msdn® training

Module 10: Creating a Web Application with Web Forms

Contents

Overview	1
Lesson: Creating a Web Forms Application	2
Lesson: Accessing Data by Using a Web Forms Application	19
Lesson: Configuring ASP.NET Application Settings	29
Review	38
Lab 10.1: Developing an ASP.NET Web Application	40
Course Evaluation	46

Microsoft®

Instructor Notes

Presentation:
60 minutes

Lab:
60 minutes

In this module, students learn that in Microsoft® Visual Studio® .NET, you can use Web Forms to create programmable Web pages. This module introduces the **System.Web.UI** namespace and describes how to create a Web application with a Web Form. Students learn how to add controls to a Web Form and then use the Web Form to submit data and respond to events. The module also covers Microsoft ASP.NET state management, security, and configuration settings.

It is helpful here to point out the similarities and differences between Microsoft Windows® Forms and Web Forms.

Important This module includes guided, hands-on, and matching practices. Each practice or the lab may also include optional tasks to accommodate advanced learners.

After completing this module, students will be able to:

- Create a Web Forms application.
- Handle events on a Web Forms application.
- Access data from a Web Form application.
- Configure ASP.NET application settings.

Required materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2609A_010.ppt
- Module 10, Creating a Web Application with Web Forms
- Multimedia animation 2609A_ASP_NETExec.exe

Preparation tasks

To prepare for this module:

- Read all of the materials for this module.
- Review the multimedia demonstration.
- Complete the practices and lab.

How to Teach This Module

This section contains information that will help you to teach this module.

Lesson: Creating a Web Forms Application

This section describes the instructional methods for teaching each topic in this lesson.

This lesson points to ASP.NET as the hidden technology that operates behind Web applications. Explain to your students that just as the user interface of a Windows-based application is made up of Windows Forms, an ASP.NET Web application user interface is made up of Web Forms. Avoid detailed discussion of this technology to keep the students focused on their tasks.

You may consider using an instructor-led demonstration similar to the one that is provided for Windows Forms to show the students how to add a control to a Web Form and how to add an event handler for the control.

A hands-on practice concludes this lesson.

Lesson: Accessing Data by Using a Web Forms Application

In this lesson, students learn how to use Microsoft ADO.NET to access data by using a Web Forms application. Point out the similarities of accessing data by using Windows Forms and Web Forms.

A guided practice concludes this lesson.

Lesson: Configuring ASP.NET Application Settings

The animation that is provided for this lesson, *ASP.NET Execution Model*, depicts how clients' requests are processed by servers that use ASP.NET. To run this animation, click the icon in the center of the slide.

A short matching practice concludes this lesson.

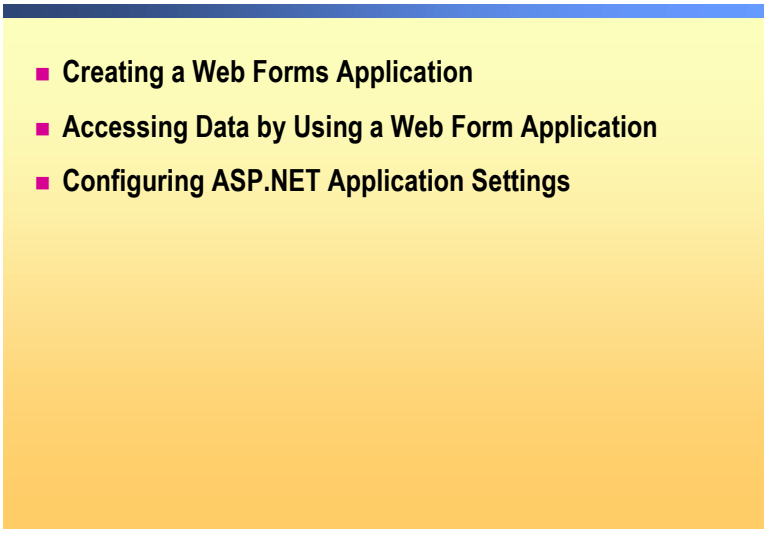
Review

The review questions are based mostly on conceptual understanding and procedures that were covered thoroughly in the module. You can use a discussion format to answer the questions so that everyone gets the benefit of knowing the right answers.

Lab 10.1: Developing an ASP.NET Web Application

Before beginning this lab, students should have completed all of the practices and answered the review questions. Students must be able to perform most of the tasks that they learned in the lessons and the practices. The lab is simple but comprehensive. It leads students through the entire process of creating a Web application with Web Forms as described in the lessons of this module.

Overview



Introduction

Microsoft® Visual Studio® .NET allows you to create applications that take advantage of the important features of the World Wide Web. These features include traditional Web sites that use HTML pages, fully-featured business applications that run on an intranet or the Internet, and sophisticated business-to-business applications that provide Web-based components that can exchange data by using Extensible Markup Language (XML).

In Visual Studio .NET, you can use Web Forms to create powerful, programmable Web pages. These Web pages serve as the user interface for your Web application. This module introduces the **System.Web.UI** namespace and describes how to create a Web application with a Web Form. The module also explains how to add controls to a Web Form and then use the Web Form to submit data and respond to events. The module also covers Microsoft ASP.NET state management, security, and configuration settings.

Objectives

After completing this module, you will be able to:

- Create a Web Forms application.
- Handle events on a Web Form application.
- Access data from a Web Forms application.
- Configure ASP.NET application settings.

Lesson: Creating a Web Forms Application

- What Is ASP.NET?
- What Is a Web Forms Application?
- How to Create a Web Forms Application
- What Are the Components of a Web Forms Application?
- What Is the Life Cycle of a Web Forms Application?
- How to Add Controls to a Web Forms Application
- How to Add an Event Handler for the Control

Introduction

This lesson introduces Web Forms and describes how to create a Web Form, add controls to it, and add event handlers for the controls.

Lesson objectives

After completing this lesson, you will be able to:

- Explain ASP.NET.
- Create a Web Form and add controls.
- Write event handlers for the controls.

Lesson agenda

This lesson includes the following topics and activity:

- What Is ASP.NET?
- What Is a Web Forms Application?
- How to Create a Web Forms Application
- What Are the Components of a Web Forms Application?
- What Is the Life Cycle of a Web Forms Application?
- How to Add Controls to a Web Forms Application
- How to Add an Event Handler for the Control
- Practice: Creating a Web Forms Application

What Is ASP.NET?

- Evolutionary, more flexible successor to Active Server Pages (ASP)
- Dynamic Web pages that can access server resources
- Server-side processing of Web forms
- Language independent
- Browser independent
- XML Web services let you create distributed Web applications

Introduction

For many years, developers have used Active Server Pages (ASP) technology to build dynamic Web pages. ASP.NET is the logical development of ASP; it runs on a Web server and provides a way for you to develop content-rich, dynamic, personalized Web sites by using the power of Microsoft .NET.

Definition

ASP.NET is a Web run-time environment that is built on top of .NET. ASP technology mixed HTML and script together in the same document. In ASP.NET, the code, which can be any .NET-compatible language, is held separately from the HTML page.

A new component of ASP.NET is the Web Form. As the user interface to an application based on Microsoft Windows® is made up of Windows Forms, an ASP.NET Web application user interface is made up of Web Forms. An ASP.NET Web application includes one or more Web Forms.

XML Web services

The ASP.NET technology provides the platform for running XML Web services. XML Web services allow distributed applications to transfer information between clients, applications, and other XML Web services. It is also possible from within an ASP.NET application to consume XML Web services from other servers.

What Is a Web Forms Application?

- Based on ASP.NET technology to create powerful programmable Web pages
- Compatible with any browser or mobile device
- Compatible with any language supported by common language runtime
- Allow for separation between code and content on a page
- Support a rich set of controls
- Provide a set of state management features that preserve the view state of a Web page between requests

Introduction

Just as you use Windows Forms to create Windows-based applications, you can use Web Forms to build powerful programmable Web pages dynamically. Web Forms pages are built with ASP.NET technology. You can add Web Forms pages to several types of Visual Studio .NET projects. Most often, when you want to work with Web Forms pages, you will use the project template for the ASP.NET Web Application.

Definition

A Web Form is a dynamic Web page, which users view in a browser that can access server resources.

Features

Web Forms Applications:

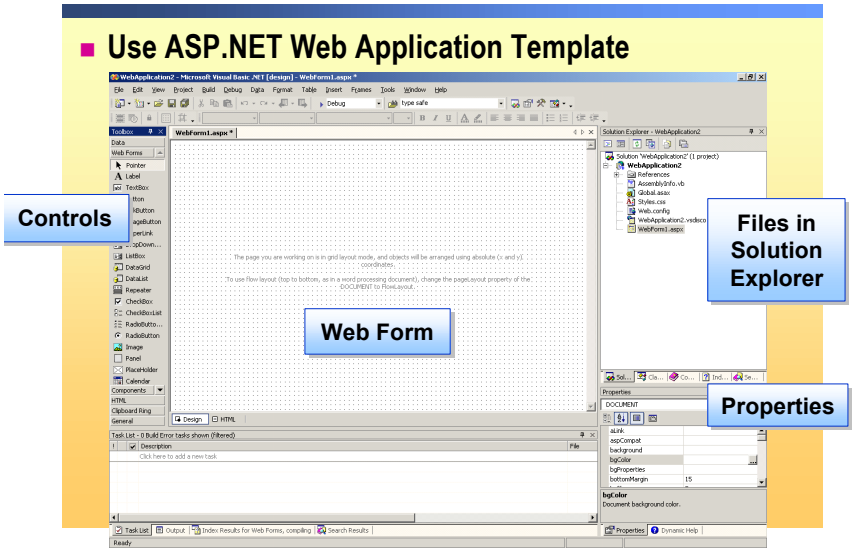
- Are based on ASP.NET technology to create powerful programmable Web pages.
- Run on any browser and automatically render the correct, browser-compliant HTML code for features such as styles and layout.
- Are programmable in any language that the common language runtime supports, including C#, Microsoft Visual Basic®, and Microsoft JScript® .NET.
- Support WYSIWYG (what you see is what you get) editing tools and powerful rapid application development (RAD) tools, such as Visual Studio .NET, for designing and programming your forms.
- Provide a rich set of controls that allow you to encapsulate page logic into reusable components and declaratively handle page events.

State management

Web Forms provide a set of state management features that automatically preserve the view state of a page between requests. *State* refers to the information that an application must maintain about a Web page.

When a Web server receives a request for a page, it finds the page, processes it, sends it to the browser, and then discards all page information. If the user requests the same page again, the server repeats the entire sequence, reprocessing the page from the beginning. Servers have no memory of the pages that they have processed. Therefore, if an application must maintain information about a page, you must provide for it in application code. Web Forms automatically handle the task of maintaining the state of your form and its controls, and provides you with explicit ways to maintain the state of application-specific information.

How to Create a Web Forms Application



Introduction

The first step in creating a Web Form is to create an ASP.NET Web Application project.

Creating a Web Application project

To create an ASP.NET Web Application project:

1. On the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, perform the following steps:
   a. In the Project Types pane, click **Visual C# Projects**.
   b. In the Templates pane, click **ASP.NET Web Application**.
   c. In the **Location** box, enter the complete Uniform Resource Locator (URL) for your application, including http://, the name of the server, and the name of your project.
   d. Click **OK**.

   When you click **OK**, a new project is created at the root of the Web server that you specified. Also, a new Web Forms page named WebForm1.aspx is displayed, in Design view, in the Web Forms Designer.

Project files created

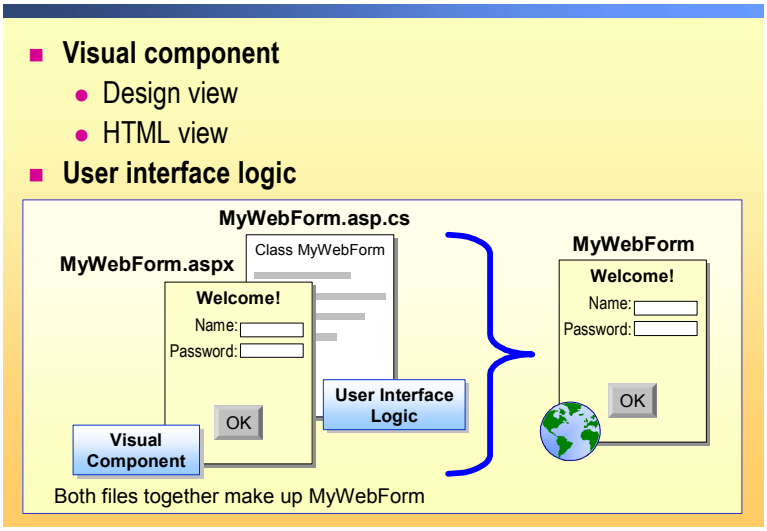
When you create an ASP.NET Web Application project by using Microsoft Visual C#™, Visual Studio .NET creates several files on your local computer. The following table lists and describes some of these files.

File Created	Description
WebForm1.aspx and WebForm1.aspx.cs files	These two files make up a single Web Forms page. The .aspx file contains the visual elements of the Web Forms page, for example the HTML elements and Web Forms controls. The WebForm1.aspx.cs class file is a dependent file of WebForm1.aspx. It contains the code-behind class for the Web Forms page, which contains event-handler code.
AssemblyInfo.cs	A project information file (AssemblyInfo.vb or AssemblyInfo.cs file) that contains metadata about the assemblies in a project, such as name, version, and culture information.
Web.config	An XML-based file that contains configuration data about each unique URL resource that is used in the project.
Global.asax and Global.asax.cs files	Global.asax is an optional file for handling application-level events. This file resides in the root directory of an ASP.NET application. The Global.asax.cs class file is a hidden, dependent file of Global.asax. It contains the code for handling application events, such as the Application_OnError event. At run time, this file is parsed and compiled.
.vsdisco (project discovery) file	An XML-based file that contains links (URLs) to resources providing discovery information for an XML Web service.

Viewing the files

To view all of the files in a project, click the **Show All Files** button in the toolbar of Solution Explorer.

What Are the Components of a Web Forms Application?



Introduction

Web Forms pages provide a distinction between the visual component, the visible portion of the form, and user interface logic, the code that interacts with the form.

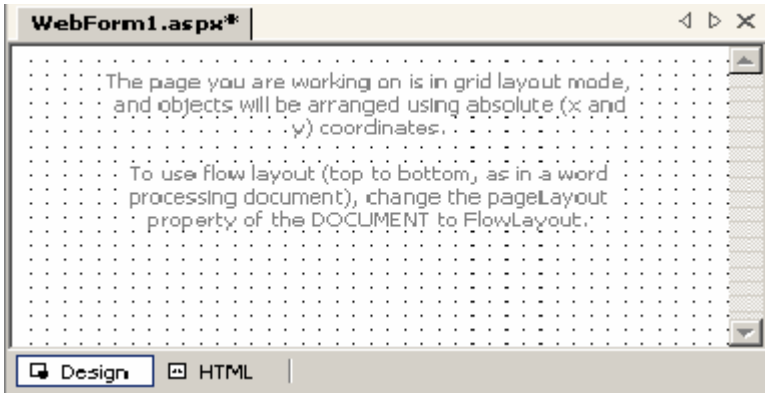
Visual component views

When you work with Web Forms, you must understand the two views that Visual Studio .NET provides. These views are the Design view and the HTML view.

You can work in either view. When you switch between them, each view is updated with the changes that you make in the other view.

Design view

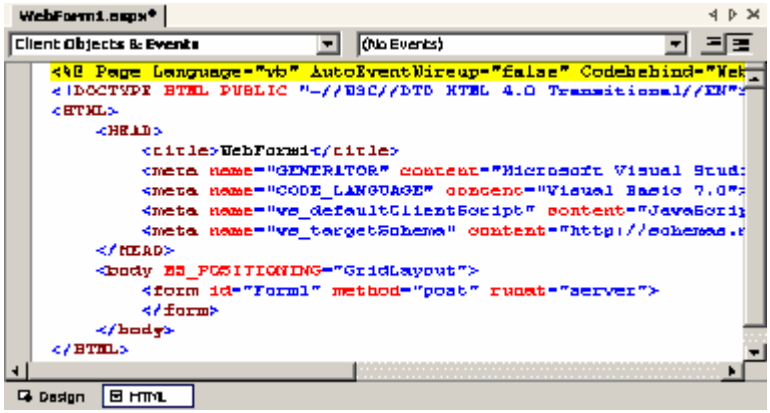
The **Design** tab is located at the bottom of the Web Forms Designer.



The Design view shows you the WYSIWYG view of the .aspx file that you are working with. In the Design view, you can drag controls from the Toolbox and use the Properties window to configure the controls.

HTML view

The **HTML** tab is located at the bottom of the Web Forms Designer.



The HTML view shows you the HTML format of the .aspx file that you are working with. As in other code editor views, the Web Forms Designer supports Microsoft IntelliSense® for elements in HTML view.

User interface logic

A Web Forms page code model consists of two files:

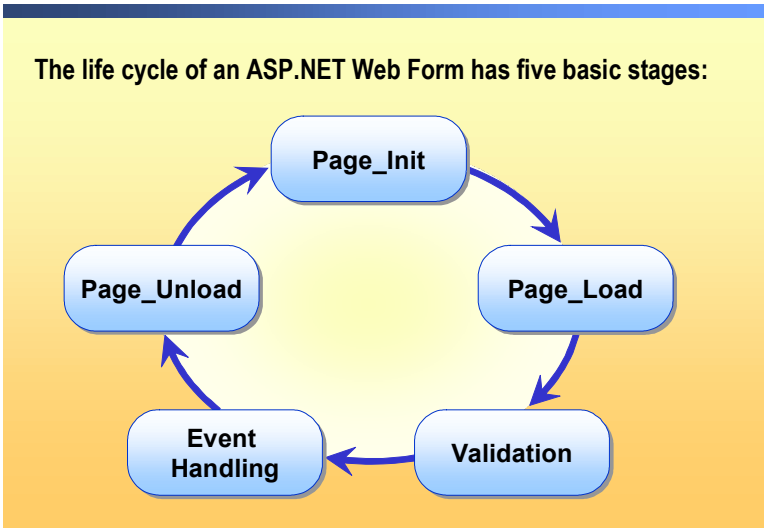
- WebForm.aspx

  The WebForm1.aspx file is referred to as the *page*. This file contains Hypertext Markup Language (HTML), static text, and the server controls that make up the visual components of the page. HTML is the computer language that is used to create documents for the Web. The page works as a container for the text and controls that you want to display.

- WebForm.aspx.cs

  This file, which is also referred to as the code-behind file, contains code that you create to interact with the form. The extension for this file is language specific. For example, the extension is vb if you use Visual Basic .NET and cs if you are using C#.

What Is the Life Cycle of a Web Forms Application?



Introduction

It is helpful to understand some fundamental characteristics of how Web Forms pages work in Web applications before you examine the details of what occurs inside a page when it is processed.

Round trips

It is important to understand the division of labor on a Web Forms page. The browser presents the user with a form, and the user interacts with the form, causing the form to post back to the server. However, because all processing that interacts with server components must occur on the server, for each action that requires processing, the form must be posted to the server, processed, and returned to the browser. This sequence of events is referred to as a *round trip*.

In Web Forms, most user actions—such as clicking a button—result in a round trip. For that reason, the events that are available in ASP.NET server controls are usually limited to click-type events.

Recreating the page

In any Web scenario, pages are re-created with every round trip. As soon as the server finishes processing and sending the page to the browser, it discards the page information. By freeing server resources after each request, a Web application can scale to support hundreds or thousands of simultaneous users. The next time the page is posted, the server starts over in creating and processing it, and for this reason, Web pages are said to be stateless.

The life cycle of an ASP.NET Web Form

The life cycle of an ASP.NET Web Form has five basic stages:

1. *Page_Init*. The ASP.NET page framework uses this event to restore control properties and postback data, which is data that the user entered in controls before the form was submitted.
2. *Page_Load*. The developer uses this event either to perform some initial processing, if this is the first visit to the page, or to restore control values, if this is a postback.
3. *Validation*. The **Validate** method of ASP.NET server controls is called to perform validation for the controls.
4. *Other event handling*. Various controls expose many events. For example, the **Calendar** control exposes a **SelectionChanged event**. If the page contains validation controls, you should check the **IsValid** property of the page and individual validation controls to determine whether validation has been passed.
5. *Page_Unload*. This event is called as the page finishes rendering.

Tip It is at this last stage where you clean up any resources that were allocated, especially expensive resources such as the file handlers and database connections.

How to Add Controls to a Web Forms Application

- **To add a Web server control**
  - In Design view, drag Web Server control object from the Toolbox Web Forms tab
- **To add an HTML server control**
  - Drag an HTML element onto the page from the HTML tab of the Toolbox
  - Right-click the element and choose Run As Server Control to convert it to a control

Introduction

After you create the Web Form, you can add controls to build the user interface. Visual Studio .NET provides Forms Designer, an editor, controls, and debugging tools, which together allow you to build programmable user interfaces for the Web.

Server controls

Controls for Web Forms are called server controls because when the page runs, the controls are instantiated in server code as part of the page class. When users interact with a control, the code that is associated with the control runs on the server after the page is posted. For example, when a user clicks a **Web Forms** button control, the code for the button runs on the server after the page is displayed. You can set properties and write event handlers in the server code.

There are two types of server controls:

- Web server controls

  These are controls specific to Web Forms that provide more features than HTML server controls and do not map directly to HTML elements.

- HTML server controls

  These are HTML elements that are marked to be programmable in server code. Typically, you convert HTML elements to HTML server controls only if you want to program them from server code.

Converting client controls to run as server controls

Not every element on the Web Forms page is a server control. For example, by default, static HTML text is not a server control, and you cannot control it from server code. Even standard HTML controls, such as an HTML button, are not server controls by default. You can program the HTML elements in the client code. Therefore, to work with controls on a Web Forms page, you must add them as server controls.

Adding an HTML server control by using Web Forms Designer

Adding an HTML server control to a Web Forms page is a two-step process.

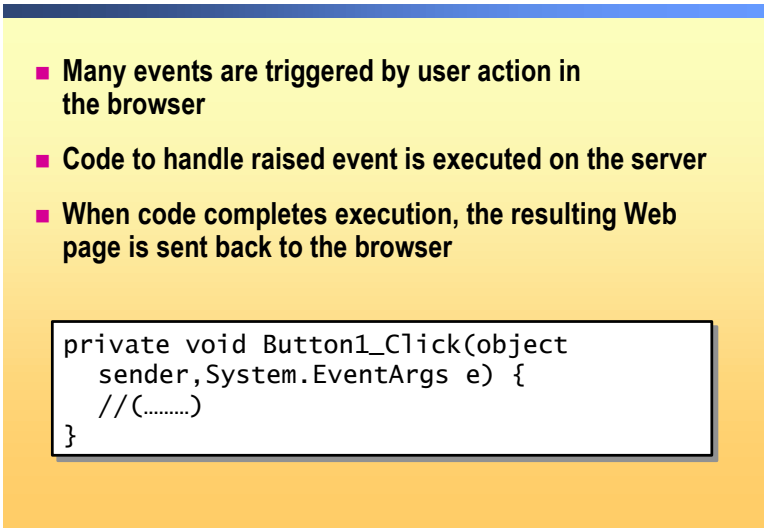
To add an HTML control to a Web Forms page and convert it to a server control:

1. From the **HTML** tab of the Toolbox, drag an HTML element onto the page.
2. Convert the element to a server control by right-clicking it, and then clicking **Run As Server Control**.

The Web Forms Designer adds the attribute **runat="server"** to the element, which alerts the server to treat the element as a server control. A symbol appears on the control in Design view to indicate that it is a server-based control.

By default, the Web Forms page uses Grid layout, and you place controls at absolute positions on the page by using x and y coordinates. If you want to use linear layout, in which the page elements flow in the same manner as in a word processing document, you can change the **pageLayout** property or include a **Flow Layout Panel** HTML server control.

How to Add an Event Handler for the Control



Introduction

After you create the project and add controls, you can add event handlers to the controls. Web Forms bring to Web applications the model of writing event-handling methods for events that occur in either the client or server. The Web Forms framework abstracts this model in such a way that the underlying mechanism of capturing an event on the client, transmitting it to the server, and calling the appropriate handler is automatic and invisible. The result is a clear, easily written code structure.

Adding events

Interacting with users is one of the primary reasons for creating ASP.NET Web Forms. You program various events to handle these interactions. The Web page itself can execute code, and so can the many events that are raised by various objects, including all of the server controls.

You can add events to individual controls, to a page, to an application, or to a session.

Controls have a default event, which is the event that is most commonly associated with that control. For example, the default event for a button is the **Click** event. You can create event handlers for both the default event and other events, but the procedure is different for each type of event.

Creating an event handler for a non-default event

To create an event handler for non-default events:

1. In Design view, select the control, and then press F4 to display the Properties window.
2. In the Properties window, click the **Events** button ( ).

   The Properties window displays a list of the events for the control, with boxes to the right that display the names of the event handlers that are bound to those events.
3. Locate the event that you want to create a handler for, and then, in the event name box, type the name of an event handler.

Creating the event handler for a default event

To create the event handler for a default event:

- In Design view of the Web Forms Designer, double-click the control or page. The Code Editor opens with the insertion point in the event handler.

Example

A **Button** Web server control can raise a **Click** event when a user clicks a button on the page.

The code to handle the raised event is executed on the server. When the user clicks a button, the page is posted back to the server. The framework for the ASP.NET page parses the event information, and if you have an event handler corresponding to the event, your code is called automatically. When your code finishes, the page is sent back to the browser with any changes that the event handler code made.

To create an event handler for the Button **Web** server control:

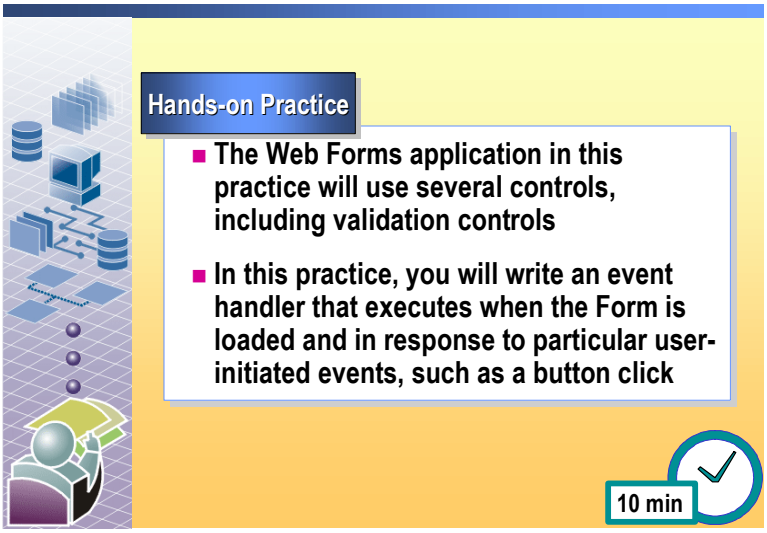
- Double-click the **Button** Web server control.

  The designer opens the class file for the current form and creates a skeleton event handler for the **Click** event of the button control. The code is as follows:

```
private void button1_Click(object sender, System.EventArgs
e) {
}
```

Note For more information about recommendations for ASP.NET server controls, see the Visual Studio. NET documentation. Use the Help index and search for ASP.NET Server Control.

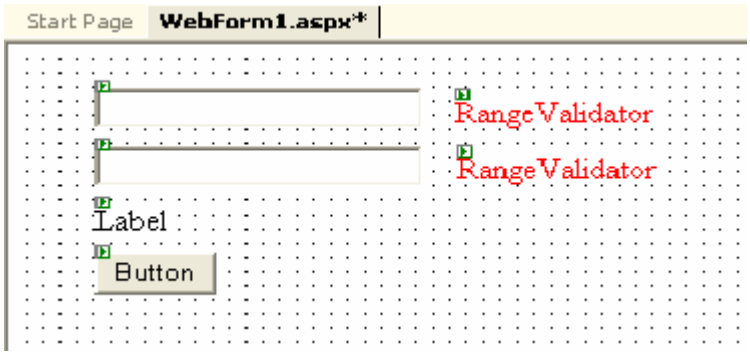
Practice: Creating a Web Forms Application



In this practice, you will develop a simple Web Forms application that calculates the sum of two values that are entered into the form. The form will validate that the first number entered is in the range of 1 to 1000 and that the second number entered is in the range of 1 to 500.

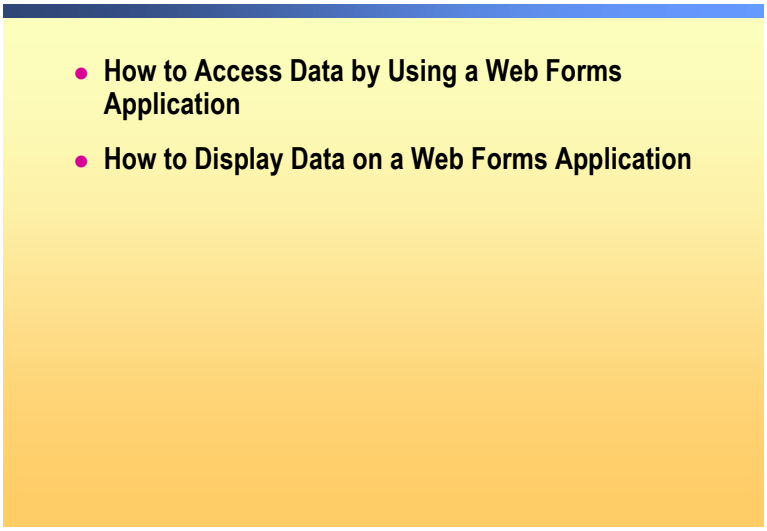
The solution for this practice is in *install_folder*\Practices\Mod10\WebFormPractice_Solution. To open the solution, follow the instructions in the Readme.txt file located in the folder.

Tasks	Detailed steps
1. Start Visual Studio .NET.	▪ Start a new instance of Visual Studio .NET.
2. Create a new project: Project Type: Visual C# Project Template: ASP.NET Web Application. Location: http://localhost/WebFormPractice	**a.** On the **Start** page, click **New Project**. **b.** In the **New Project** dialog box, under **Project Types**, click **Visual C# Projects**. **c.** Under **Templates**, click **ASP.NET Web Application**. **d.** In the **Location** box, type **http://localhost/WebFormPractice** and then click **OK**.
3. From the **Web Forms** tab of the Toolbox, add the following controls to the form: 2 TextBox controls 1 Label control 1 Button control	**a.** Point to the Toolbox, and then drag a **TextBox** control onto the form. **b.** Point to the Toolbox, and then drag another **TextBox** control onto the form. **c.** Point to the Toolbox, and then drag a **Label** control onto the form. **d.** Point to the Toolbox, and then drag a **Button** control onto the form.

Tasks	Detailed steps
4. (*Optional*) From the **Web Forms** tab of the Toolbox add the following controls to the form: 2 **RangeValidator** controls	**a.** Point to the Toolbox, and then drag a **RangeValidator** control onto the form. **b.** Point to the Toolbox, and then drag another **RangeValidator** control onto the form.
5. Lay out the controls on the form to match the following illustration.	▪ Lay out the controls on the form to match the following illustration.
Note: Lay out your form to match the illustration, **RangeValidator** controls are optional. 	
6. Set the Text property of the button to **Calculate**.	**a.** In the WebForm1.aspx form editor, click the button. **b.** In the Properties window, click **Text**, and then type **Calculate**
7. (*Optional*) Change the following properties of the **RangeValidator** located at the top of the form. Errormessage: **The number entered must be in the range 1-1000**. ControlToValidate: **TextBox1** MaximumValue: **1000** MinimumValue: **1** Type: **Integer**	**a.** In the WebForm1.aspx form editor, click the **RangeValidator** control located at the top of the form. **b.** In the Properties window, click **Errormessage**, and then type **The number entered must be in the range 1-1000** **c.** In the **ControlToValidate** list, click **TextBox1**. **d.** Click **MaximumValue**, and then type **1000** **e.** Click **MinimumValue**, and then type **1** **f.** In the **Type** list, click **Integer**.
8. (*Optional*) Change the following properties of the **RangeValidator** located beneath the other **RangeValidator**. Errormessage: **The number entered must be in the range 1-500**. ControlToValidate: **TextBox2** MaximumValue: **500** MinimumValue: **1** Type: **Integer**	**a.** In the WebForm1.aspx form editor window, click the **RangeValidator** control located beneath the other **RangeValidator** control. **b.** In the Properties window, click **Errormessage**, and then type **The number entered must be in the range 1-500** **c.** In the **ControlToValidate** list, click **TextBox2**. **d.** Click **MaximumValue**, and then enter **500** **e.** Click **MinimumValue**, and then enter **1** **f.** In the **Type** list, click **Integer**.

Tasks	Detailed steps
9. Add code into the Button click event to calculate the sum of the two text boxes and place the result into the label.	**a.** Double-click **Calculate** on the form to open WebForm1.aspx.cs in the Code Editor. The cursor is placed in the **Button1_Click** event. **b.** Type the following code: `Label1.Text = (System.Convert.ToInt32(TextBox1.Text) + System.Convert.ToInt32(TextBox2.Text)) .ToString();`
10. Run the application, enter various values into the textboxes displayed and examine the behavior of the application.	**a.** On the standard toolbar, click **Start**. **b.** Enter various values into the text boxes that are displayed, and examine the behavior of the application.
11. Quit Microsoft Internet Explorer.	▪ Quit Internet Explorer.
12. Save any changes to the project, and then quit Visual Studio .NET.	**a.** On the **File** menu, click **Save All**. **b.** On the **File** menu, click **Exit**.

Lesson: Accessing Data by Using a Web Forms Application



Introduction

This lesson explains how use Microsoft ADO.NET to access data by using a Web Forms application.

Lesson objectives

After completing this lesson, you will be able to:

- Use ADO.NET from a Web Forms application.
- Display data on a Web Forms application.

Lesson agenda

This lesson includes the following topics:

- How to Access Data by Using a Web Forms Application
- How to Display Data on a Web Forms Application
- Practice: Displaying Data from a Database on a Web Forms Application

How to Access Data by Using a Web Forms Application

- **Fundamental principles**
  - Using a disconnected model
  - Reading data more often than updating it
  - Minimizing server resource requirements
  - Accessing data using remote processes
- **Data sources for Web Forms pages**
  - Database access, ADO.NET
  - XML data
  - Other sources

Introduction

Many Web Forms pages involve data access, displaying data and, in some cases, allowing users to edit and update data. Knowledge of data access technology in Web Forms pages helps you create efficient Web applications.

Principles

Data access in Web Forms pages is built around the following fundamental principles:

- Using a disconnected model

  Web Forms pages are disconnected. Each time a Web Forms page is requested, it is built, processed, sent to the browser, and discarded from server memory. By extension, the same process applies to data access in a Web Forms page. Data is read or updated while the page is processed on the server. After the page is processed and sent to the browser, data is discarded along with other page elements.

- Reading data more often than updating it

  The Web Forms data model presumes that most data access by Web pages is read-only. Typical examples are catalog or search listings that display data items. In most cases, the user does not enter data that is written back to the data source.

- Minimizing server resource requirements

  Data access in Web Forms pages therefore requires careful attention to how you use resources.

- Accessing data by using remote processes

  Web Forms pages are the presentation tier of your Web application. You can build data access into your pages, but it is also common to separate data access logic from the user interface by building it into another component, such as an XML Web service, that interacts with the data source.

Data sources for Web Forms

The Web Forms page architecture provides a very flexible notion of data. This includes everything from traditional database access, to using XML documents as a data source, to generating data at run time and storing it in an array:

- Database access

  To read and write database data, you use ADO.NET. ADO.NET includes managed data providers (connection and command objects) to communicate with Microsoft SQL Server™ or OLE DB–compatible databases. ADO.NET also includes support for disconnected data access by means of a dataset, which is an in-memory cache into which you can read records to work with.

  Alternatively, you can use ADO.NET objects to execute SQL commands or stored procedures directly. This allows you to read data straight from the database and send updates back.

- XML data

  Another possible source of data in a Web Forms page is an XML document or stream. You can work with XML data in two ways:

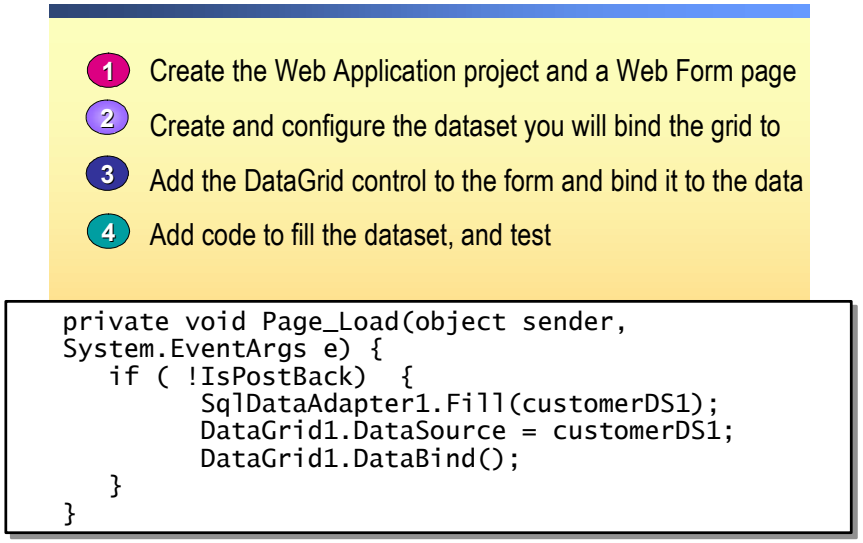
  - If the XML data is structured—that is, if it can be represented as relational data—you can convert the XML data into a dataset and use ADO.NET data functions to read and update the data. This feature allows you to take advantage of the comparatively sophisticated and simple data-processing functionality of datasets. You can then convert the data back to XML to share with other processes.
  - If the XML data cannot be represented as relational data, you can use XML parsing and processing functions from the **System.Xml** namespace to manipulate the data. In Web Forms pages, you can do this by using the XML Web server control. Alternatively, you can work directly with XML documents in code.

- Other data sources

  Web Forms pages allow you to work with virtually any other type of data also.

  The data-binding architecture of Web Forms pages allows you to bind a control to any structure. In practice, this means you can bind to any arrays or collections that are available in the page, as well as to properties of the page or of other controls.

How to Display Data on a Web Forms Application



Introduction

Web Forms pages often must display information that is derived from a database, an XML document or stream, or some other data source. The architecture of Web Forms pages provides you with ways to incorporate data sources, or references to them, in the page, to bind controls to data, and to manipulate data in various ways.

Note Displaying data in Web Forms is similar to displaying data in Windows Forms.

Displaying data

To display data on a Web Form:

1. Create the Web application project and a Web Forms page.
2. Create and configure the dataset that you will bind the grid to. This includes creating a query that populates the dataset from the database.
3. Add the **DataGrid** control to the form and bind it to data.
4. Add code to fill the dataset, and test.

This procedure is described in detail in the following steps.

Step 1

To create a Web application project and a Web Forms page:

1. On the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, do the following:
   a. In the Project Types pane, click **Visual C# Projects**.
   b. In the Templates pane, click **ASP.NET Web Application**.
   c. In the **Location** box, enter the complete URL for your application, including **http://**, the name of the server, and the name of your project.
   d. Click **OK**. A new Web Forms project is created at the root of the Web server that you specified. Also, a new Web Forms page named WebForm1.aspx is displayed in the Web Forms Designer in Design view.

Step 2

The second step is to create the data connection and data adapter.

1. From the **Data** tab of the Toolbox, drag a **SqlDataAdapter** object onto the form. The Data Adapter Configuration Wizard starts which helps you create both the connection and the adapter. In the wizard, do the following:
2. Click **Next**. On **Choose Your Data Connection**, create or select a connection pointing to the **SQL Server Northwind** database.
3. Click **Next**. On **Choose a Query Type**, specify that you want to use a SQL statement to access the database.
4. On **Generate the SQL Statements**, create the SQL statements, or to build the SQL statement click **Query Builder** to launch the **Query Builder** dialog box.
5. Click **Finish**. The wizard creates a connection, **sqlConnection1**, containing information about how to access your database. The wizard also creates a data adapter, **sqlDataAdapter1**, that contains a query specifying the table and columns in the database that you want to access.

Step 3

The next step is to create and configure the dataset.

After you establish the connection to the database and specify the information you want to access by means of the SQL command in the data adapter, you can have Visual Studio .NET create a dataset. The dataset is an instance of the **DataSet** class based on a corresponding schema, .xsd file, that describes the elements of the class, such as the table, columns, and constraints. Visual Studio .NET can generate the dataset automatically based on the query that you specified for the data adapter.

1. On the **Data** menu, click **Generate DataSet**.

> **Tip** If you do not see the **Data** menu, make sure that the focus is on the form, and then click the form. The **Generate Dataset** dialog box appears.

2. Click **New** and name the dataset (CustomerDS in this example). In the list under **Choose which table(s) to add to the dataset**, make sure that the table you want to display is selected.
3. Select the **Add this dataset to the designer** check box, and then click **OK**. Visual Studio .NET generates a typed dataset class and a schema that defines the dataset. The new schema, **CustomerDS.xsd**, is displayed in Solution Explorer.

At this point, you have set up everything you need in order to get information out of a database and into a dataset.

Step 4

Next, add a DataGrid to display data.

1. From the **Web Forms** tab of the Toolbox, drag a **DataGrid** control onto the page.
2. In the **DataSource** property, select **customerDS1** (this is the instance of the CustomerDS class) as the data source. This binds the grid to the dataset as a whole.
3. In the **DataMember** property, select the table specified in your SQL query from Step 2. If a data source contains more than one bindable object, you can use the **DataMember** property to specify which object to bind to. Setting these two properties binds the specified data table in the **customerDS1** dataset to the grid.

Step 5

The next step is to fill the dataset and display data in the **DataGrid** control.

Although the grid is bound to the dataset that you created, the dataset itself is not automatically filled in. Instead, you must fill the dataset yourself by calling a data-adapter method.

1. Double-click the page to display the class file of the page in the Code Editor.
2. In the **Page_Load** event handler, call the **Fill** method of the adapter, passing it the dataset that you want to populate:

   ```
   sqlDataAdapter1.Fill(customerDS1);
   ```

3. Call the **DataBind** method of the **DataGrid** control (DataGrid1) to bind the control to the dataset.

Tip You do not need to refill the dataset and bind the grid with each round trip. After the **DataGrid** control is populated with data, its values are preserved in view state each time the page is posted. Therefore, you must fill the dataset and bind the grid only the first time that the page is called. You can test for this by using the **IsPostBack** property of the page.

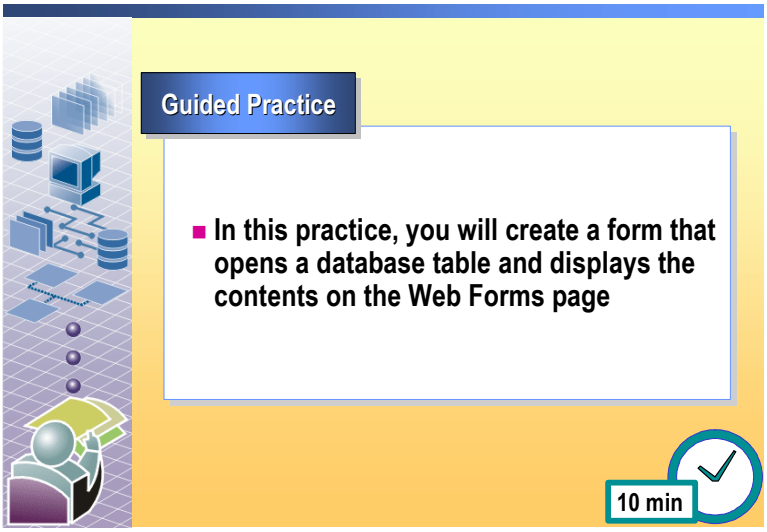
The following code shows the complete handler:

```
private void Page_Load(object sender, System.EventArgs e) {
// Put user code to initialize the page here
if ( !IsPostBack ) {
      sqlDataAdapter1.Fill(customerDS1);
      DataGrid1.DataBind( );
      }
}
```

The final step is to test the Web Forms page.

1. Save the page.
2. In Solution Explorer, right-click the page, and then click **Build and Browse**.
3. Confirm that a list of categories is displayed in the grid.

Practice: Displaying Data from a Database on a Web Forms Application



In this practice, you will develop a simple Web Forms application that contains a **DataGrid** control. The **DataGrid** control will be used to display the contents of the SQL Server table **BankCustomers**.

The solution for this practice is in *install_folder*\Practices\Mod10\DataGridPractice_Solution. To open the solution, follow the instructions in the Readme.txt file located in the folder.

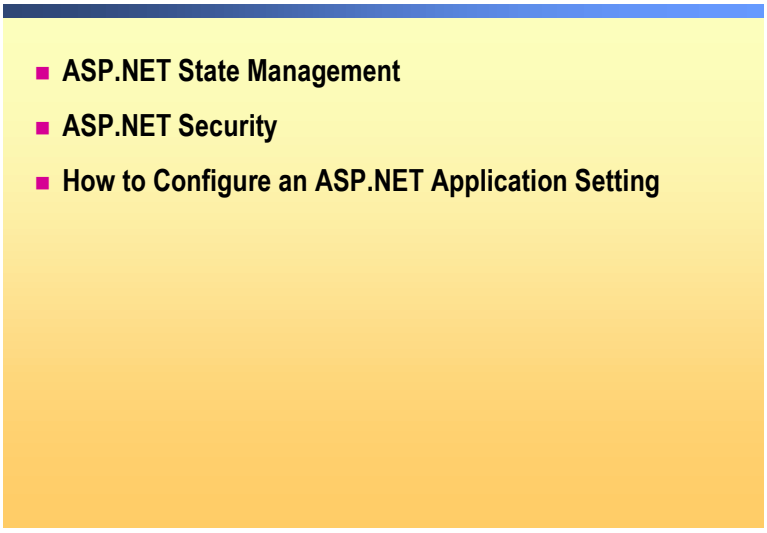
Tasks	Detailed steps
1. Start Visual Studio .NET.	▪ Start a new instance of Visual Studio .NET.
2. Create a new project: Project Type: Visual C# Project Template: ASP.NET Web Application. Location: http://localhost/DataGridPractice	a. On the **Start** page, click **New Project**. b. In the **New Project** dialog box, under **Project Types**, click **Visual C# Projects**. c. Under **Templates**, click **ASP.NET Web Application**. d. In the **Location** box, type **http://localhost/DataGridPractice** and then click **OK**.
3. Drop a **DataGrid** control onto the form.	▪ Point to the Toolbox, and then drag a **DataGrid** control onto the form.
4. Position the **DataGrid** control in the upper-left corner of the form.	▪ Drag the **DataGrid** control to the upper-left of the form.
5. Drop a **SqlDataAdapter** control onto the form. This control is held under the **Data** tab of the Toolbox.	▪ Point to the Toolbox, click the **Data** tab, and then drag a **SqlDataAdapter** control onto the form.

Tasks	Detailed steps
6. Step through the Data Adapter Configuration Wizard. Create a new connection to your local server; use Microsoft Windows NT® Integrated Security and database 2609. Use SQL statements for the query type. Enter a SQL statement of **Select * from BankCustomers**.	**a.** In the Data Adapter Configuration Wizard, on the **Welcome to the Data Adapter Configuration Wizard** page, click **Next**. **b.** On the **Choose Your Data Connection** page, click **New Connection**. **c.** In the **Data Link Properties** dialog box, in the **Select or enter a server name** box, type *your_server_name* **d.** Under **Enter information to log on to the server**, click **Use Windows NT Integrated security**. **e.** Under **Select the database on the server**, click **2609**, and then click **OK**. **f.** On the **Choose Your Data Connection** page, click **Next**. **g.** On the **Choose A Query Type** page, click **Use SQL Statements**, and then click **Next**. **h.** Under **What data should the data adapter load into the dataset**, type **SELECT * from BankCustomers** and then click **Next**. **i.** On the **View Wizard Results** page, click **Finish**.
Note: Your form now has two hidden controls. A **sqlDataAdapter1** control and a **sqlConnection1** object. The connection object is generated automatically by the wizard.	
7. Use the **Generate Dataset** option on the **Data** menu to create a dataset named **Customers**. Add the **BankCustomers** table to this dataset.	**a.** On the **Data** menu, click **Generate Dataset**. **b.** In the **Generate Dataset** dialog box, under **Choose a dataset**, click **New**, replace DataSet1 with **Customers** and then click **OK**.
8. Set the **DataSource** property of the **DataGrid** control to **customers1**.	**a.** In the **WebForm1.aspx** form editor, click the **DataGrid** control. **b.** In the Properties window, in the **DataSource** list, click **customers1**.
9. Use the **Auto Format** URL at the bottom of the Properties window to set the format to **Colorful 2**.	**a.** In the Properties window, click the **Auto Format** link. **b.** In the **Auto Format** dialog box, under **Select a scheme**, click **Colorful 2**, and then click **OK**.
10. Add the following code to the **Form_Load** event of the form.	**a.** Double-click an area of the form that is not covered by the **DataGrid** control. **b.** Add the following code to the **Page_Load** event procedure.

```
if (!IsPostBack) {
  sqlDataAdapter1.Fill(customers1);
  DataGrid1.DataBind();
}
```

Tasks	Detailed steps
11. Run the application and view the contents of the data grid.	▪ On the standard toolbar, click **Start**.
Note: The **DataGrid** control contains all the columns of data held in the dataset. The following steps show how the columns of the dataset can be modified.	
12. Close the browser window.	▪ Close the browser window.
13. Change the **AutoGenerateColumns** property to **False**.	**a.** In the editor, click the **WebForm1.aspx** tab. **b.** Click the **DataGrid** control on the form. **c.** In the Properties window, change AutoGenerateColumns from **True** to **False**.
14. Use the Property Builder feature of the **DataGrid** control to add the following columns to the DataGrid. **CustomerID** **CustomerName** **CustomerAddress** **CustomerPhone**	**a.** Right-click the **DataGrid** control, and then click **Property Builder**. **b.** In the DataGrid1 Properties window, click **Columns**. **c.** Under **Available Columns**, click **CustomerID**, and then click **>**. **d.** Under **Available Columns**, click **CustomerName**, and then click **>**. **e.** Under **Available Columns**, click **CustomerAddress**, and then click **>**. **f.** Under **Available Columns**, click **CustomerPhone**, click **>** and then click **OK**.
15. Run the application and view the contents of the DataGrid.	▪ On the standard toolbar, click **Start**.
Note: The DataGrid now contains only the required columns.	
16. Close the browser window.	▪ Close the browser window.
17. Save the changes to the project, and then quit Visual Studio .NET.	**a.** On the **File** menu, click **Save All**. **b.** On the **File** menu, click **Exit**.

Lesson: Configuring ASP.NET Application Settings



Introduction

This lesson introduces ASP.NET technology, as it relates to developing Web applications.

Lesson objectives

After completing this lesson, you will be able to:

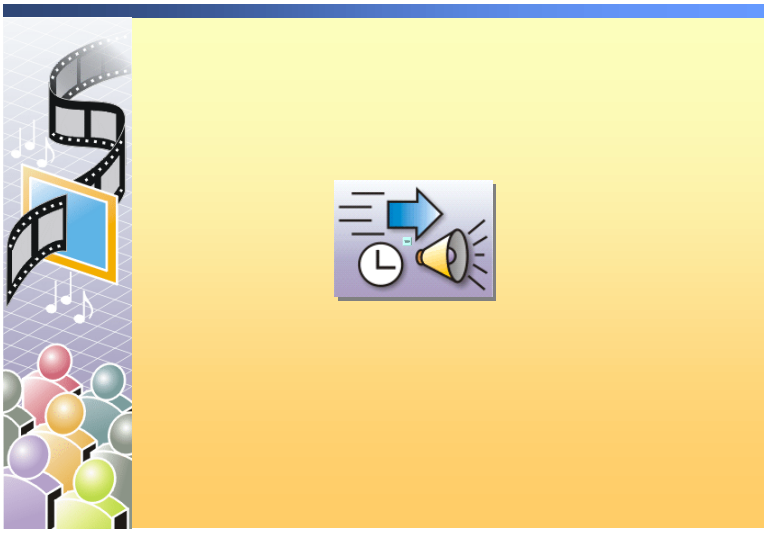
- Explain ASP.NET state management.
- Explain ASP.NET security.
- Configure an ASP.NET application setting.

Lesson agenda

This lesson includes the following topics and activities:

- Multimedia: ASP.NET Execution Model
- ASP.NET State Management
- ASP.NET Security
- How to Configure an ASP.NET Application Setting
- Practice: Configuring a Web Application Using Web.Config

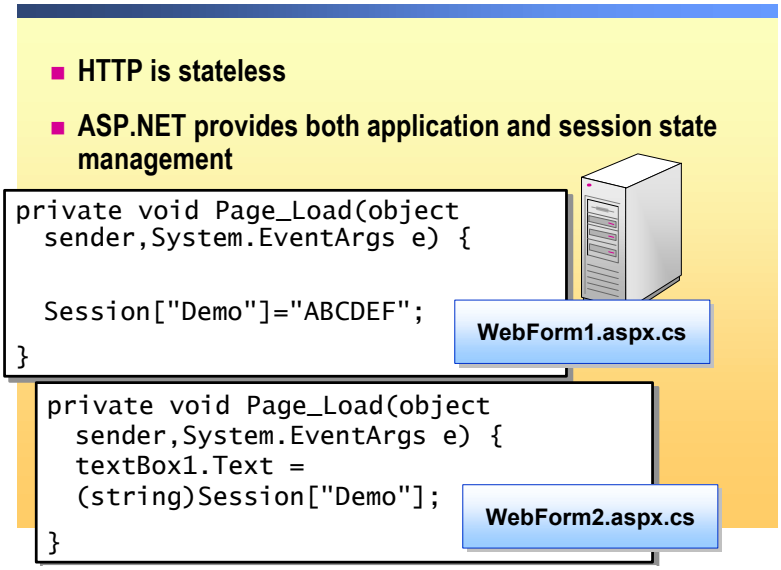
Multimedia: ASP.NET Execution Model



Introduction

In this animation, you will see how ASP.NET works to send information to a requesting client.

ASP.NET State Management



Introduction

HTTP is a stateless protocol. A browser requests a page from a Web server; the Web server receives that request, fetches the page from the hard disk, and then returns that page to the browser.

However, if the browser requests another page from the same Web server, the Web server has no knowledge of the fact that the browser recently requested a page. In the early days of the Internet, when much of the available content was static, this process was not a problem.

The fact that the Internet is currently being used for more than just the casual browsing of static Web pages presents the Web application developer with a problem. How can the developer maintain information between requests to different pages while the user browses the developer's Web application? This problem is solved by ASP.NET state management.

ASP.NET state management

ASP.NET improves on the state management mechanism provided by the earlier versions of ASP.NET, Active Server Pages. ASP.NET provides two objects: **Application** and **Session**. You use the **Application** object to hold information that is common to all users of a particular Web application. You use the **Session** object to store information on a user-by-user basis.

Example

As shown in the following code, the string ABCDEF is placed into the **Session** object for storage between pages. As you keep track of variables by using different names, this string is held in the **Session** object against the name Demo. When a user accesses this page, ASP.NET creates a **Session** object for the user and stores the string against the name Demo. When the user accesses the second page in the example, ASP.NET retrieves the object from the **Session** object by the name Demo.

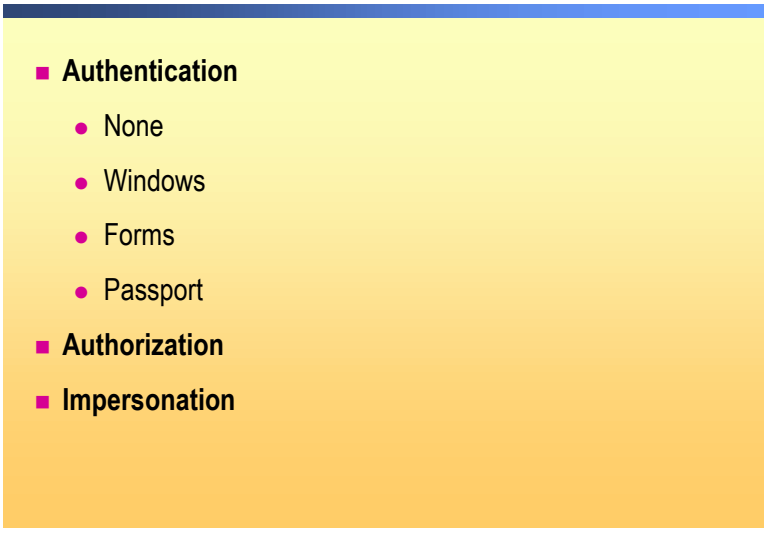
```
private void Page_Load(object sender,System.EventArgs e){
  Session["Demo"]="ABCDEF";
}
```

Notice the cast from Session["Demo"], which is type **object**, to string.

```
private void Page_Load(object sender,System.EventArgs e) {
  textBox1.Text = (string)Session["Demo"];
}
```

Note In ASP.NET, the previous implementation of state management was enhanced to allow it to scale better for large installations and to be more reliable.

ASP.NET Security



Introduction

Developing dynamic and interactive Web sites typically involves some element of security.

Authentication

Allowing users of a Web site to save their preferred airlines for future use requires an ability to validate the user who is accessing the Web site so that the correct information can be retrieved. This type of security is called *authentication*.

Authorization

You may want to restrict some pages of your Web site to particular users. After your Web application authenticates the user, access to specific pages can be allowed or denied. This type of security is called *authorization*.

Impersonation

Finally, you may want your Web application to interact with other applications, such as a database. You also may want it to appear to other applications that it is being accessed by the user of the Web site, or possibly by a different, fixed user account. This type of security is called *impersonation*.

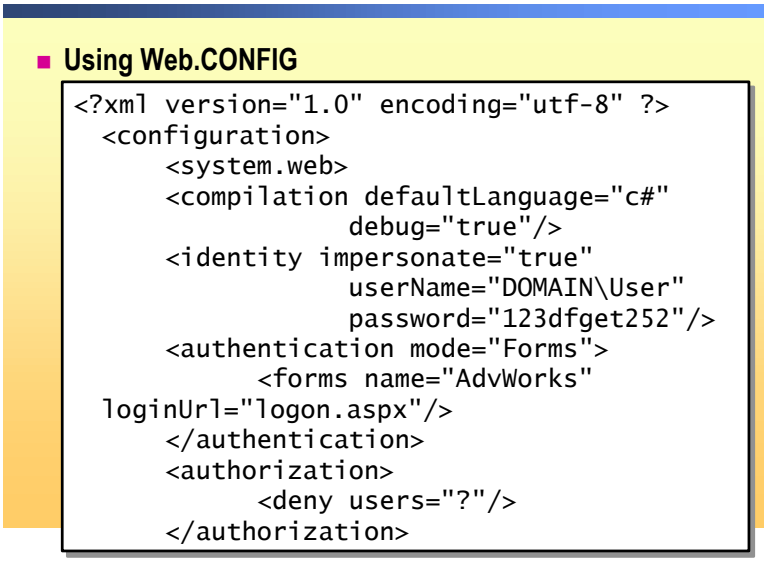
To enable an authentication provider

To enable an authentication provider for an ASP.NET application, you need only create an entry for the application configuration file as shown in the following code:

```
// web.config file
<authentication mode= "[Windows/Forms/Passport/None]">
</authentication>

ASP.Net Authorization
<authentication mode="Forms">
  <forms name="Test Application Logon Page"
loginURL="logonform.aspx" />
</authentication>
<authorization>
  <deny users="?">
</authorization>
```

How to Configure an ASP.NET Application Setting



Introduction

The configuration file for an ASP.NET application is contained in the file Web.config. Configuration files in ASP.NET applications inherit the settings of configuration files in the URL path.

For example, given the URL www.microsoft.com/aaa/bbb, where www.microsoft.com/aaa is the Web application, the configuration file that is associated with the application is located at www.microsoft.com/aaa. ASP.NET pages that are in the subdirectory /bbb use both the settings that are in the configuration file at the application level and the settings in the configuration file that is in /bbb.

Hierarchical configuration architecture

ASP.NET applies configuration settings to resources in a hierarchical manner. Web.config files supply configuration information to the directories in which they are located and to all child directories. The configuration settings for a Web resource are supplied by the configuration file that is located in the same directory as the resource and by all configuration files in all parent directories.

Using Visual Studio .NET to change Web.Config

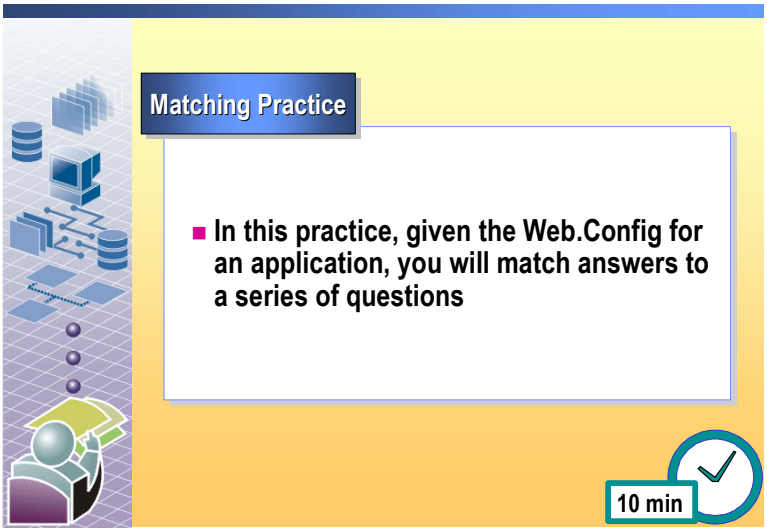
Web.Config should be listed as a file in your project in Solution Explorer. Double-click **Web.Config** to open an editing window to make changes to the file.

Example

In the following example, the ASP.NET application impersonates the identity DOMAIN/User. Authentication for the application uses the **Forms** method and specifically the Web page **logon.aspx** to validate logon requests. Users must be authenticated as anonymous users ("?") or will otherwise be denied access.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
  <compilation defaultLanguage="c#"
         debug="true"/>
  <identity impersonate="true"
         userName="DOMAIN\User"
         password="123dfget252"/>
  <authentication mode="Forms">
      <forms name="AdvWorks" loginUrl="logon.aspx"/>
  </authentication>
  <authorization>
      <deny users="?"/>
  </authorization>
….
```

Practice: Configuring a Web Application Using Web.Config



In this practice, you will match answers to a series of questions about the following Web.Config file for an application:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
      <compilation defaultLanguage="c#" debug="true" />
      <!--  AUTHENTICATION
           This section sets the authentication policies of the
application. Possible modes are "Windows", "Forms",
           "Passport" and "None"
          -->
      <authentication mode="windows" />
      <authorization>
          <deny users="?" />
      </authorization>
      <trace enabled="false" requestLimit="10"
pageOutput="false" traceMode="SortByTime" localOnly="true" />
      <sessionState mode="InProc"
stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1;user
id=sa;password=" cookieless="true" timeout="20" />
      <!--  GLOBALIZATION
           This section sets the globalization settings of the
application.
     -->
      <globalization requestEncoding="utf-8"
responseEncoding="utf-8" />
  </system.web>
</configuration>
```

With reference to the preceding Web.Config file, answer the following questions:

1. What kind of authentication mechanism is specified by this Web.Config file?

   Windows

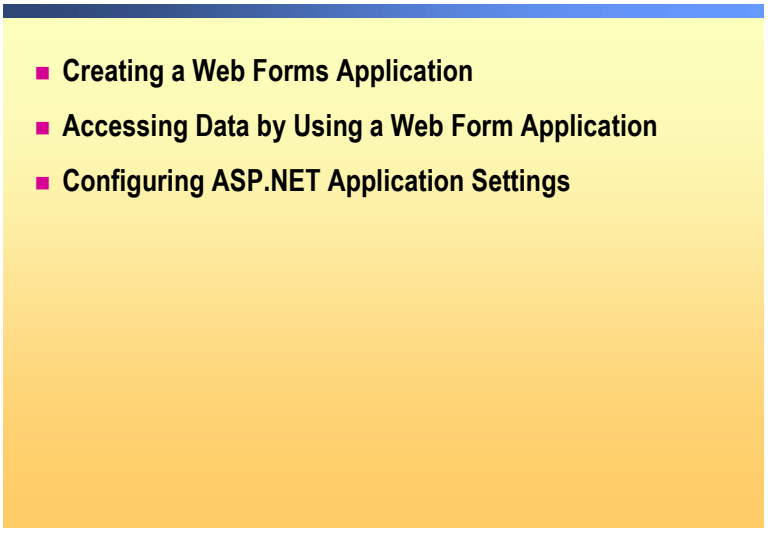
2. Are un-authenticated users allowed access to this Web application?

   No. This is specified by the <deny users="?" />.

3. What is the default language of this Web application?

   C#

Review



1. How can you view all of the files that Visual Studio .NET creates for a new ASP.NET Web application?

   Click Show All Files in the toolbar of Solution Explorer.

2. What are the extensions for the two files that make up a Web Forms page?

   The WebForm1.aspx file and the WebForm1.aspx.cs file.

3. What are the three components, not visible on a Web page, that are required to display the contents of a SQL Server table in a **DataGrid** control?

   SqlConnection object, SqlDataAdapter object, and DataSet object.

4. Name the five basic stages of a Web Forms page life cycle.

   Page_Init, Page_Load, Validation, other event handling, and Page_Unload.

5. Name the five steps that are necessary to display data on a Web Forms page.

   Create a Web application and Web Forms page, create the data connection and data adapter, create the DataSet, add a DataGrid to display data, and fill the DataSet and display data in the DataGrid control.

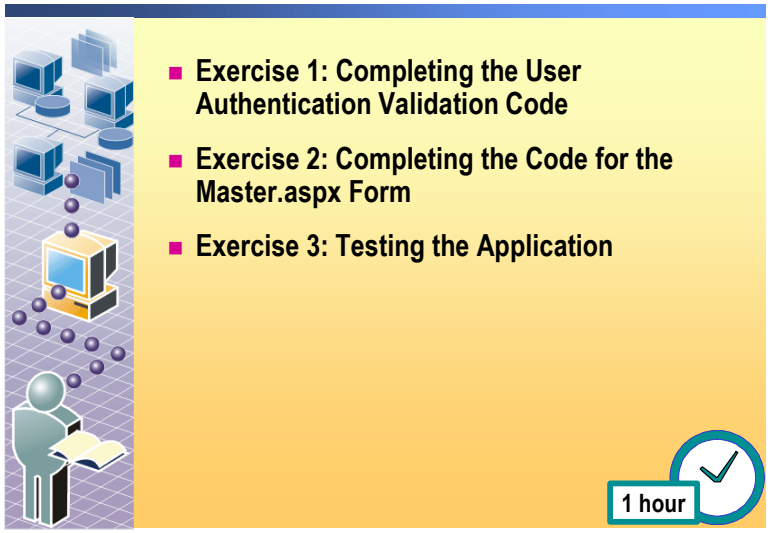
6. What browsers are supported by ASP.NET?

   All browsers.

7. Can you have more than one Web.Config applications settings?

   Yes, in different cascading folders.

Lab 10.1: Developing an ASP.NET Web Application



Objectives

After completing this lab, you will be able to:

- Develop an ASP.NET Web application.
- Use Forms based authentication.

Prerequisites

Before working on this lab, you must have the ability to:

- Create a Web Form.
- Handle events on a Web Form.
- Access data from Web Form applications.
- Configure ASP.NET application settings.

Scenario

In this lab, you will complete the code development of a basic Web interface to the AdventureWorks bank. This Web application allows users to view their accounts, display the transactions against an account, change their logon password, transfer money between accounts, and request traveler's checks in various currencies.

This lab consists of two exercises. In the first exercise, you will write the code to validate the user's logon. In the second exercise, you will complete the code in the event procedures for the main form, master.aspx. The code for the main form is provided. However, you must examine each code block, and then copy and paste it into the appropriate event procedure because this information is not given to you.

The solution for this lab is provided in install_folder\Labfiles\Lab10_1\AdvWorksBank_Solution. To open the solution, follow the instructions in the Readme.txt file located in the folder.

Estimated time to complete this lab: 60 minutes

Exercise 0
Lab Setup

The Lab Setup section lists the tasks that you must perform before you begin the lab. The ASP.NET application calls a local XML Web service. This Web service must be installed before you complete the lab.

Tasks	Detailed steps
1. Log on to Windows as **Student** with a password of **P@ssw0rd**.	▪ Log on to Windows with the following account: • User name: **Student** • Password: **P@ssw0rd**.
2. Install the Travelers Checks Web Service using the Setup.exe located in *install_folder*\Labfiles\Lab10_1\Web Service.	**a.** Click **Start**, and then click **Run**. **b.** In the **Run** dialog box, in the **Open** box, type *install_folder***Labfiles\Lab10_1\WebService\Setup.exe** and then click **OK**. **c.** In the **Travelers Checks Web Service** setup wizard, on the **Welcome to the Travelers Checks Web Service Setup Wizard** page, click **Next**. **d.** On the **Select Installation Address** page, click **Next**. **e.** On the **Confirm Installation** page click **Next**. **f.** On the **Installation Complete** page, click **Close**.

Exercise 1
Completing the User Authentication Validation Code

Because this ASP.NET Web Application uses Forms-based authentication, the first exercise involves writing the code that authenticates the user's logon attempt.

When the user attempts to access the Web site, ASP.NET will test the user for the presence of an encrypted session cookie. If the session cookie is not present on the client computer, ASP.NET automatically redirects the user to the logonform.aspx page. This redirection is accomplished by the authentication element that is contained in the Web.Config file. The logonform.aspx page contains two text boxes: a label to display any error messages, and a button. Users enter their customer number in one text box and then enter their password in the second text box. You must add code to the button click event that tests whether the password entered on the form matches the password stored in the database. If the passwords match, the user is issued a session cookie and redirected to the main page of the application.

Tasks	Detailed steps
1. Copy the folder *install_folder*\Labfiles\Lab10_1\AdvWorksBank to C:\Inetpub\wwwroot.	**a.** Click **Start**, point to **All Programs**, point to **Accessories**, and then click **Windows Explorer**. **b.** Click *install_folder***\labfiles\Lab10_1,** in the list of items in this folder right-click **AdvWorksBank**, and then click **Copy**. **c.** In the left pane of Windows Explorer, right-click **C:\Inetpub\wwwroot**, and then click **Paste**. **d.** Close the **Windows Explorer** window.
2. Using the Microsoft Internet Information Services application, change the virtual folder **AdvWorksBank** from the default application to standalone application named **AdvWorksBank**. Execute Permissions: **Scripts only**. Application Protection: **Medium (Pooled)**.	**a.** Click **Start**, and then click **Control Panel**. **b.** In Control Panel, double-click **Performance and Maintenance**, and then double-click **Administrative Tools**. **c.** In the **Administrative Tools** dialog box, double-click **Internet Information Services**. **d.** Expand *your_servername*. **e.** Expand **Web Sites**. **f.** Click **Default Web Site** **g.** In the content list of the **Default Web Site**, right-click **AdvWorksBank**, and then click **Properties**. **h.** In the AdvWorksBank Properties window, under **Application Settings**, click **Create** (this gives the Web site the default Execute Permissions of Scripts only and the Application Protection of Medium), and then click **OK**. **i.** Close **Internet Information Services**. **j.** Close **Administrative Tools**. **k.** Close **Control Panel**.

Tasks	Detailed steps
3. Open the AdvWorksBank solution in the folder C:\Inetpub\wwwroot\AdvWorksBank	**a.** Start Visual Studio .NET. **b.** On the **File** menu, click **Open Solution**. **c.** In the **Open Solution** dialog box, select the AdvWorksBank folder in the **Look In** folder (under C:\Inetpub\wwwroot). **d.** If it is not already selected, click **AdvWorksBank.sln**, and then click **Open**.
4. View the code for the file logonform.aspx.	▪ In Solution Explorer, right-click **logonform.aspx**, and then click **View Code**.
5. Scroll down and locate the Button1_Click procedure. Add code to this procedure as directed by the comments included.	▪ Scroll down and locate the Button1_Click procedure. Using the comments included in the procedure, write the code necessary to authenticate the user logon.

Exercise 2
Completing the Code for the Master.aspx Form

In this exercise, you will complete the code for the main form, master.aspx. Five event procedures in the form have no code. You are provided with five (5) code blocks in a text file called (Code Blocks.txt). These code blocks contain no comments to indicate which event procedure they should be copied to. You must examine each code block, determine what the code does, and then paste the code into the correct event procedure. In the file master.aspx.cs, you will find 5 procedures with a comment indicating that the procedure needs one of the code blocks from the code block text file.

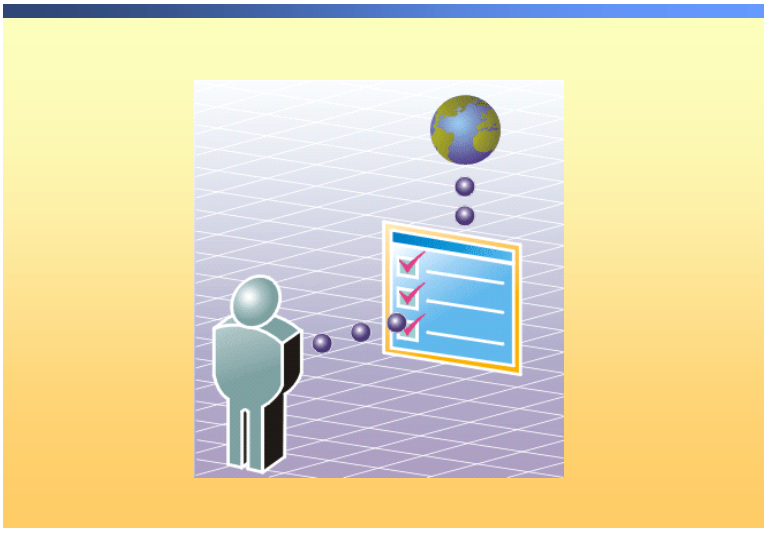
Tasks	Detailed steps
1. Open the Code Editor for master.aspx.	▪ In Solution Explorer, right-click **master.aspx**, and then click **View Code**.
2. Open the text file **Code Blocks.txt**.	▪ In Solution Explorer, right-click **Code Blocks.txt**, and then click **Open**.
3. Read each code block and determine its functionality. Paste the code block into the correct event procedure in master.aspx. Repeat for each code block.	**a.** Read the first code block and determine its functionality. **b.** Highlight the lines of code for the code block, and then on the **Edit** menu, click **Copy**. **c.** In the Code Editor, click the **master.aspx.cs** tab. **d.** Scroll down the code to the area where the event procedures contain the comment **Code block required here**. Click inside the correct event procedure. **e.** Press ENTER to start a new line. **f.** On the **Edit** menu, click **Paste**. **g.** Repeat steps **a** through **f** until all event procedures in master.aspx.cs have code.

Exercise 3
Testing the Application

In this exercise, you will test that the application functions correctly.

Tasks	Detailed steps
1. Run the application. If prompted to set the initial Web page, select **master.aspx.**	▪ On the standard toolbar, click **Start**. If prompted to set the initial Web page, select **master.aspx**.
2. Log on to the application using Customer Number: 100 Password: Password	**a.** In the AdventureWorks Internet Banking form, in the **Customer number** box, enter **100** **b.** In the **Password** box, type **Password** and then click **Logon**.
Note: If you entered the customer number and password correctly, the page displays the accounts that are held by this customer. If you failed to log on, try logging on again. If you still cannot log on or your application generates an error, you must debug the application.	
3. Display the transactions for account 1000.	▪ In the list of accounts, in the row for account 1000, click **View**.
4. Request $1000 of traveler's checks in British Pounds Sterling.	**a.** Click **Request Travelers Checks**. **b.** In the **Amount of Travelers Checks in USD** box, type **1000** **c.** In the **Check Currency** list, click **British Pounds Sterling**, and then click **Order**.
Note: A message should appear on the page stating "**Order reference is *nn* for 720.00 British Pounds Sterling Made up of 1 500s 4 50s 2 10s. Total cost $1,1014.08**".	

Course Evaluation



Your evaluation of this course will help Microsoft understand the quality of your learning experience.

At a convenient time between now and the end of the course, please complete a course evaluation, which is available at http://www.metricsthatmatter.com/survey.

Microsoft will keep your evaluation strictly confidential and will use your responses to improve your future learning experience.