table. This is because the page size is set to **5**.

3. Close Internet Explorer.
   - In the Countries – Windows Internet Explorer window, click the **Close** button.

▶ Task 6: View the Country Import Page

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View the **Import Countries** page.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **Import**.

> **Note:** Notice that it is possible to save the countries by using the **Save Countries** button.

3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 7: Set the default value for enabling saving of the filtered countries

1. Open the **web.config** file.
   - In Solution Explorer, under D:\Labfiles\Starter\M14\CS \CustomerManagement, right-click **web.config**, and then click **Open**.
2. In the web.config file, add a new self-closing **add** element within the **appSettings** element, with a **key** attribute value of **EnableSaveImportedCountries**, and a **value** attribute value of **true**.

   ```
   <add key="EnableSaveImportedCountries" value="true" />
   ```

   - In the web.config window, type the following markup,

     ```
     <add key="EnableSaveImportedCountries" value="true" />
     ```

     after the markup:

     ```
     <add key="ListViewPagerSize" value="5" />
     ```
3. Save and close the web.config file.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.
   b. In the web.config window, click the **Close** button.
4. Open the **ImportCountries** Web Form in Code view.
   - In Solution Explorer, right-click **ImportCountries.aspx**, and then click **View Code**.
5. Check whether the **EnableSaveImportedCountries** key exists in the **appSettings** element of the web.config file in the **Page_Load** event handler of the partial **ImportCountries** class. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **if** construct.

- In the ImportCountries.aspx.cs window, in the **Page_Load** event handler of the partial **ImportCountries** class, add the following code.

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings[
"EnableSaveImportedCountries"] != null)
{
}
```

6. Retrieve the default value for enabling the user to save the imported countries from the **web.config** file in the **Page_Load** event handler, in the **if** construct, by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in a variable named **enableSaveCountries**, of type **bool**.

   - In the **Page_Load** event handler of the partial **ImportCountries** class, in the **if** construct, add the following code,

```
// Get enable save countries value from configuration file
bool enableSaveCountries = (bool)

(System.Web.Configuration.WebConfigurationManager.AppSettings[
"EnableSaveImportedCountries"].ToLower() == "true");
```

   after the code:

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings[
"EnableSaveImportedCountries"] != null)
{
```

7. Enable the **SaveButton** control, depending on the value of the **enableSaveCountries** local variable.

- In the **Page_Load** event handler of the partial **ImportCountries** class, in the **if** construct, append the following code,

```
// Enable/disable SaveButton
SaveButton.Enabled = enableSaveCountries;
```

after the code:

```
// Get enable save countries value from configuration file
bool enableSaveCountries = (bool)

(System.Web.Configuration.WebConfigurationManager.AppSettings[
"EnableSaveImportedCountries"].ToLower() == "true");
```

8. Save the changes, and close the ImportCountries.aspx.cs file.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save ImportCountries.aspx.cs**.
   b. In the ImportCountries.aspx.cs window, click the **Close** button.

▶ Task 8: View the Country Import Page

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View the Import Countries page.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **Import**.

> **Note:** Notice that the **Save Countries** button is enabled.

3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 9: Set the default value for enabling saving of the filtered countries

1. Open the **web.config** file.
   - In Solution Explorer, under D:\Labfiles\Starter\M14\CS \CustomerManagement, right-click **web.config**, and then click **Open**.
2. In the web.config file, modify the self-closing **add** element in the **appSettings** element, with a **key** attribute value of **EnableSaveImportedCountries**, to a **value** attribute value of **false**.

```
<add key="EnableSaveImportedCountries" value="false" />
```

3. Save and close the web.config file.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.
   b. In the web.config window, click the **Close** button.

▶ Task 10: View the Country Import page

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Debug** menu, click **Start Without Debugging**.
2. View all countries.
   - In the Contoso Customer Management – Windows Internet Explorer window, on the **Countries** menu, click **Import**.

> **Note:** Notice that the **Save Countries** button is disabled.

3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Configuring the Visitor Counter

▶ Task 1: Set the default visitor counter value

1. Open the **web.config** file.
   - In Solution Explorer, under D:\Labfiles\Starter\M14\CS \CustomerManagement, right-click **web.config**, and then click **Open**.
2. In the web.config file, append a self-closing **add** element to the **appSettings** element, with a **key** attribute value of **VisitorCounter**, and a **value** attribute value of **0**.

```
<add key="VisitorCounter" value="0" />
```

3. Save and close the web.config file.
   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.
   b. In the web.config window, click the **Close** button.

▶ Task 2: Set the visitor counter value in the Application_Start method

1. Open the **Global.asax** file.
   - In Solution Explorer, right-click **Global.asax**, and then click **Open**.
2. In the **Application_Start** method, declare a local variable named **numVisitors**, of type **long**, and initialize the variable with a value of **0**.
   - In the Global.asax window, in the **Application_Start** method, type the following code,

```
long numVisitors = 0;
```

     after the code:

```
Register(RouteTable.Routes);
```

3. Check whether the **VisitorCounter** key exists in the **appSettings** element of the web.config file in the **Application_Start** method. To check this, use the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **if** construct.

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings["Vis
itorCounter"] != null)
{
}
```

- In the Global.asax window, in the **Application_Start** method, append the following code,

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings[
"VisitorCounter"] != null)
{
}
```

  after the code:

```
long numVisitors = 0;
```

4. Retrieve the visitor counter value from the web.config file in the **Application_Start** method by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class, and save the retrieved value in the **numVisitors** variable.

```
// Get visitor counter from configuration file
numVisitors = long.Parse(

System.Web.Configuration.WebConfigurationManager.AppSettings["Visi
torCounter"]);
```

- In the Global.asax window, in the **if** construct of the **Application_Start** method, type the following code,

```
// Get visitor counter from configuration file
numVisitors = long.Parse(

System.Web.Configuration.WebConfigurationManager.AppSettings["
VisitorCounter"]);
```

after the code:

```
if
(System.Web.Configuration.WebConfigurationManager.AppSettings[
"VisitorCounter"] != null)
{
```

5. Save the visitor counter value to the **Application** state in the **Visitors** application variable, by modifying the existing assignment of the value to **numVisitors**.

```
// Save application variable
Application["Visitors"] = numVisitors;
```

- In the Global.asax window, in the **Application_Start** method, modify the following code,

```
// Save application variable
Application["Visitors"] = 0;
```

as:

```
// Save application variable
Application["Visitors"] = numVisitors;
```

6. Save the Global.asax file.

- In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Global.asax**.

▶ Task 3: Save the visitor counter value in the Application_End method

1. In the **Application_End** method, declare a local variable named **mainConfiguration**, of type **System.Configuration.Configuration**, and initialize the variable with a value returned by the **System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration("~/")** method.

```
// Get configuration instance
System.Configuration.Configuration mainConfiguration =

System.Web.Configuration.WebConfigurationManager.OpenWebConfigurat
ion("~/");
```

   - In the Global.asax window, in the **Application_End** method, type the following code.

```
// Get configuration instance
System.Configuration.Configuration mainConfiguration =

System.Web.Configuration.WebConfigurationManager.OpenWebConfig
uration("~/");
```

2. Check whether the **VisitorCounter** key exists in the **appSettings** element of the web.config file in the **Application_End** method by using the **AppSettings** property of the **System.Web.Configuration.WebConfigurationManager** class in an **if**...**else** construct.

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings["Vis
itorCounter"] == null)
{
}
else
{
}
```

- In the **Application_End** method in the Global.asax file, append the following code,

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings[
"VisitorCounter"] == null)
{
}
else
{
}
```

after the code:

```
// Get configuration instance
System.Configuration.Configuration mainConfiguration =

System.Web.Configuration.WebConfigurationManager.OpenWebConfig
uration("~/");
```

3. In the **Application_End** method, in the **If** construct, save a new **VisitorCounter** key and value in the web.config file. To do this, use the **Add** method of the **System.Web.Configuration.WebConfigurationManager.AppSettings** property. Set the value to the value saved in the **Visitors** application variable, if it exists. If the application variable does not exist, save the value of **0**.

```
// Save new key and value
mainConfiguration.AppSettings.Settings.Add(new
KeyValueConfigurationElement("VisitorCounter",
    Application["Visitors"] == null ? "0" :
Application["Visitors"].ToString()));
```

- In the Global.asax window, in the **if** construct of the **Application_End** method, type the following code,

```
// Save new key and value
mainConfiguration.AppSettings.Settings.Add(new
KeyValueConfigurationElement("VisitorCounter",
    Application["Visitors"] == null ? "0" :
Application["Visitors"].ToString()));
```

after the code:

```
// Check if key exists in web.config
if
(System.Web.Configuration.WebConfigurationManager.AppSettings[
"VisitorCounter"] == null)
{
```

4. In the **Application_End** method, in the **else** construct, save a new **VisitorCounter** value in the web.config file. To do this, use the **System.Web.Configuration.WebConfigurationManager.AppSettings** property. If the **Visitors** application variable exists, set the **VisitorCounter** value to the value saved in the **Visitors** application variable. If the **Visitors** application variable does not exist, assign the value, **0**, to the **VisitorCounter** value.

```
// Save new value
mainConfiguration.AppSettings.Settings["VisitorCounter"].Value =
    Application["Visitors"] == null ? "0" :
Application["Visitors"].ToString();
```

- In the Global.asax window, in the **else** construct of the **Application_End** method, type the following code,

```
// Save new value
mainConfiguration.AppSettings.Settings["VisitorCounter"].Value
=
    Application["Visitors"] == null ? "0" :
Application["Visitors"].ToString();
```

after the code:

```
else
{
```

5. In the **Application_End** method, save the changes to the web.config file by using the **System.Configuration.Configuration.Save** method on the **mainConfiguration** object.

```
// Save to file
mainConfiguration.Save();
```

   - In the Global.asax window, in the **Application_End** method, append the following code,

```
// Save to file
mainConfiguration.Save();
```

     after the code:

```
    // Save new value

mainConfiguration.AppSettings.Settings["VisitorCounter"].Value
=
        Application["Visitors"] == null ? "0" :
Application["Visitors"].ToString();
}
```

6. Save and close the **Global.asax** file.

   a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save Global.asax**.

   b. In the Global.asax window, click the **Close** button.

Exercise 3: Deploying the Web Application

▶ Task 1: Build the application

- Build the application, and fix any compile-time errors.
  - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **Build** menu, click **Build Web Site**, and fix any errors that are displayed.

▶ Task 2: Copy the Web site

1. Open the Copy Web Site tool.
   - In Solution Explorer, right-click the **D:\Labfiles\Starter\M14\CS\CustomerManagement** Web site, and then click **Copy Web Site**.
2. Connect to the **CM** application on the **Default Web Site** on the local Internet Information Services (IIS).
   a. In the Copy Web Site tool window, click **Connect**.
   b. In the **Open Web Site** dialog box, click **Local IIS**.
   c. In the **Select the Web site you want to open** box, expand **Default web Site**, click **CM**, and then click **Open**.
3. Copy source files to the destination site. Overwrite the existing files.
   a. In the Source Web site pane, select all files and folders, and then click the **Copy selected files from source to remote web site** button.
   b. In the **Confirm File Overwrite** dialog box, click **Yes**.

> **Note:** Notice that all files and folders from the Source Web site pane are copied to the Remote Web sites pane.

4. Close the Copy Web Site tool.
   - In the Copy Web Site tool window, click the **Close** button.

▶ Task 3: Test the deployed Web site

1. Open Internet Explorer, and enter the URL, **http://localhost:1112/CM**.
   a. On the **Start** menu, click **Internet Explorer**.
   b. In the Internet Explorer Address, type **http://localhost:1112/CM**, and then press ENTER.
2. On the default page of the deployed Contoso Customer Management Web site, notice that the **Number of visitors** is set to **1**.
3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
4. Open Internet Explorer again, and browse to **http://localhost:1112/CM**.
   a. On the Start menu, click **Internet Explorer**.
   b. In the Internet Explorer Address bar, type **http://localhost:1112/CM**, and then press ENTER.
5. On the default page of the deployed Customer Management Web site, notice that the **Number of visitors** is now set to **2**.
6. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
7. Open IIS Manager as an Administrator.
   a. On the **Start** menu, click **Control Panel**.
   b. In the Control Panel window, click the arrow next to **Control Panel**, and then click **All Control Panel Items**.
   c. In the Control Panel window, double-click **Administrative Tools**, right-click **Internet Information Services (IIS) Manager**, and then click **Run as administrator**.
   d. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.
8. Stop IIS.
   - In the Internet Information Services (IIS) Manager window, right-click **10267A-GEN-DEV (10267A-GEN-DEV\Admin)**, and click **Stop**.

9. Open the **web.config** file from the folder **C:\inetpub\wwwroot\CM**, by using Windows Explorer.
    a. On the **Start** menu, point to **All Programs**, click **Accessories**, and then click **Windows Explorer**.
    b. In Windows Explorer, navigate to **Computer\Local Disk (C:) \inetpub\wwwroot\CM**, and then double-click **web.config**.
10. In Visual Studio 2010, notice the **add** element in the **appSettings** element, with a **key** attribute value of **VisitorCounter** and a **value** attribute value of **2**.
11. Close the deployed **web.config** file.
    - In the web.config window, click the **Close** button.
12. Open the **web.config** file from the project Web site that is currently open in Visual Studio 2010.
    - In Solution Explorer of Visual Studio 2010, under D:\Labfiles\Starter \M14\CS\CustomerManagement, right-click **web.config**, and then click **Open**.
13. In Visual Studio 2010, modify the **value** attribute value to **100** for the **add** element in the **appSettings** element, with a **key** attribute value of **VisitorCounter**.
14. Save and close the **web.config** file.
    a. In the CustomerManagement – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Save web.config**.
    b. In the web.config window, click the **Close** button.
15. Open the Copy Web Site tool.
    - In Solution Explorer, right-click the **D:\Labfiles\Starter\M14 \CS\CustomerManagement** Web site, and then click **Copy Web Site**.
16. Connect to the local IIS, and copy the web.config source file to the destination site.
    a. In the Copy Web Site tool window, click **Connect**.
    b. In the **Open Web Site** dialog box, ensure that **CM** is selected, and then click **Open**.
    c. In the Source Web site pane, click **web.config**, and then click the **Copy selected files from source to remote web site** button.
    d. In the **Confirm File Overwrite** dialog box, click **Yes**.

17. Close the Copy Web Site tool.
    - In the Copy Web Site tool window, click the **Close** button.
18. Close Visual Studio 2010.
    - In the CustomerManagement – Microsoft Visual Studio (Administrator) window, click the **Close** button.
19. Close Windows Explorer.
    a. In Windows Explorer, click the **Close** button.
    b. In the Control Panel window, click the **Close** button.
20. Start IIS.
    - In the Internet Information Services (IIS) Manager window, right-click **10267A-GEN-DEV (10267A-GEN-DEV\student)**, and then click **Start**.
21. Close IIS Manager.
    - In the Internet Information Services (IIS) Manager window, click the **Close** button.
22. Open Internet Explorer, and browse to **http://localhost:1112/CM**.
    a. On the **Start** menu, click **Internet Explorer**.
    b. In the Internet Explorer Address bar, type **http://localhost:1112/CM**, and then press ENTER.

    On the default page of the deployed Contoso Customer Management Web site, notice that the **Number of visitors** is set to **101**.
23. Close Internet Explorer.
    - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 15

Lab Answer Key: Securing a Microsoft® ASP.NET Web Application (Visual Basic)

Contents:

Lab: Securing a Microsoft® ASP.NET Web Application

(Visual Basic)

Exercise 1: Enabling Forms Authentication

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M15\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M15\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.
  - In Solution Explorer, right-click **Default.aspx**, and then click **Set As Start Page**.

▶ Task 3: Add the Login controls

1. Open the master page, **Site.master**, in Source view.
   - In Solution Explorer, right-click **Site.master**, and then click **View Markup**.
2. Add a **div** element with a **class** attribute with the value of **login**, to the **body** element of the master page **Site.master**, after the **div** element with a **class** attribute set to the value of **appTitle**.

```
<div class="login">
</div>
```

   - In the Site.master window, locate the **div** element with a **class** attribute set to the value of **appTitle**, place the cursor after the closing **div** tag, press ENTER, and then type the following markup.

```
<div class="login">
</div>
```

3. Add a **LoginStatus** control named **MainLoginStatus**, to the **div** element with a **class** attribute value of **login**.

```
<asp:LoginStatus ID="MainLoginStatus" runat="server"
LogoutAction="RedirectToLoginPage"
    LogoutPageUrl="~/Account/Login.aspx" />
```

Note: The Login.aspx Web Form does not yet exist. You will add one later in this exercise.

Note: The **LoginStatus** control must have the **LogoutAction** attribute set to the value of **RedirectToLoginPage** to send the user to the login page. Optionally, the attribute **LogoutPageUrl**, can be set to the value of **~/Account/Login.aspx**, to redirect to this specific page, instead of the one specified in the web.config file.

   a. In the Site.master window, place the cursor at the end of the opening **div** tag that you have added, and then press ENTER.
   b. In the Site.master window, point to **Toolbox**.
   c. In Toolbox, expand **Login**, and then double-click **LoginStatus**.

d. Change the **ID** attribute of the **LoginStatus** control from **LoginStatus1** to **MainLoginStatus**.

e. Add a **LogoutAction** attribute, and set the value to **RedirectToLoginPage**.

f. Add a **LogoutPageUrl** attribute, and set the value to **~/Account/Login.aspx**.

```
<asp:LoginStatus ID="MainLoginStatus" runat="server"
LogoutAction="RedirectToLoginPage"
    LogoutPageUrl="~/Account/Login.aspx" />
```

4. Add a **LoginName** control named **MainLoginName**, to the **div** element, after the **MainLoginStatus** control.

```
<asp:LoginName ID="MainLoginName" runat="server" />
```

a. In the Site.master window, place the cursor at the end of the self-closing **MainLoginStatus** element, and then press ENTER.

b. In the Site.master window, point to **Toolbox**.

c. In Toolbox, expand **Login**, and then double-click **LoginName**.

d. Change the **ID** attribute of the **LoginName** control from **LoginName1** to **MainLoginName**.

```
<asp:LoginName ID="MainLoginName" runat="server" />
```

5. Save and close the master page.

a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**.

b. In the Site.master window, click the **Close** button.

6. Add the following style to the Styles/Site.css stylesheet.

```
div.login
{
    position: fixed;
    top: 49px;
    right: 10px;
    padding-left: 10px;
    padding-bottom: 10px;
    background-color: #ffffff;
}
```

a. In Solution Explorer, expand **Styles**, and double-click **Site.css**.

b. In the Styles/Site.css window, press CTRL+END, press ENTER, and then type the following markup.

```
div.login
{
    position: fixed;
    top: 49px;
    right: 10px;
    padding-left: 10px;
    padding-bottom: 10px;
    background-color: #ffffff;
}
```

7. Save and close the Site.css file.

- In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Styles/Site.css**, or press CTRL +S.
- In the Styles/Site.css window, click the **Close** button.

▶ Task 4: Test the Login controls

1. Run the application.

- In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

2. View the Login controls.

Note: Notice the Login controls in the upper-right corner of the window, a **Logout** link, and the currently signed-in user, 10267A-GEN-DEV\student, because of the default Windows authentication.

3. Close Windows® Internet Explorer®.

- In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 5: Create a sample ASP.NET Web site

1. Open a second instance of Visual Studio 2010.
   - On the **Start menu of 10267A-GEN-DEV**, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. Create a new file system-based ASP.NET Web site in D:\Labfiles\Starter \M15\VB\SampleWebSite, by using the ASP.NET Web Site template.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **New Web Site**.
   b. In the **New Web Site** dialog box, in the left pane, ensure that **Visual Basic** is selected, and in the middle pane, click **ASP.NET Web Site**. In the **Web location** list, click **File System**, in the adjacent text box, type **D:\Labfiles\Starter\M15\VB\SampleWebSite**, and then click **OK**.
3. Close second instance of Visual Studio 2010.
   - In the SampleWebSite – Microsoft Visual Studio window, click the **Close** button, or press the ALT+F4 keys.

▶ Task 6: Add the Account files and folders

1. Add the **Account** folder and default account content from the SampleWebSite Web site to the **CustomerManagement** Web site, by copying the Account folder from the path D:\Labfiles\Starter\M15\VB \SampleWebSite\Account. Use Windows Explorer to copy the Account folder.
   a. On the **Start** menu, click **Run**.
   b. In the **Run** dialog box, in the **Open** box, type **D:\Labfiles\Starter\M15 \VB\SampleWebSite**, and then click **OK**.
   c. In the Windows Explorer window, right-click **Account**, and then click **Copy**.
   d. Right-click **CustomerManagement**, and then click **Paste**.
   e. Close the Windows Explorer window.

2. In Solution Explorer, refresh the Web site content to ensure the addition of the **Account** folder.

   - In Solution Explorer, right-click **D:\Labfiles\Starter\M15\VB \CustomerManagement**, and then click **Refresh Folder**.

▶ Task 7: Update Account Web Forms to use an existing master page

1. Expand the **Account** folder.
2. Open the **ChangePassword.aspx** Web Form in Source view.

   - In Solution Explorer, under **Account**, double-click **ChangePassword.aspx**.

3. Replace the name of the **ContentPlaceHolder** control referenced **MainContent**, with the name of the ContentPlaceHolder control used in the existing master page, named **MainContentPlaceHolder**.

   - In the Account/ChangePassword.aspx window, replace **MainContent** in the **BodyContent** Content control with **MainContentPlaceHolder**.

4. Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

   a. In the Account/ChangePassword.aspx window, locate and select the **Content** control with an **ID** attribute value of **HeaderContent**.

   ```
   <asp:Content ID="HeaderContent" runat="server"
   ContentPlaceHolderID="HeadContent">
   </asp:Content>
   ```

   b. On the **Edit** menu, click **Delete**, or press DELETE, to delete the selected markup.

5. Save and close the **ChangePassword.aspx** Web Form.

   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Account/ChangePassword.aspx**, or press CTRL+SHIFT+S.

   b. In the Account/ChangePassword.aspx window, click the **Close** button.

6. Open the **ChangePasswordSuccess.aspx** Web Form in Source view.

   - In Solution Explorer, under **Account**, double-click **ChangePasswordSuccess.aspx**.

7. Replace the name of the **ContentPlaceHolder** control referenced, **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.

   - In the Account/ChangePasswordSuccess.aspx window, replace **MainContent** in the **BodyContent** Content control with **MainContentPlaceHolder**.

8. Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

   a. In the Account/ChangePasswordSuccess.aspx window, locate and select the **Content** control with an **ID** attribute value of **HeaderContent**.

   ```
   <asp:Content ID="HeaderContent" runat="server"
   ContentPlaceHolderID="HeadContent">
   </asp:Content>
   ```

   b. On the **Edit** menu, click **Delete**, or press DELETE, to delete the selected markup.

9. Save and close **ChangePasswordSuccess.aspx** Web Form.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Account/ChangePasswordSuccess.aspx**, or press CTRL+SHIFT+S.

   b. In the Account/ChangePasswordSuccess.aspx window, click the **Close** button.

10. Open the **Login.aspx** Web Form in Source view.

    - In Solution Explorer, under **Account**, double-click **Login.aspx**.

11. Replace the name of the **ContentPlaceHolder** control referenced **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.

    - In the Account/Login.aspx window, replace **MainContent** in the **BodyContent** Content control with **MainContentPlaceHolder**.

12. Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

    a. In the Account/Login.aspx window, locate and select the **Content** control with an **ID** attribute value of **HeaderContent**.

    ```
    <asp:Content ID="HeaderContent" runat="server"
    ContentPlaceHolderID="HeadContent">
    </asp:Content>
    ```

    b. On the **Edit** menu, click **Delete**, or press DELETE, to delete the selected markup.

13. Save and close **Login.aspx** Web Form.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Account/Login.aspx**, or press CTRL+SHIFT+S.

    b. In the Account/Login.aspx window, click the **Close** button.

14. Open the **Register.aspx** Web Form in Source view.

    - In Solution Explorer, under **Account**, double-click **Register.aspx**.

15. Replace the name of the **ContentPlaceHolder** control referenced **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.

    - In the Account/Register.aspx window, replace **MainContent** in the **BodyContent** Content control with **MainContentPlaceHolder**.

16. Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

    a. In the Account/Register.aspx window, locate and select the **Content** control with an **ID** attribute value of **HeaderContent**.

    ```
    <asp:Content ID="HeaderContent" runat="server"
    ContentPlaceHolderID="HeadContent">
    </asp:Content>
    ```

    b. On the **Edit** menu, click **Delete**, or press DELETE, to delete the selected markup.

17. Save and close **Register.aspx** Web Form.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Account/Register.aspx**, or press CTRL+SHIFT+S.

    b. In the Account/Register.aspx window, click the **Close** button.

▶ Task 8: Modify the Login Web Form

1. Open the **Login** Web Form.
   - In Solution Explorer, under Account, double-click **Login.aspx**.
2. Set the page title as **Contoso Customer Management − User Login**.
   - In the Login.aspx window, locate the **Title** attribute of the **Page** directive, and replace **Log In** with **Contoso Customer Management - User Login**.
3. Save and close the **Login** Web Form.
   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Account/Login.aspx**.
   b. In the Account/Login.aspx window, click the **Close** button.

▶ Task 9: Enable Forms authentication

1. Open the web.config file in the project root folder.
   - In Solution Explorer, right-click **web.config**, and then click **Open**.
2. Add to the web.config file an **authentication** element that specifies **Forms** authentication. Place the authentication element in the **system.web** element.
   - In the web.config window, add an **authentication** element to the **system.web** element, just below the closing **compilation** tag.

```
<authentication mode="Forms">
</authentication>
```

3. Add to the **authentication** element a **forms** element that specifies the value of the **loginUrl** attribute as **~/Account/Login.aspx**.
   - In the web.config window, add a **forms** element to the **authentication** element.

```
<forms loginUrl="~/Account/Login.aspx" />
```

4. Save the web.config file.
   - In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save web.config**, or press CTRL+S.

▶ Task 10: Test the Login controls

1. Stop the ASP.NET Development Server by using the ASP.NET Development Server icon in the System tray.
   - In the System Tray of the Windows taskbar, right-click **ASP.NET Development Server – Port 1111**, and then click **Stop**.
2. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.
3. View the Login controls.

> **Note:** Notice the Login controls at the upper-right corner of the window, which now display a login link. No user is currently signed in, because of the Forms authentication.

4. Open the **Login** Web Form by clicking the **Login** link.
   - In the Contoso Customer Management – Windows Internet Explorer window, click **Login**.
5. Close Internet Explorer.
   - In the Contoso Customer Management – User Login – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Implementing Authorization

▶ Task 1: Set up the connection to the membership database

1. Add a new connection string named **ApplicationServices** to the web.config file by adding an **add** element in the **connectionStrings** element.

```
<add name="ApplicationServices" connectionString="Data
Source=.\SQLEXPRESS;Integrated
Security=True;AttachDBFilename=|DataDirectory|\ASPNETDB.MDF;User
Instance=true" providerName="System.Data.SqlClient" />
```

   - In the web.config file, append the following markup in the **connectionStrings** element.

```
<add name="ApplicationServices" connectionString="Data
Source=.\SQLEXPRESS;Integrated
Security=True;AttachDBFilename=|DataDirectory|\ASPNETDB.MDF;User
Instance=true" providerName="System.Data.SqlClient" />
```

2. Save the web.config file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

▶ Task 2: Disallow anonymous access

1. Add an **authorization** element to the web.config file, after the **authentication** element.

```
<authorization>
</authorization>
```

   - In the web.config file, type the following markup,

```
<authorization>
</authorization>
```

   after the markup:

```
<authentication mode="Forms">
    <forms loginUrl="~/Account/Login.aspx" />
</authentication>
```

2. Add a self-closing **deny** element to the authorization element that disallows all anonymous users.

   - In the web.config file, type the following markup within the opening and closing tags of the **authorization** element.

```
<deny users="?" />
```

3. Save the web.config file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

▶ Task 3: Allow access to the Styles folder

1. Add a **location** element to the web.config file with a **path** attribute value of **Styles**, within the **configuration** element.

```
<location path="Styles">
</location>
```

   - In the web.config file, append the following markup after the closing tag of the **ConnectionStrings** element.

```
<location path="Styles">
</location>
```

2. Add a **system.web** element to the web.config file, in the **location** element.

```
<system.web>
</system.web>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **location** element.

```
<system.web>
</system.web>
```

3. Add an **authorization** element to the web.config file, in the **system.web** element.

```
<authorization>
</authorization>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **system.web** element.

```
<authorization>
</authorization>
```

4. Add an **allow** element to the web.config file **authorization** element with a **roles** attribute value of **Admins**.

```
<allow users="*"/>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **authorization** element.

```
<allow users="*"/>
```

5. Save the web.config file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

▶ Task 4: Create the default database for application services

1. In Solution Explorer, click the **ASP.NET Configuration** button.

> **Note:** This step will take awhile.

2. In the ASP.Net Web Administration- Windows Internet Explorer window, in the Home tab, click **Security**.
3. In the **Security** tab, click **Create user**.

4. In the **Security** tab, in the **Create user** section, complete the following settings, and then click **Create User**:
   - User Name: **student**
   - Password: **Pa$$w0rd**
   - Confirm Password: **Pa$$w0rd**
   - E-mail: **student@contoso.com**
   - Security Question: **Contoso Founder**
   - Security Answer: **John Contoso**
5. In the ASP.Net Web Administration- Windows Internet Explorer window, click the **Close** button.
6. In Solution Explorer, click the **Refresh** button, and then expand **App_Data**.

> **Note:** Notice that the App_Data folder now contains the default ASPNETDB.MDF database file that is used for the application services.

▶ Task 5: Test the anonymous access

1. Run the application.
   - In the CustomerManagement − Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.
2. View the **Login** Web Form.

> **Note:** Notice that the Login Web Form displays instead of the Default Web Form. This is because you have not yet logged in to the Web site.

3. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management − User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.

4. View the **Default** Web Form.

> **Note:** Notice that the Default Web Form displays after redirecting you from the Login Web Form.

5. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 6: Enable the security trimming of the site map

1. Add a **siteMap** element to the **system.web** element in the web.config file, with a **defaultProvider** attribute value of **AspNetXmlSiteMapProvider**, and an **enabled** attribute value of **true**.

```
<siteMap defaultProvider="AspNetXmlSiteMapProvider"
enabled="true">
</siteMap>
```

   - In the web.config file, type the following markup after the opening tag of the **system.web** element, which is contained in the **configuration** element.

```
<siteMap defaultProvider="AspNetXmlSiteMapProvider"
enabled="true">
</siteMap>
```

2. Add a **providers** element to the web.config file, in the **siteMap** element.

```
<providers>
</providers>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **siteMap** element.

```
<providers>
</providers>
```

3. Remove the default **AspNetXmlSiteMapProvider** provider element from the web.config file by using a self-closing **remove** element.

```
<remove name="AspNetXmlSiteMapProvider"/>
```

   - In the web.config file, type the following markup within the **providers** element.

```
<remove name="AspNetXmlSiteMapProvider"/>
```

4. Add a new **AspNetXmlSiteMapProvider** element to the web.config file by using a self-closing **add** element with a **type** attribute value of **System.Web.XmlSiteMapProvider, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a**, a **siteMap** attribute value of **web.sitemap**, and a **securityTrimmingEnabled** attribute value of **true**.

```
<add name="AspNetXmlSiteMapProvider"
    type="System.Web.XmlSiteMapProvider, System.Web,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
    siteMapFile="web.sitemap"
    securityTrimmingEnabled="true" />
```

   - In the web.config file, append the following markup to the **providers** element.

```
<add name="AspNetXmlSiteMapProvider"
    type="System.Web.XmlSiteMapProvider, System.Web,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"
    siteMapFile="web.sitemap"
    securityTrimmingEnabled="true" />
```

5. Save the web.config file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

▶ Task 7: Disallow access to the Import Countries page

1. Add a **location** element to the web.config file **configuration** element, with a **path** attribute value of **ImportCountries.aspx** .

```
<location path="ImportCountries.aspx">
</location>
```

- In the web.config file, append the following markup in the **configuration** element.

```
<location path="ImportCountries.aspx">
</location>
```

2. Add a **system.web** element to the web.config file, in the **location** element.

```
<system.web>
</system.web>
```

- In the web.config file, type the following markup within the opening and closing tags of the **location** element.

```
<system.web>
</system.web>
```

3. Add an **authorization** element to the web.config file, in the **system.web** element.

```
<authorization>
</authorization>
```

- In the web.config file, type the following markup within the opening and closing tags of the **system.web** element.

```
<authorization>
</authorization>
```

4. Add an **allow** element to the web.config file with a **roles** attribute value of **Admins** in the **authorization** element.

```
<allow roles="Admins"/>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **authorization** element.

```
<allow roles="Admins"/>
```

5. Add a **deny** element to the web.config file with a **users** attribute value of *, in the **authorization** element.

```
<deny users="*"/>
```

   - In the web.config file, append the following markup in the **authorization** element.

```
<deny users="*"/>
```

6. Save and close the web.config file.

   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

   b. In the web.config window, click the **Close** button.

▶ Task 8: Allow access to the Customers, Countries, and Help menu items

1. Open the web.sitemap file.

   - In Solution Explorer, right-click **web.sitemap**, and then click **Open**.

2. Set the **roles** attribute of the **Customers siteMapNode** element to a value of *.

   - In the web.sitemap file, locate the **siteMapNode** with a **title** attribute value of **Customers**, and add a **roles** attribute with a value of *.

```
roles="*"
```

3. Set the **roles** attribute of the **Countries siteMapNode** element to a value of *.
   - In the web.sitemap file, locate the **siteMapNode** with a **title** attribute value of **Countries**, and add a **roles** attribute with a value of *.

```
roles="*"
```

4. Set the **roles** attribute of the **Help siteMapNode** element to a value of *.
   - In the web.sitemap file, locate the **siteMapNode** with a **title** attribute value of **Help**, and add a **roles** attribute with a value of *.

```
roles="*"
```

▶ Task 9: Disallow access to the Import Countries menu item

1. Set the **roles** attribute of the **Import siteMapNode** element to a value of **Admins**.
   - In the web.sitemap file, locate the **siteMapNode** with a **title** attribute value of **Import**, and add a **roles** attribute with a value of **Admins**.

```
roles="Admins"
```

2. Save and close the web.sitemap file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**.
   b. In the web.sitemap window, click the **Close** button.

▶ Task 10: Test the access to the Import Countries menu item

1. Stop the ASP.NET Development Server by using the icon in the System tray.
   - In the System Tray of the Windows Taskbar, right-click **ASP.NET Development Server – Port 1111**, and click **Stop**.
2. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

3. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management – User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.
4. Notice that the **Import** menu item is missing.
   - In the Contoso Customer Management page, click the **Countries** menu, and verify that the Import menu item does not display.

> **Note:** Notice that the **Import** menu item is hidden, because the user student is not a member of the Admins role.

5. Browse to the **ImportCountries.aspx** Web Form by using the URL **http://localhost:1111/CustomerManagement/ImportCountries.aspx**.
   - In the Address bar of the Contoso Customer Management – Windows Internet Explorer window, type **http://localhost:1111/CustomerManagement/ImportCountries.aspx**, and then press ENTER.

> **Note:** Notice that you are redirected to the login page, because you do not have access to the ImportCountries page.

6. Close Internet Explorer.
   - In the Contoso Customer Management – User Login – Windows Internet Explorer window, click the **Close** button.

▶ Task 11: Add student to the Admins role

1. Open the ASP.NET Web Site Administration tool.
   - In Solution Explorer, click the **ASP.NET Configuration** button.
2. Enable roles by using the **Security** tab.
   a. In the ASP.NET Web Site Administration Tool page, click the **Security** tab.
   b. On the **Security** tab, under **Roles**, click **Enable roles**, and then click **Create or Manage roles**.

3. Create an **Admins** role.
   - In the **New role name** box, type **Admins**, and then click **Add Role**.
4. Add a student user to the **Admins** role.
   a. On the **Security** tab, under **Add/Remove Users**, click **Manage**.
   b. In the **Search for Users** area, click **S**.
   c. On the **Security** tab, select the **User Is In Role** check box next to the user name, **Student**.
5. Close Internet Explorer.
   - In the ASP.NET Web Application Administration – Windows Internet Explorer window, click the **Close** button.

▶ Task 12: Test the Import Country access

1. Stop the ASP.NET Development Server by using the ASP.NET Development Server icon in the System tray.
   - In the System Tray of the Windows Taskbar, right-click **ASP.NET Development Server – Port 1111**, and then click **Stop**.
2. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.
3. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management – User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.
4. View the **Import** menu item.
   - In the **Contoso Customer Management** page, click the **Countries** menu, and verify that the Import menu item is displayed.

> **Note**: Notice the **Import** menu item now displays, because the user, Student, is a member of the Admins role.

5. Open the **Import Countries** page.
   - In the **Contoso Customer Management** page, click the **Countries** menu, and then click **Import**.

> **Note:** Notice that the Import Countries page displays.

6. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
7. Close Microsoft Visual Studio 2010.
   - In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

Exercise 3: Protecting Configuration File

▶ Task 1: Grant the IIS Application Pool Identity access to the RSA key container

1. Open the Visual Studio 2010 command prompt as an Administrator.
    a. On the **Start** menu, point to **All Programs**, expand **Microsoft Visual Studio 2010**, expand **Visual Studio Tools**, right-click **Visual Studio Command Prompt (2010)**, and then click **Run as administrator**.
    b. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.
2. Grant the NETWORK SERVICE account access to the default machine-level **NetFrameworkConfigurationKey** RSA key container by running the following command from the Visual Studio 2010 Command Prompt window, which you should type in on a single line.

```
aspnet_regiis -pa "NetFrameworkConfigurationKey"
 "NT AUTHORITY\NETWORK SERVICE"
```

- At the command prompt of the Administrator: Visual Studio Command Prompt (2010) window, type the following command, and then press ENTER.

```
aspnet_regiis -pa "NetFrameworkConfigurationKey"
 "NT AUTHORITY\NETWORK SERVICE"
```

▶ Task 2: Copy the Web site

1. Open Microsoft® Visual Studio® 2010 as an Administrator.
    a. On the **Start** menu of **10267A-GEN-DEV**, point to **All Programs**, click **Microsoft Visual Studio 2010**, right-click **Microsoft Visual Studio 2010**, and then click **Run as administrator**.
    b. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.

2. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M15\VB folder.
   a. In the Start Page – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M15\VB\CustomerManagement.sln**, and then click **Open**.
3. Open the Copy Web Site tool.
   - In Solution Explorer of Visual Studio 2010, right-click **D:\Labfiles\Starter\M15\VB\CustomerManagement**, and then click **Copy Web Site**.
4. Connect to the CM application on the **Default Web Site** on the local IIS.
   a. In the Copy Web Site tool window, click **Connect**.
   b. In the **Open Web Site** dialog box, click **Local IIS**.
   c. In the **Select the Web site you want to open** box, expand **Default Web Site**, click **CM**, and then click **Open**.
5. Copy the source files to the destination site.
   - In the Source Web site pane, select all files and folders, and then click the **Copy selected files from source to remote web site** button.

> **Note:** Notice that all the modified files and folders from the Source Web site pane are copied to the Remote Web site pane.

6. Close the Copy Web Site tool.
   - In the Copy Web Site tool window, click the **Close** button.

▶ Task 3: Test the deployed Web site

1. Open Internet Explorer, and enter the URL **http://localhost:1112/CM**.
   a. On the **Start** menu, click **Internet Explorer**.
   b. In the Internet Explorer Address bar, type **http://localhost:1112/CM**, and then press ENTER.
2. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management – User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.
3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Encrypt the connectionStrings section in the configuration file

1. Encrypt the **connectionStrings** section of the web.config file for the deployed Web site, which has an application name of **CM**, by running the following command from the Visual Studio Command Prompt (2010).

```
aspnet_regiis -pef "connectionStrings" "C:\inetpub\wwwroot\CM"
```

   - At the command prompt of the Administrator: Visual Studio Command Prompt (2010) window, type the following command, and then press ENTER.

```
aspnet_regiis -pef "connectionStrings" "C:\inetpub\wwwroot\CM"
```

2. Close the Visual Studio 2010 Command Prompt.
   - In the Administrator: Visual Studio Command Prompt (2010) window, click the **Close** button.
3. In Windows Explorer, open the **web.config** file from the folder **C:\inetpub\wwwroot\CM**.
   a. On the **Start** menu, point to **All Programs**, click **Accessories**, and then click **Windows Explorer**.
   b. In Windows Explorer, navigate to **Computer\Local Disk (C:)\inetpub\wwwroot\CM**, and then double-click **web.config**.

4. In Visual Studio 2010, notice the **connectionStrings** element, which is now encrypted, and looks similar to the following markup.

```
<connectionStrings
configProtectionProvider="RsaProtectedConfigurationProvider">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
            xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <EncryptedKey
xmlns="http://www.w3.org/2001/04/xmlenc#">
                <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
                <KeyInfo
xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <KeyName>Rsa Key</KeyName>
                </KeyInfo>
                <CipherData>

<CipherValue>uARxpqWHFtXbkOCpRvcOcsfnluYD7osFsViSqCcGn/KTvfFfg1PJE
ZSd18zT/XH4GvzLhDukDHfUdqowZRnjCDhZpmrcNaovD3wK5NdWLdsloiTDsbaE5nL
tOC44fkesotaiVrRxiDlMoQPFwkqBYUPxRFZqj79mv4aKk3hvUFO=</CipherValue
>
                </CipherData>
            </EncryptedKey>
        </KeyInfo>
```

(*Code continued on the following page.*)

```
        <CipherData>

<CipherValue>yk3L/rAFdZiaJdEbznV6LlrOUkGLcvtc41NFunqrlEmxZfFnpRfC9
gdWXJbnrR16D8oKdLomM4S5MAHmzhLi9WHLIvyQyNET3ydR1VPj8GV4+O5PpRKVMYs
4TfgpDKpF+c+LXwUPQDFIWzuGxIuO4yaHdugi8qg7Yw9uGEvFKxI2JDGDnoXDPqfHM
GzTw/7Yiwcoz1G23vecBnGAL3NutjWFcRil2yHCE8fu886xG6C/92LxFp3LlmhtCaC
JO8hL6dN7DCG7BDvO6hMnLaV145PHSYY4szcTXKcuxGHa4zbDwTbPEaOfEkayojOmD
fiLOFZ8LfI1aIGLz+7H2DzzMPesjGgC2kzClisXz7WfqHKaQgOSnuHOFrLmTC7fLSS
29HwzoX5kWFObZtu7gJBlBCWn9JjmRXOjY8ixvm25+MhUxm2V5sBGKejhIgtg/q82t
KCBQyjf1dFx5EkvDLeO7AUgS5oLLiSOoJdyYQo6VQpgJ8PuLtQthM8+XDOC8Nwsv19
Sq4qfJw1YDX1pSAleScayjd4ZOAuW8nzyXzDodnTIhyQtRZldTJ8nuHWu1Q2mDq8Xm
tSMu/pzMoN2CBw6dkUUEsAek2ickDwxNeXxycWgrzblCHPksklbYwT1HME7LSQAISO
Z8OjIenYdeDVOKQmfbVWHdYDouGe3AZLYOPLzmkp8OHywxcGbjLnzEv5p44RCtRNgP
I9ZC2FQZ8ZvVDVDjUe9juRhsiPTJAyvsocPimrV1Xkd5KUSnOr1FCz2clizY4fefNf
in1HoKlMaF7ZuGc/OEiQ1EcGOzLRUetAC58fJDZYKYXcdlcKmuowsTiJyHFszM7BrJ
OQx7SYj3URoM844XrgK8bMxe2URPL+X29U8oyfI8/amsU9YWLlKIP9GO6826zSAKaD
mwZqgl+tCjfjmmncu9S3Qt2LRpRZo7RGDZBlot2z+IsfsWtVUq/MkB9nzq6213131lX
PIcYx2O5dvcSaJoqzO9UmXQxGLCDoQCMKJBq2aDIuLJgmmAsx9yXc2Waoyg1dmjHXc
2/bCia93MfePu9LbxixHqq3x27fOjWvUivr8YNmMWoDWwatYr8oxNvH+RzHieoPaes
eFNXF96QBGsjVOCY9YVHwdXZWvvpt32YChYSSgclDSQEDFMam15z2RvRb0uthvVC4O
Y8v5U2w2km9PLonG2TPhm452SR8MN+/CXXsTJ+c5SxDDcx19Hr1OMo+QtG3IOOxZPB
ovPzTfqvaxjpQVI9utbrIoiI4EwAhw+9qEFCSO5vK6RYhyLe2/MWOM+8uhNz90hxsg
6AKHpVV3OZzrpiROxRYyFeqWEbA==</CipherValue>
        </CipherData>
    </EncryptedData>
</connectionStrings>
```

5. Close the web.config file.
   - In the web.config window, click the **Close** button.

▶ Task 5: Test the deployed Web site

1. Open Internet Explorer, and enter the URL **http://localhost:1112/CM**.
   a. On the **Start** menu, click **Internet Explorer**.
   b. In the Address bar of Internet Explorer, type **http://localhost:1112/CM**, and the press ENTER.
2. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management – User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.

3. Show all customers.
   - In the **Contoso Customer Management** page, click the **Customers** menu, and then click **All**.
4. Close Internet Explorer.
   - In the Customers – Windows Internet Explorer window, click the **Close** button.
5. Close Microsoft Visual Studio 2010.
   - In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 6: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 15

Lab Answer Key: Securing a Microsoft® ASP.NET Web Application (Visual C#)

Contents:

Lab: Securing a Microsoft® ASP.NET Web Application

(C#)

Exercise 1: Enabling Forms Authentication

▶ Task 1: Open an existing ASP.NET Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of 10267A-GEN-DEV, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M15\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M15\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page for the project.
  - In Solution Explorer, right-click **Default.aspx**, and then click **Set As Start Page**.

▶ Task 3: Add the Login controls

1. Open the master page, **Site.master**, in the Source view.
   - In Solution Explorer, right-click **Site.master**, and then click **View Markup**.
2. Add a **div** element with a **class** attribute with the value of **login**, to the **body** element of the master page **Site.master**, after the **div** element with a **class** attribute set to the value of **appTitle**.

```
<div class="login">
</div>
```

   - In the Site.master window, locate the **div** element with a **class** attribute set to the value of **appTitle**, place the cursor after the closing **div** tag, press ENTER, and then type the following markup.

```
<div class="login">
</div>
```

3. Add a **LoginStatus** control named, **MainLoginStatus**, to the **div** element with a **class** attribute value of **login**.

```
<asp:LoginStatus ID="MainLoginStatus" runat="server"
LogoutAction="RedirectToLoginPage"
    LogoutPageUrl="~/Account/Login.aspx" />
```

Note: The Login.aspx Web Form does not yet exist. You will add one later in this exercise.

Note: The **LoginStatus** control must have the **LogoutAction** attribute set to the value of **RedirectToLoginPage** to send the user to the login page. Optionally, the attribute, **LogoutPageUrl**, can be set to the value of **~/Account/Login.aspx**, to redirect to this specific page, instead of the one specified in the web.config file.

   a. In the Site.master window, place the cursor at the end of the opening **div** tag that you have added, and then press ENTER.
   b. In the Site.master window, point to **Toolbox**.
   c. In Toolbox, expand **Login**, and then double-click **LoginStatus**.

d. Change the **ID** attribute of the **LoginStatus** control from **LoginStatus1** to **MainLoginStatus**.

e. Add a **LogoutAction** attribute, and set the value to **RedirectToLoginPage**.

f. Add a **LogoutPageUrl** attribute, and set the value to **~/Account/Login.aspx**.

```
<asp:LoginStatus ID="MainLoginStatus" runat="server"
LogoutAction="RedirectToLoginPage"
    LogoutPageUrl="~/Account/Login.aspx" />
```

4. Add a **LoginName** control named **MainLoginName** to the **div** element, after the **MainLoginStatus** control.

```
<asp:LoginName ID="MainLoginName" runat="server" />
```

a. In the Site.master window, place the cursor at the end of the self-closing **MainLoginStatus** element, and then press ENTER.

b. In the Site.master window, point to **Toolbox**.

c. In Toolbox, expand **Login**, and then double-click **LoginName**.

d. Change the **ID** attribute of the **LoginName** control from **LoginName1** to **MainLoginName**.

```
<asp:LoginName ID="MainLoginName" runat="server" />
```

5. Save and close the master page.

a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Site.master**.

b. In the Site.master window, click the **Close** button.

6. Add the following style to the Styles/Site.css stylesheet.

```
div.login
{
    position: fixed;
    top: 49px;
    right: 10px;
    padding-left: 10px;
    padding-bottom: 10px;
    background-color: #ffffff;
}
```

a. In Solution Explorer, expand **Styles**, and double-click **Site.css**.

b. In the Styles/Site.css window, press CTRL+END, press ENTER, and then type the following markup.

```
div.login
{
    position: fixed;
    top: 49px;
    right: 10px;
    padding-left: 10px;
    padding-bottom: 10px;
    background-color: #ffffff;
}
```

7. Save and close the Site.css file.

a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Styles/Site.css**, or press CTRL +S.

b. In the Styles/Site.css window, click the **Close** button.

▶ Task 4: Test the Login controls

1. Run the application.

- In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

2. View the Login controls.

Note: Notice the Login controls in the upper-right corner of the window, a **Logout** link, and the currently signed-in user, 10267A-GEN-DEV\student, because of the default Windows authentication.

3. Close Windows® Internet Explorer®.

- In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 5: Create a sample ASP.NET Web site

1. Open a second instance of Visual Studio 2010.
   - On the **Start** menu of **10267A-GEN-DEV**, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
2. Create a new file system-based ASP.NET Web site in **D:\Labfiles\Starter\M15\CS\SampleWebSite**, by using the ASP.NET Web Site template.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **New Web Site**.
   b. In the **New Web Site** dialog box, in the left pane, ensure that **Visual C#** is selected, and in the middle pane, click **ASP.NET Web Site**. In the **Web location** list click **File System**, in the adjacent text box, type **D:\Labfiles\Starter\M15\CS\SampleWebSite**, and then click **OK**.
3. Close second instance of Visual Studio 2010.
   - In the SampleWebSite – Microsoft Visual Studio window, click the **Close** button, or click the ALT+F4 keys.

▶ Task 6: Add the Account files and folders

1. Add the **Account** folder and default account content from the SampleWebSite Web site to the **CustomerManagement** Web site, by copying the Account folder from the path D:\Labfiles\Starter\M15\CS\SampleWebSite\Account. Use Windows Explorer to copy the Account folder.
   a. On the **Start** menu, click **Run**.
   b. In the **Run** dialog box, in the **Open** box, type **D:\Labfiles\Starter\M15\CS\SampleWebSite**, and then click **OK**.
   c. In the Windows Explorer window, right-click **Account**, and then click **Copy**.
   d. Right-click **CustomerManagement**, and then click **Paste**.
   e. Close the Windows Explorer window.

2. In Solution Explorer, refresh the Web site content to ensure the addition of the **Account** folder.
   - In Solution Explorer, right-click **D:\Labfiles\Starter\M15\CS\CustomerManagement**, and then click **Refresh Folder**.

▶ Task 7: Update Account Web Forms to use an existing master page

1. Expand the **Account** folder.
2. Open the **ChangePassword.aspx** Web Form in Source view.
   - In Solution Explorer, under **Account**, double-click **ChangePassword.aspx**.
3. Replace the name of the **ContentPlaceHolder** control referenced, **MainContent**, with the name of the ContentPlaceHolder control used in the existing master page, named **MainContentPlaceHolder**.
   - In the Account/ChangePassword.aspx window, replace **MainContent** in the **BodyContent** Content control with **MainContentPlaceHolder**.
4. Remove the **Content** control with an **ID** attribute value of **HeaderContent**.
   a. In the Account/ChangePassword.aspx window, locate and select the **Content** control with an **ID** attribute value of **HeaderContent**.

```
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
```

   b. On the **Edit** menu, click **Delete**, or press DELETE, to delete the selected markup.
5. Save and close the **ChangePassword.aspx** Web Form.
   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save Account/ChangePassword.aspx**, or press CTRL+SHIFT+S.
   b. In the Account/ChangePassword.aspx window, click the **Close** button.
6. Open the **ChangePasswordSuccess.aspx** Web Form in Source view.
   - In Solution Explorer, under **Account**, double-click **ChangePasswordSuccess.aspx**.

7. Replace the name of the **ContentPlaceHolder** control referenced, **MainContent**, with the name of the ContentPlaceHolder control used in the existing master page, **MainContentPlaceHolder**.

   - In the Account/ChangePasswordSuccess.aspx window, replace **MainContent** in the **BodyContent** Content control with **MainContentPlaceHolder**.

8. Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

   a. In the Account/ChangePasswordSuccess.aspx window, locate and select the **Content** control with an **ID** attribute value of **HeaderContent**.

   ```
   <asp:Content ID="HeaderContent" runat="server"
   ContentPlaceHolderID="HeadContent">
   </asp:Content>
   ```

   b. On the **Edit** menu, click **Delete**, or press DELETE, to delete the selected markup.

9. Save and close the **ChangePasswordSuccess.aspx** Web Form.

   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Account/ChangePasswordSuccess.aspx**, or press CTRL+SHIFT+S.

   b. In the Account/ChangePasswordSuccess.aspx window, click the **Close** button.

10. Open the **Login.aspx** Web Form in Source view.

    - In Solution Explorer, under **Account**, double-click **Login.aspx**.

11. Replace the name of the **ContentPlaceHolder** control named **MainContent**, with the name of the **ContentPlaceHolder** control used in the existing master page, named **MainContentPlaceHolder**.

    - In the Account/Login.aspx window, replace **MainContent** in the **BodyContent** Content control with **MainContentPlaceHolder**.

12. Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

    a. In the Account/Login.aspx window, locate and select the **Content** control with an **ID** attribute value of **HeaderContent**.

    ```
    <asp:Content ID="HeaderContent" runat="server"
    ContentPlaceHolderID="HeadContent">
    </asp:Content>
    ```

    b. On the **Edit** menu, click **Delete**, or press the DELETE key, to delete the selected markup.

13. Save and close the **Login.aspx** Web Form.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Account/Login.aspx**, or press CTRL+SHIFT+S.

    b. In the Account/Login.aspx window, click the **Close** button.

14. Open the **Register.aspx** Web Form in Source view.

    - In Solution Explorer, under **Account**, double-click **Register.aspx**.

15. Replace the name of the **ContentPlaceHolder** control named **MainContent**, with the name of the ContentPlaceHolder control used in the existing master page, named **MainContentPlaceHolder**.

    - In the Account/Register.aspx window, replace **MainContent** in the **BodyContent** Content control with **MainContentPlaceHolder**.

16. Remove the **Content** control with an **ID** attribute value of **HeaderContent**.

    a. In the Account/Register.aspx window, locate and select the **Content** control with an **ID** attribute value of **HeaderContent**.

    ```
    <asp:Content ID="HeaderContent" runat="server"
    ContentPlaceHolderID="HeadContent">
    </asp:Content>
    ```

    b. On the **Edit** menu, click **Delete**, or press DELETE, to delete the selected markup.

17. Save and close the **Register.aspx** Web Form.

    a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Account/Register.aspx**, or press CTRL+SHIFT+S.

    b. In the Account/Register.aspx window, click the **Close** button.

▶ Task 8: Modify the Login Web Form

1. Open the **Login** Web Form.
   - In Solution Explorer, under Account, double-click **Login.aspx**.
2. Set the page title as **Contoso Customer Management – User Login**.
   - In the Login.aspx window, locate the **Title** attribute of the **Page** directive, and replace **Log In** with **Contoso Customer Management - User Login**.
3. Save and close the **Login** Web Form.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save Account/Login.aspx**.
   b. In the Account/Login.aspx window, click the **Close** button.

▶ Task 9: Enable Forms authentication

1. Open the web.config file in the project root folder.
   - In Solution Explorer, right-click **web.config**, and then click **Open**.
2. Add to the web.config file an **authentication** element that specifies **Forms** authentication. Place the authentication element in the **system.web** element.
   - In the web.config window, add an **authentication** element to the **system.web** element, just below the closing **compilation** tag.

```
<authentication mode="Forms">
</authentication>
```

3. Add to the **authentication** element a **forms** element that specifies the value of the **loginUrl** attribute as **~/Account/Login.aspx**.
   - In the web.config window, add a **forms** element to the **authentication** element.

```
<forms loginUrl="~/Account/Login.aspx" />
```

4. Save the web.config file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**, or press CTRL+S.

▶ Task 10: Test the Login controls

1. Stop the ASP.NET Development Server by using the ASP.NET Development Server icon in the System tray.
   - In the System Tray of the Windows taskbar, right-click **ASP.NET Development Server – Port 1110**, and then click **Stop**.
2. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**, or press CTRL+F5.
3. View the **Login** controls.

> **Note:** Notice the Login controls at the upper-right corner of the window, which now display a login link. No user is currently signed in, because of the Forms authentication.

4. Open the **Login** Web Form by clicking the **Login** link.
   - In the Contoso Customer Management – Windows Internet Explorer window, click **Login**.
5. Close Internet Explorer.
   - In the Contoso Customer Management – User Login – Windows Internet Explorer window, click the **Close** button.

Exercise 2: Implementing Authorization

▶ Task 1: Set up the connection to the membership database

1. Add a new connection string named **ApplicationServices** to the web.config file by adding an **add** element in the **connectionStrings** element.

```
<add name="ApplicationServices" connectionString="Data
Source=.\SQLEXPRESS;Integrated
Security=True;AttachDBFilename=|DataDirectory|\ASPNETDB.MDF;User
Instance=true" providerName="System.Data.SqlClient" />
```

   - In the web.config file, append the following markup in the **connectionStrings** element.

```
<add name="ApplicationServices" connectionString="Data
Source=.\SQLEXPRESS;Integrated
Security=True;AttachDBFilename=|DataDirectory|\ASPNETDB.MDF;User
Instance=true" providerName="System.Data.SqlClient" />
```

2. Save the web.config file.

   - In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

▶ Task 2: Disallow anonymous access

1. Add an **authorization** element to the web.config file, after the **authentication** element.

```
<authorization>
</authorization>
```

   - In the web.config file, type the following markup,

```
<authorization>
</authorization>
```

   after the markup:

```
<authentication mode="Forms">
    <forms loginUrl="~/Account/Login.aspx" />
</authentication>
```

2. Add a self-closing **deny** element to the authorization element that disallows all anonymous users.
   - In the web.config file, type the following markup, within the opening and closing tags of the **authorization** element.

```
<deny users="?" />
```

3. Save the web.config file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

▶ Task 3: Allow access to the Styles folder

1. Add a **location** element to the web.config file with a **path** attribute value of **Styles**, within the **configuration** element.

```
<location path="Styles">
</location>
```

   - In the web.config file, append the following markup after the closing tag of the **ConnectionStrings** element.

```
<location path="Styles">
</location>
```

2. Add a **system.web** element to the web.config file, in the **location** element.

```
<system.web>
</system.web>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **location** element.

```
<system.web>
</system.web>
```

3. Add an **authorization** element to the web.config file, in the **system.web** element.

```
<authorization>
</authorization>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **system.web** element.

```
<authorization>
</authorization>
```

4. Add an **allow** element to the web.config file with a **users** attribute value of *****, in the **authorization** element.

```
<allow users="*"/>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **authorization** element.

```
<allow users="*"/>
```

5. Save the web.config file.

   - In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

▶ Task 4: Create the default database for application services

1. In Solution Explorer, click the **ASP.NET Configuration** button.

> **Note:** This step will take awhile.

2. In the ASP.Net Web Administration- Windows Internet Explorer window, in the Home tab, click **Security**.
3. In the **Security** tab, click **Create user**.

4. In the **Security** tab, in the **Create user** section, complete the following settings, and then click **Create User**:
   - User Name: **student**
   - Password: **Pa$$w0rd**
   - Confirm Password: **Pa$$w0rd**
   - E-mail: **student@contoso.com**
   - Security Question: **Contoso Founder**
   - Security Answer: **John Contoso**
5. In the ASP.Net Web Administration- Windows Internet Explorer window, click the **Close** button.
6. In Solution Explorer, click the **Refresh** button, and then expand **App_Data**.

> **Note:** Notice that the App_Data folder now contains the default ASPNETDB.MDF database file that is used for the application services.

▶ Task 5: Test the anonymous access

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.
2. View the **Login** Web Form.

> **Note:** Notice that the Login Web Form displays instead of the Default Web Form. This is because you have not yet logged in to the Web site.

3. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management – User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.

4. View the **Default** Web Form.

> **Note:** Notice that the Default Web Form is displays after redirecting you from the Login Web Form.

5. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 6: Enable the security trimming of the site map

1. Add a **siteMap** element to the **system.web** element in the web.config file, with a **defaultProvider** attribute value of **AspNetXmlSiteMapProvider** and an **enabled** attribute value of **true**.

```
<siteMap defaultProvider="AspNetXmlSiteMapProvider"
enabled="true">
</siteMap>
```

   - In the web.config file, type the following markup after the opening tag of the **system.web** element, which is contained in the **configuration** element.

```
<siteMap defaultProvider="AspNetXmlSiteMapProvider"
enabled="true">
</siteMap>
```

2. Add a **providers** element to the web.config file, in the **siteMap** element.

```
<providers>
</providers>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **siteMap** element.

```
<providers>
</providers>
```

3. Remove the default **AspNetXmlSiteMapProvider** provider element from the web.config file by using a self-closing **remove** element.

```
<remove name="AspNetXmlSiteMapProvider"/>
```

   - In the web.config file, type the following markup within the **providers** element.

```
<remove name="AspNetXmlSiteMapProvider"/>
```

4. Add a new **AspNetXmlSiteMapProvider** element to the web.config file by using a self-closing **add** element with a **type** attribute value of **System.Web.XmlSiteMapProvider, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a**, a **siteMap** attribute value of **web.sitemap**, and a **securityTrimmingEnabled** attribute value of **true**.

```
<add name="AspNetXmlSiteMapProvider"
    type="System.Web.XmlSiteMapProvider, System.Web,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
    siteMapFile="web.sitemap"
    securityTrimmingEnabled="true" />
```

   - In the web.config file, append the following markup in the **providers** element.

```
<add name="AspNetXmlSiteMapProvider"
    type="System.Web.XmlSiteMapProvider, System.Web,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"
    siteMapFile="web.sitemap"
    securityTrimmingEnabled="true" />
```

5. Save the web.config file.

   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

▶ Task 7: Disallow access to the Import Countries page

1. Add a **location** element to the web.config file **configuration** element, with a **path** attribute value of **ImportCountries.aspx**.

```
<location path="ImportCountries.aspx">
</location>
```

- In the web.config file, append the following markup in the **configuration** element.

```
<location path="ImportCountries.aspx">
</location>
```

2. Add a **system.web** element to the web.config file, in the **location** element.

```
<system.web>
</system.web>
```

- In the web.config file, type the following markup within the opening and closing tags of the **location** element.

```
<system.web>
</system.web>
```

3. Add an **authorization** element to the web.config file, in the **system.web** element.

```
<authorization>
</authorization>
```

- In the web.config file, type the following markup within the opening and closing tags of the **system.web** element.

```
<authorization>
</authorization>
```

4. Add an **allow** element to the web.config file **authorization** element with a **roles** attribute value of **Admins**.

```
<allow roles="Admins"/>
```

   - In the web.config file, type the following markup within the opening and closing tags of the **authorization** element.

```
<allow roles="Admins"/>
```

5. Add a **deny** element to the web.config file with a **users** attribute value of *, in the **authorization** element.

```
<deny users="*"/>
```

   - In the web.config file, append the following markup in the **authorization** element.

```
<deny users="*"/>
```

6. Save and close the web.config file.

   a. In the CustomerManagement − Microsoft Visual Studio window, on the **File** menu, click **Save web.config**.

   b. In the web.config window, click the **Close** button.

▶ Task 8: Allow access to the Customers, Countries, and Help menu items

1. Open the web.sitemap file.

   - In Solution Explorer, right-click **web.sitemap**, and then click **Open**.

2. Set the **roles** attribute of the **Customers siteMapNode** element to a value of *.

   - In the web.sitemap file, locate the **siteMapNode** with a **title** attribute value of **Customers**, and add a **roles** attribute with a value of *.

```
roles="*"
```

3. Set the **roles** attribute of the **Countries siteMapNode** element to a value of *.
   - In the web.sitemap file, locate the **siteMapNode** with a **title** attribute value of **Countries**, and add a **roles** attribute with a value of *.

```
roles="*"
```

4. Set the **roles** attribute of the **Help siteMapNode** element to a value of *.
   - In the web.sitemap file, locate the **siteMapNode** with a **title** attribute value of **Help**, and add a **roles** attribute with a value of *.

```
roles="*"
```

▶ Task 9: Disallow access to the Import Countries menu item

1. Set the **roles** attribute of the **Import siteMapNode** element to a value of **Admins**.
   - In the web.sitemap file, locate the **siteMapNode** with a **title** attribute value of **Import**, and add a **roles** attribute with a value of **Admins**.

```
roles="Admins"
```

2. Save and close the web.sitemap file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save web.sitemap**.
   b. In the web.sitemap window, click the **Close** button.

▶ Task 10: Test the access to the Import Countries menu item

1. Stop the ASP.NET Development Server by using the icon in the System tray.
   - In the System Tray of the Windows Taskbar, right-click **ASP.NET Development Server – Port 1110**, and click **Stop**.
2. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

3. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management – User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.
4. Notice that the **Import** menu item is missing.
   - In the Contoso Customer Management page, click the **Countries** menu, and verify that the Import menu item does not display.

> **Note:** Notice that the **Import** menu item is hidden, because the user student is not a member of the Admins role.

5. Browse to the **ImportCountries.aspx** Web Form by using the URL **http://localhost:1110/CustomerManagement/ImportCountries.aspx**.
   - In the Address bar of the Contoso Customer Management – Windows Internet Explorer window, type **http://localhost:1110/CustomerManagement/ImportCountries.aspx**, and then press ENTER.

> **Note:** Notice that you are redirected to the login page, because you do not have access to the ImportCountries page.

6. Close Internet Explorer.
   - In the Contoso Customer Management – User Login – Windows Internet Explorer window, click the **Close** button.

▶ Task 11: Add student to the Admins role

1. Open the ASP.NET Web Site Administration tool.
   - In Solution Explorer, click the **ASP.NET Configuration** button.
2. Enable roles by using the **Security** tab.
   a. In the ASP.NET Web Site Administration Tool page, click the **Security** tab.
   b. On the **Security** tab, under **Roles**, click **Enable roles**, and then click **Create or Manage roles**.

3. Create an **Admins** role.
   - In the **New role name** box, type **Admins**, and then click **Add Role**.
4. Add a student user to the **Admins** role.
   a. On the **Security** tab, under **Add/Remove Users**, click **Manage**.
   b. In the **Search for Users** area, click **S**.
   c. On the **Security** tab, select the **User Is In Role** check box next to the user name, **Student**.
5. Close Internet Explorer.
   - In the ASP.NET Web Application Administration – Windows Internet Explorer window, click the **Close** button.

▶ Task 12: Test the Import Country access

1. Stop the ASP.NET Development Server by using the icon in the System tray.
   - In the System Tray of the Windows Taskbar, right-click **ASP.NET Development Server – Port 1110**, and then click **Stop**.
2. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.
3. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management – User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.
4. View the **Import** menu item.
   - In the **Contoso Customer Management** page, click the **Countries** menu, and verify that the Import menu item is displayed.

Note: Notice the Import menu item now displays, because the user, Student, is a member of the Admins role.

5. Open the **Import Countries** page.
   - In the **Contoso Customer Management** page, click the **Countries** menu, and then click **Import**.

> **Note:** Notice that the Import Countries page now displays.

6. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.
7. Close Microsoft Visual Studio 2010.
   - In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

Exercise 3: Protecting Configuration File

▶ Task 1: Grant the IIS Application Pool Identity access to the RSA key container

1. Open the Visual Studio 2010 command prompt as an Administrator.
   a. On the **Start** menu, point to **All Programs**, expand **Microsoft Visual Studio 2010**, expand **Visual Studio Tools**, right-click **Visual Studio Command Prompt (2010)**, and then click **Run as administrator**.
   b. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.
2. Grant the NETWORK SERVICE account access to the default machine-level **NetFrameworkConfigurationKey** RSA key container by running the following command from the Visual Studio Command Prompt (2010) window, which should be typed in on a single line.

```
aspnet_regiis -pa "NetFrameworkConfigurationKey"
 "NT AUTHORITY\NETWORK SERVICE"
```

   - At the command prompt of the Administrator: Visual Studio Command Prompt (2010) window, type the following command, and then press ENTER.

```
aspnet_regiis -pa "NetFrameworkConfigurationKey"
 "NT AUTHORITY\NETWORK SERVICE"
```

▶ Task 2: Copy the Web site

1. Open Microsoft® Visual Studio® 2010 as an Administrator.
   a. On the **Start** menu of **10267A-GEN-DEV**, point to **All Programs**, click **Microsoft Visual Studio 2010**, right-click **Microsoft Visual Studio 2010**, and then click **Run as administrator**.
   b. In the **User Account Control** dialog box, in the text box, type **Pa$$w0rd**, and then click **Yes**.

2. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M15\CS folder.
   a. In the Start Page – Microsoft Visual Studio (Administrator) window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles \Starter\M15\CS\CustomerManagement.sln**, and then click **Open**.
3. Open the Copy Web Site tool.
   - In Solution Explorer of Visual Studio 2010, right-click **D:\Labfiles\Starter\M15\CS\CustomerManagement**, and then click **Copy Web Site**.
4. Connect to the **CM** application on the **Default Web Site** on the local Internet Information Services (IIS).
   a. In the Copy Web Site tool window, click **Connect**.
   b. In the **Open Web Site** dialog box, click **Local IIS**.
   c. In the **Select the Web site you want to open** box, expand **Default Web Site**, click **CM**, and then click **Open**.
5. Copy the source files to the destination site.
   - In the Source Web site pane, select all files and folders, and then click the **Copy selected files from source to remote web site** button.

> **Note:** Notice that all the modified files and folders from the Source Web site pane are copied to the Remote Web site pane.

6. Close the Copy Web Site tool.
   - In the Copy Web Site tool window, click the **Close** button.

▶ Task 3: Test the deployed Web site

1. Open Internet Explorer and enter the URL **http://localhost:1112/CM**.
   a. On the **Start** menu, click **Internet Explorer**.
   b. In the Address bar of Internet Explorer, type **http://localhost:1112/CM**, and then press ENTER.
2. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management – User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.
3. Close Internet Explorer.
   - In the Contoso Customer Management – Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Encrypt the connectionStrings section in the configuration file

1. Encrypt the **connectionStrings** section of the web.config file for the deployed Web site, which has an application name of **CM**, by running the following command from the Visual Studio Command Prompt (2010).

```
aspnet_regiis -pef "connectionStrings" "C:\inetpub\wwwroot\CM"
```

   - At the command prompt of the Administrator: Visual Studio Command Prompt (2010) window, type the following command, and then press ENTER.

```
aspnet_regiis -pef "connectionStrings" "C:\inetpub\wwwroot\CM"
```

2. Close the Visual Studio 2010 Command Prompt.
   - In the Administrator: Visual Studio Command Prompt (2010) window, click the **Close** button.

3. In Windows Explorer, open the **web.config** file from the folder **C:\inetpub\wwwroot\CM**.

   a. On the **Start** menu, point to **All Programs**, click **Accessories**, and then click **Windows Explorer**.

   b. In Windows Explorer, navigate to **Computer\Local Disk (C:)\inetpub\wwwroot\CM**, and then double-click **web.config**.

4. In Visual Studio 2010, notice the **connectionStrings** element, which is now encrypted, and looks similar to the following markup.

```
<connectionStrings
configProtectionProvider="RsaProtectedConfigurationProvider">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
            xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <EncryptedKey
xmlns="http://www.w3.org/2001/04/xmlenc#">
                <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
                <KeyInfo
xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <KeyName>Rsa Key</KeyName>
                </KeyInfo>
                <CipherData>

<CipherValue>uARxpqWHFtXbkOCpRvcOcsfnluYD7osFsViSqCcGn/KTvfFfg1PJE
ZSd18zT/XH4GvzLhDukDHfUdqowZRnjCDhZpmrcNaovD3wK5NdWLdsloiTDsbaE5nL
tOC44fkesotaiVrRxiDlMoQPFwkqBYUPxRFZqj79mv4aKk3hvUFO=</CipherValue
>
                </CipherData>
            </EncryptedKey>
        </KeyInfo>
```

(*Code continued on the following page.*)

```
        <CipherData>

<CipherValue>yk3L/rAFdZiaJdEbznV6LlrOUkGLcvtc41NFunqrlEmxZfFnpRfC9
gdWXJbnrR16D8oKdLomM4S5MAHmzhLi9WHLIvyQyNET3ydR1VPj8GV4+O5PpRKVMYs
4TfgpDKpF+c+LXwUPQDFIWzuGxIuO4yaHdugi8qg7Yw9uGEvFKxI2JDGDnoXDPqfHM
GzTw/7Yiwcoz1G23vecBnGAL3NutjWFcRil2yHCE8fu886xG6C/92LxFp3LlmhtCaC
JO8hL6dN7DCG7BDvO6hMnLaV145PHSYY4szcTXKcuxGHa4zbDwTbPEaOfEkayojOmD
fiLOFZ8LfI1aIGLz+7H2DzzMPesjGgC2kzClisXz7WfqHKaQgOSnuHOFrLmTC7fLSS
29HwzoX5kWFObZtu7gJBlBCWn9JjmRXOjY8ixvm25+MhUxm2V5sBGKejhIgtg/q82t
KCBQyjf1dFx5EkvDLeO7AUgS5oLLiSOoJdyYQo6VQpgJ8PuLtQthM8+XDOC8Nwsv19
Sq4qfJw1YDX1pSAleScayjd4ZOAuW8nzyXzDodnTIhyQtRZldTJ8nuHWu1Q2mDq8Xm
tSMu/pzMoN2CBw6dkUUEsAek2ickDwxNeXxycWgrzblCHPksklbYwT1HME7LSQAISO
Z8OjIenYdeDVOKQmfbVWHdYDouGe3AZLYOPLzmkp8OHywxcGbjLnzEv5p44RCtRNgP
I9ZC2FQZ8ZvVDVDjUe9juRhsiPTJAyvsocPimrV1Xkd5KUSnOr1FCz2clizY4fefNf
in1HoKlMaF7ZuGc/OEiQ1EcGOzLRUetAC58fJDZYKYXcdlcKmuowsTiJyHFszM7BrJ
OQx7SYj3URoM844XrgK8bMxe2URPL+X29U8oyfI8/amsU9YWLlKIP9GO6826zSAKaD
mwZqgl+tCjfjmmncu9S3Qt2LRpRZo7RGDZBlot2z+IsfsWtVUq/MkB9nzq6213131X
PIcYx2O5dvcSaJoqzO9UmXQxGLCDoQCMKJBq2aDIuLJgmmAsx9yXc2Waoyg1dmjHXc
2/bCia93MfePu9LbxixHqq3x27fOjWvUivr8YNmMWoDWwatYr8oxNvH+RzHieoPaes
eFNXF96QBGsjVOCY9YVHwdXZWvvpt32YChYSSgclDSQEDFMam15z2RvRb0uthvVC4O
Y8v5U2w2km9PLonG2TPhm452SR8MN+/CXXsTJ+c5SxDDcx19Hr1OMo+QtG3IOOxZPB
ovPzTfqvaxjpQVI9utbrIoiI4EwAhw+9qEFCSO5vK6RYhyLe2/MWOM+8uhNz90hxsg
6AKHpVV3OZzrpiROxRYyFeqWEbA==</CipherValue>
        </CipherData>
    </EncryptedData>
</connectionStrings>
```

5. Close the web.config file.
   - In the web.config window, click the **Close** button.

▶ Task 5: Test the deployed Web site

1. Open Internet Explorer, and enter the URL **http://localhost:1112/CM**.
   a. On the **Start** menu, click **Internet Explorer**.
   b. In the Address bar of Internet Explorer, type **http://localhost:1112/CM**, and the press ENTER.
2. Log in to the Web site as **Student**, with the password, **Pa$$w0rd**.
   - In the Contoso Customer Management – User Login window, in the **User Name** box, type **Student**, in the **Password** box, type **Pa$$w0rd**, and then click **Log In**.

3. Show all customers.
   - In the Contoso Customer Management page, click the **Customers** menu, and then click **All**.
4. Close Internet Explorer.
   - In the Customers – Windows Internet Explorer window, click the **Close** button.
5. Close Microsoft Visual Studio 2010.
   - In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 6: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 16

Lab Answer Key: Implementing Advanced Technologies Supported by Microsoft® Visual Studio® 2010 for Web Development (Visual Basic)

Contents:

Lab: Implementing Advanced Technologies Supported by Microsoft® Visual Studio® 2010 for Web Development

(Visual Basic)

Exercise: Implementing a Silverlight Application

▶ Task 1: Open an existing Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of **10267A-GEN-DEV**, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M16\VB folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M16\VB\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page of the project.
  - In Solution Explorer, right-click **Default.aspx**, and then click **Set As Start Page**.

▶ Task 3: Add a Silverlight project

- Add a Silverlight project to the **CustomerManagement** solution that is hosted by the existing **CustomerManagement** Web site. Use the default settings for creating the Silverlight project.

  a. In Solution Explorer, right-click **Solution 'CustomerManagement' (1 project)**, point to **Add**, and then click **New Project**.

  b. In the **Add New Project** dialog box, in the left pane, click **Visual Basic**.

  c. In the middle pane, click **Silverlight Application**, and ensure that **the .NET Framework 4** option is selected in the **Framework Version** list.

  d. In the **Name** box, type **CustomerManagementSL**, and then click **OK**.

  e. In the **New Silverlight Application** dialog box, ensure that the following, check boxes are selected, and then click **OK**.

     - **Host the Silverlight application in a new or existing Web site in the solution**
     - **Add a test page that references the application**
     - **Make it the start page**
     - **Enable Silverlight debugging (disables JavaScript debugging)**

▶ Task 4: Add a media file to the Silverlight application

1. Add the file **D:\Labfiles\Starter\M16\Robotica_720.wmv**, to the Silverlight application.

   a. In Solution Explorer, right-click **CustomerManagementSL**, point to **Add**, and then click **Existing Item**.

   b. In the **Add Existing Item – CustomerManagementSL** dialog box, navigate to **D:\Labfiles\Starter\M16**, click **Robotica_720.wmv**, and then click **Add**.

2. Add the media file to the application package as a resource.

   a. In Solution Explorer, click **Robotica_720.wmv**.

   b. In the Properties window, in the **Build Action** property, click **Resource**.

▶ Task 5: Add grid row definitions to the Grid control

1. In the MainPage.xaml document, add a **Grid.RowDefinitions** element to the existing **Grid** control.
   - In the MainPage.xaml window, add a **Grid.RowDefinitions** element in **Grid** control.

```
<Grid.RowDefinitions>
</Grid.RowDefinitions>
```

2. Add a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **50**.
   - In the MainPage.xaml window, add a **RowDefinition** element in the **Grid.RowDefinitions** element, with a **Height** attribute value of **50**.

```
<RowDefinition Height="50"/>
```

3. Append a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **32**.
   - In the MainPage.xaml window, add a **RowDefinition** element to the **Grid.RowDefinitions** element, with a **Height** attribute value of **32**, after the **RowDefinition** element with a **Height** attribute value of **50**.

```
<RowDefinition Height="32"/>
```

4. Append a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **Auto**.
   - In the MainPage.xaml window, add a **RowDefinition** element to the **Grid.RowDefinitions** element, with a **Height** attribute value of **Auto**, after the **RowDefinition** element with a **Height** attribute value of **32**.

```
<RowDefinition Height="Auto"/>
```

5. Save the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 6: Add grid column definitions to the Grid control

1. Add a **Grid.ColumnDefinitions** element to the existing **Grid** control.
   - In the MainPage.xaml window, add a **Grid.ColumnDefinitions** element in the **Grid** control.

```
<Grid.ColumnDefinitions>
</Grid.ColumnDefinitions>
```

2. Add a **ColumnDefinition** element to the **Grid.ColumnDefinitions** element with a **Width** attribute value of **Auto**.
   - In the MainPage.xaml window, add a **ColumnDefinition** element in the **Grid.ColumnDefinitions** element with a **Width** attribute value of **Auto**.

```
<ColumnDefinition Width="Auto" />
```

3. Save the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 7: Add the TextBlock control to the Grid control

1. In the MainPage.xaml document, append a **TextBlock** control to the **Grid** control.

> **Note:** The **TextBlock** control must have a **Text** attribute value of **Contoso Customer Management**, and it must be displayed in the first row.

```
<TextBlock Text="Contoso Customer Management" Grid.Row="0" />
```

   - In the MainPage.xaml window, add a **TextBlock** control in the **Grid** control, after the closing **Grid.ColumnDefinitions** tag.

```
<TextBlock Text="Contoso Customer Management" Grid.Row="0" />
```

2. Save the MainPage.xaml file.

- In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 8: Add a Button control to the Grid control

1. In the MainPage.xaml document, append a **Button** control to the **Grid** control.

> **Note:** The **Button** control must have a **Content** attribute value of **Play Media**, a **Margin** attribute value of **10,3,10,3**, it must be 75 pixels wide and 25 pixels high, aligned to left, and it must be displayed in the second row.

```
<Button Width="75" Height="25" HorizontalAlignment="Left"
Content="Play Media" Grid.Row="1" Margin="10,3,10,3" />
```

- In the MainPage.xaml window, add a **Button** control in the **Grid** control, after the **TextBlock** element.

```
<Button Width="75" Height="25" HorizontalAlignment="Left"
Content="Play Media" Grid.Row="1" Margin="10,3,10,3" />
```

2. Name the **Button** control **PlayButton**, by adding an **x:Name** attribute to the **Button** element.

```
x:Name="PlayButton"
```

3. Save the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 9: Add a MediaElement control to the Grid control

1. In the MainPage.xaml document, append a **MediaElement** control to the **Grid** control.

> **Note:** The **MediaElement** control must have a **Source** attribute value of **Robotica_720.wmv**, an **AutoPlay** attribute value of **False**, a **Margin** attribute value of **10,3,10,3**, it must be vertically aligned to the top of the Grid control, and it must be displayed in the third row.

```
<MediaElement AutoPlay="False" Source="Robotica_720.wmv"
Grid.Row="2" Margin="10,3,10,3" VerticalAlignment="Top" />
```

- In the MainPage.xaml window, add a **MediaElement** element in the **Grid** control, after the **Button** element.

```
<MediaElement AutoPlay="False" Source="Robotica_720.wmv"
Grid.Row="2" Margin="10,3,10,3" VerticalAlignment="Top" />
```

2. Name the **MediaElement** control **MainMediaElement**, by adding an **x:Name** attribute to the **MediaElement** element.

```
x:Name="MainMediaElement"
```

3. Save the MainPage.xaml file.

- In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 10: Add a resources definition to the Grid control

1. Add a **Grid.Resources** element at the top of the existing **Grid** control in the MainPage.xaml document.

- In the MainPage.xaml window, add a **Grid.Resources** element in the **Grid** control below the opening **Grid** tag.

```
<Grid.Resources>
</Grid.Resources>
```

2. Add a **Style** element to the **Grid.Resources** element with a **TargetType** attribute value of **TextBlock**.

- In the MainPage.xaml window, add a **Style** element in the **Grid.Resources** element with a **TargetType** attribute value of **TextBlock**.

```
<Style TargetType="TextBlock">
</Style>
```

3. Name the **Style** element **AppTitleStyle** by adding an **x:Key** attribute to the opening **Style** tag.

```
x:Key="AppTitleStyle"
```

4. Add a **Setter** element to the **Style** element with a **Property** attribute value of **FontFamily**, and a **Value** attribute value of **Trebuchet MS**, **Arial**, **sans-serif**.

```
<Setter Property="FontFamily" Value="Trebuchet MS, Arial, sans-
serif" />
```

5. Append a **Setter** element to the **Style** element with a **Property** attribute value of **FontSize**, and a **Value** attribute value of **30**.

```
<Setter Property="FontSize" Value="30" />
```

6. Append a **Setter** element to the **Style** element with a **Property** attribute value of **Foreground**, and a **Value** attribute value of **#888888**.

```
<Setter Property="Foreground" Value="#888888" />
```

7. Append a **Setter** element to the **Style** element with a **Property** attribute value of **Margin**, and a **Value** attribute value of **10,3,10,3**.

```
<Setter Property="Margin" Value="10,3,10,3" />
```

8. Format the markup by pressing the CTRL+K keys, and then pressing the CTRL+D keys.
9. Save the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 11: Apply static styles to the TextBlock control

1. In the MainPage.xaml document, add a **Style** attribute with a value of **{StaticResource AppTitleStyle}** to the **TextBlock** control.
   - In the MainPage.xaml window, locate the **TextBlock** element, and add the following markup.

```
Style="{StaticResource AppTitleStyle}"
```

2. Save the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 12: Add an Event Handler for the Button click event

1. Add a **Click** attribute to the **Button** control with the value of **PlayButton_Click**.
   - In the MainPage.xaml window, locate the **Button** element, and type the following markup,

     ```
     Click=
     ```

     and then, double-click the **New Event Handler** item in the context menu.
2. Save and close the MainPage.xaml file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.
   b. In the MainPage.xaml window, click the **Close** button.
   c. In the **Microsoft Visual Studio** dialog box, click **Yes**.
3. Open the code-behind file for the MainPage.xaml file.
   - In Solution Explorer, right-click **MainPage.xaml**, and then click **View Code**.
4. Add code to play the media file when the button is clicked.

   ```
   MainMediaElement.Play()
   ```

   - In the MainPage.xaml.vb window, locate the **PlayButton_Click** event handler, and add the following code.

     ```
     MainMediaElement.Play()
     ```
5. Save and close the code-behind file.
   a. In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml.vb**.
   b. In the MainPage.xaml.vb window, click the **Close** button.

▶ Task 13: Test the Silverlight application

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

> **Note:** Notice the progress indicator showing the download of the Silverlight application package to the Web site.

2. Play the media file.
   - In the CustomerManagementSL window, click **Play Media**.

> **Note:** Notice the media is now playing with both video and audio.

3. Close Windows® Internet Explorer®.
   a. In the CustomerManagementSL – Windows Internet Explorer window, click the **Close** button.
   b. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 14: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.

Module 16

Lab Answer Key: Implementing Advanced Technologies Supported by Microsoft® Visual Studio® 2010 for Web Development (Visual C#)

Contents:

Lab: Implementing Advanced Technologies Supported by Microsoft® Visual Studio® 2010 for Web Development

(C#)

Exercise: Implementing a Silverlight Application

▶ Task 1: Open an existing Web site

1. Log on to the **10267A-GEN-DEV** virtual machine as **Student**, with the password, **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010.
   - On the **Start** menu of **10267A-GEN-DEV**, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Open the **CustomerManagement** solution from the D:\Labfiles\Starter \M16\CS folder.
   a. In the Start Page – Microsoft Visual Studio window, on the **File** menu, click **Open Project**.
   b. In the **Open Project** dialog box, in the **File name** box, type **D:\Labfiles\Starter\M16\CS\CustomerManagement.sln**, and then click **Open**.

▶ Task 2: Set the start page

- Set the **Default** Web Form as the start page of the project.
  - In Solution Explorer, right-click **Default.aspx**, and then click **Set As Start Page**.

▶ Task 3: Add a Silverlight project

- Add a Silverlight project to the **CustomerManagement** solution that is hosted by the existing **CustomerManagement** Web site. Use the default settings for creating the Silverlight project.

    a. In Solution Explorer, right-click **Solution 'CustomerManagement' (1 project)**, point to **Add**, and then click **New Project**.

    b. In the **Add New Project** dialog box, in the left pane, click **Visual C#**.

    c. In the middle pane, click **Silverlight Application**, and ensure that the **.NET Framework 4** option is selected in the **Framework Version** list.

    d. In the **Name** box, type **CustomerManagementSL**, and then click **OK**.

    e. In the **New Silverlight Application** dialog box, ensure that the following check boxes are selected, and then click **OK**.

    - **Host the Silverlight application in a new or existing Web site in the solution**
    - **Add a test page that references the application**
    - **Make it the start page**
    - **Enable Silverlight debugging (disables JavaScript debugging)**

▶ Task 4: Add a media file to the Silverlight application

1. Add the file **D:\Labfiles\Starter\M16\Robotica_720.wmv**, to the Silverlight application.

    a. In Solution Explorer, right-click **CustomerManagementSL**, point to **Add**, and then click **Existing Item**.

    b. In the **Add Existing Item – CustomerManagementSL** dialog box, navigate to **D:\Labfiles\Starter\M16**, click **Robotica_720.wmv**, and then click **Add**.

2. Add the media file to the application package as a resource.

    a. In Solution Explorer, click **Robotica_720.wmv**.

    b. In the Properties window, in the **Build Action** property, click **Resource**.

▶ Task 5: Add grid row definitions to the Grid control

1. In the MainPage.xaml document, add a **Grid.RowDefinitions** element to the existing **Grid** control.
   - In the MainPage.xaml window, add a **Grid.RowDefinitions** element in **Grid** control.

```
<Grid.RowDefinitions>
</Grid.RowDefinitions>
```

2. Add a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **50**.
   - In the MainPage.xaml window, add a **RowDefinition** element in the **Grid.RowDefinitions** element, with a **Height** attribute value of **50**.

```
<RowDefinition Height="50"/>
```

3. Append a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **32**.
   - In the MainPage.xaml window, add a **RowDefinition** element to the **Grid.RowDefinitions** element, with a **Height** attribute value of **32**, after the **RowDefinition** element with a **Height** attribute value of **50**.

```
<RowDefinition Height="32"/>
```

4. Append a **RowDefinition** element to the **Grid.RowDefinitions** element with a **Height** attribute value of **Auto**.
   - In the MainPage.xaml window, add a **RowDefinition** element to the **Grid.RowDefinitions** element, with a **Height** attribute value of **Auto**, after the **RowDefinition** element with a **Height** attribute value of **32**.

```
<RowDefinition Height="Auto"/>
```

5. Save the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 6: Add grid column definitions to the Grid control

1. Add a **Grid.ColumnDefinitions** element to the existing **Grid** control.
   - In the MainPage.xaml window, add a **Grid.ColumnDefinitions** element in the **Grid** control.

```
<Grid.ColumnDefinitions>
</Grid.ColumnDefinitions>
```

2. Add a **ColumnDefinition** element to the **Grid.ColumnDefinitions** element with a **Width** attribute value of **Auto**.
   - In the MainPage.xaml window, add a **ColumnDefinition** element in the **Grid.ColumnDefinitions** element with a **Width** attribute value of **Auto**.

```
<ColumnDefinition Width="Auto" />
```

3. Save the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 7: Add the TextBlock control to the Grid control

1. In the MainPage.xaml document, append a **TextBlock** control to the **Grid** control.

> **Note:** The **TextBlock** control must have a **Text** attribute value of **Contoso Customer Management**, and it must be displayed in the first row.

```
<TextBlock Text="Contoso Customer Management" Grid.Row="0" />
```

   - In the MainPage.xaml window, add a **TextBlock** control in the **Grid** control, after the closing **Grid.ColumnDefinitions** tag.

```
<TextBlock Text="Contoso Customer Management" Grid.Row="0" />
```

2. Save the MainPage.xaml file.

- In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 8: Add a Button control to the Grid control

1. In the MainPage.xaml document, append a **Button** control to the **Grid** control.

> **Note:** The **Button** control must have a **Content** attribute value **of Play Media**, a **Margin** attribute value of **10,3,10,3**, it must be 75 pixels wide and 25 pixels high, aligned to left, and it must be displayed in the second row.

```
<Button Width="75" Height="25" HorizontalAlignment="Left"
Content="Play Media" Grid.Row="1" Margin="10,3,10,3" />
```

- In the MainPage.xaml window, add a **Button** control in the **Grid** control, after the **TextBlock** element.

```
<Button Width="75" Height="25" HorizontalAlignment="Left"
Content="Play Media" Grid.Row="1" Margin="10,3,10,3" />
```

2. Name the **Button** control **PlayButton** by adding an **x:Name** attribute to the **Button** element.

```
x:Name="PlayButton"
```

3. Save the MainPage.xaml file.

- In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 9: Add a MediaElement control to the Grid control

1. In the MainPage.xaml document, append a **MediaElement** control to the **Grid** control.

> **Note:** The **MediaElement** control must have a **Source** attribute value of **Robotica_720.wmv**, an **AutoPlay** attribute value of **False**, a **Margin** attribute value of **10,3,10,3**, it must be vertically aligned to the top of the Grid control, and it must be displayed in the third row.

```
<MediaElement AutoPlay="False" Source="Robotica_720.wmv"
Grid.Row="2" Margin="10,3,10,3" VerticalAlignment="Top" />
```

- In the MainPage.xaml window, add a **MediaElement** element in the **Grid** control, after the **Button** element.

```
<MediaElement AutoPlay="False" Source="Robotica_720.wmv"
Grid.Row="2" Margin="10,3,10,3" VerticalAlignment="Top" />
```

2. Name the **MediaElement** control **MainMediaElement** by adding an **x:Name** attribute to the **MediaElement** element.

```
x:Name="MainMediaElement"
```

3. Save the MainPage.xaml file.

- In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 10: Add a resources definition to the Grid control

1. Add a **Grid.Resources** element at the top of the existing **Grid** control in the MainPage.xaml document.

- In the MainPage.xaml window, add a **Grid.Resources** element in the **Grid** control below the opening **Grid** tag.

```
<Grid.Resources>
</Grid.Resources>
```

2. Add a **Style** element to the **Grid.Resources** element with a **TargetType** attribute value of **TextBlock**.

- In the MainPage.xaml window, add a **Style** element in the **Grid.Resources** element with a **TargetType** attribute value of **TextBlock**.

```
<Style TargetType="TextBlock">
</Style>
```

3. Name the **Style** element, **AppTitleStyle** by adding an **x:Key** attribute to the opening **Style** tag.

```
x:Key="AppTitleStyle"
```

4. Add a **Setter** element to the **Style** element with a **Property** attribute value of **FontFamily**, and a **Value** attribute value of **Trebuchet MS, Arial, sans-serif**.

```
<Setter Property="FontFamily" Value="Trebuchet MS, Arial, sans-
serif" />
```

5. Append a **Setter** element to the **Style** element with a **Property** attribute value of **FontSize**, and a **Value** attribute value of **30**.

```
<Setter Property="FontSize" Value="30" />
```

6. Append a **Setter** element to the **Style** element with a **Property** attribute value of **Foreground**, and a **Value** attribute value of **#888888**.

```
<Setter Property="Foreground" Value="#888888" />
```

7. Append a **Setter** element to the **Style** element with a **Property** attribute value of **Margin**, and a **Value** attribute value of **10,3,10,3**.

```
<Setter Property="Margin" Value="10,3,10,3" />
```

8. Format the markup by pressing the CTRL+K keys, and then pressing the CTRL+D keys.
9. Save the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 11: Apply static styles to the TextBlock control

1. In the MainPage.xaml document, add a **Style** attribute with a value of **{StaticResource AppTitleStyle}** to the **TextBlock** control.
   - In the MainPage.xaml window, locate the **TextBlock** element, and add the following markup.

```
Style="{StaticResource AppTitleStyle}"
```

2. Save the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.

▶ Task 12: Add an Event Handler for the Button click event

1. Add a **Click** attribute to the **Button** control with the value of **PlayButton_Click**.
   - In the MainPage.xaml window, locate the **Button** element, and type the following markup,

     ```
     Click=
     ```

     and then, double-click the **New Event Handler** item in the context menu.
2. Save and close the MainPage.xaml file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml**.
3. In the MainPage.xaml window, click the **Close** button.
4. In the **Microsoft Visual Studio** dialog box, click **Yes**.
5. Open the code-behind file for the MainPage.xaml file.
   - In Solution Explorer, right-click **MainPage.xaml**, and then click **View Code**.
6. Add code to play the media file when the button is clicked.

   ```
   MainMediaElement.Play();
   ```

   - In the MainPage.xaml.cs window, locate the **PlayButton_Click** event handler, and add the following code.

     ```
     MainMediaElement.Play();
     ```
7. Save and close the code-behind file.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **File** menu, click **Save MainPage.xaml.cs**.
8. In the MainPage.xaml.cs window, click the **Close** button.

▶ Task 13: Test the Silverlight application

1. Run the application.
   - In the CustomerManagement – Microsoft Visual Studio window, on the **Debug** menu, click **Start Without Debugging**.

> **Note:** Notice the progress indicator showing the download of the Silverlight application package to the Web site.

2. Play the media file.
   - In the CustomerManagementSL window, click **Play Media**.

> **Note:** Notice the media is now playing with both video and audio.

3. Close Windows® Internet Explorer®.
   a. In the CustomerManagementSL – Windows Internet Explorer window, click the **Close** button.
   b. In the CustomerManagement – Microsoft Visual Studio window, click the **Close** button.

▶ Task 14: Turn off the virtual machine and revert the changes

1. In Microsoft Hyper-V™ Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Turn Off**.
2. In the **Turn Off Machine** dialog box, click **Turn Off**.
3. In Hyper-V Manager, in the Virtual Machines pane, right-click **10267A-GEN-DEV**, and then click **Revert**.
4. In the **Revert Virtual Machine** dialog box, click **Revert**.