What is a Mobile Form? When do you use them?
You use ASP.NET web forms to build web pages for HTML PC clients. ASP.NET mobile web forms allow you to build pages for mobile clients, regardless of the markup languages they support.

Wireless developers have to cope with a confusing variety of different devices – small or large devices with different size screens, in color or monochrome, and that require one of the HTML, cHTML, WML, or XHTML markup languages, and quite often a specific "dialect" of one of those.
-- Wigley & Roxburgh, Ch. 1

Do you have to use a specific device (PDA, phone, etc) to access a Mobile Form? How do you design for multiple / different devices?
No, any device an access a mobile form – even a desktop computer.

You can specify different content or different parameters based on the type of device. The DeviceSpecific/Choice constructs allow you to customize your application for specific devices – for example, to set the property of a control to one value on one device, and differently on others. Each construct consists of a DeviceSpecific element and one or more Choice child elements. Thus, the server control syntax declaration of any control that inherits from System.Web.UI.MobileControl can contain a single <DeviceSpecific> element. A <DeviceSpecific> element can contain any number of <Choice> elements. You use the <Choice> element to define each device.
-- Wigley & Roxburgh, Ch. 1

What is Adaptive Rendering?
Adaptive rendering is a term which means to make the technology on the client device an irrelevance – or to put in another way, to use middleware on the Web server that takes care of worrying about the specific requirements of a particular client device on your behalf.

The ASP.NET Mobile Controls use adaptive rendering to support over 200 different handheld devices from a single application.
-- Wigley & Roxburgh, Ch. 1

How are ASP.NET Mobile Controls different from Web Controls?
In general, the mobile controls are very similar to the web controls; however, they are located in a different namespace, as shown here:

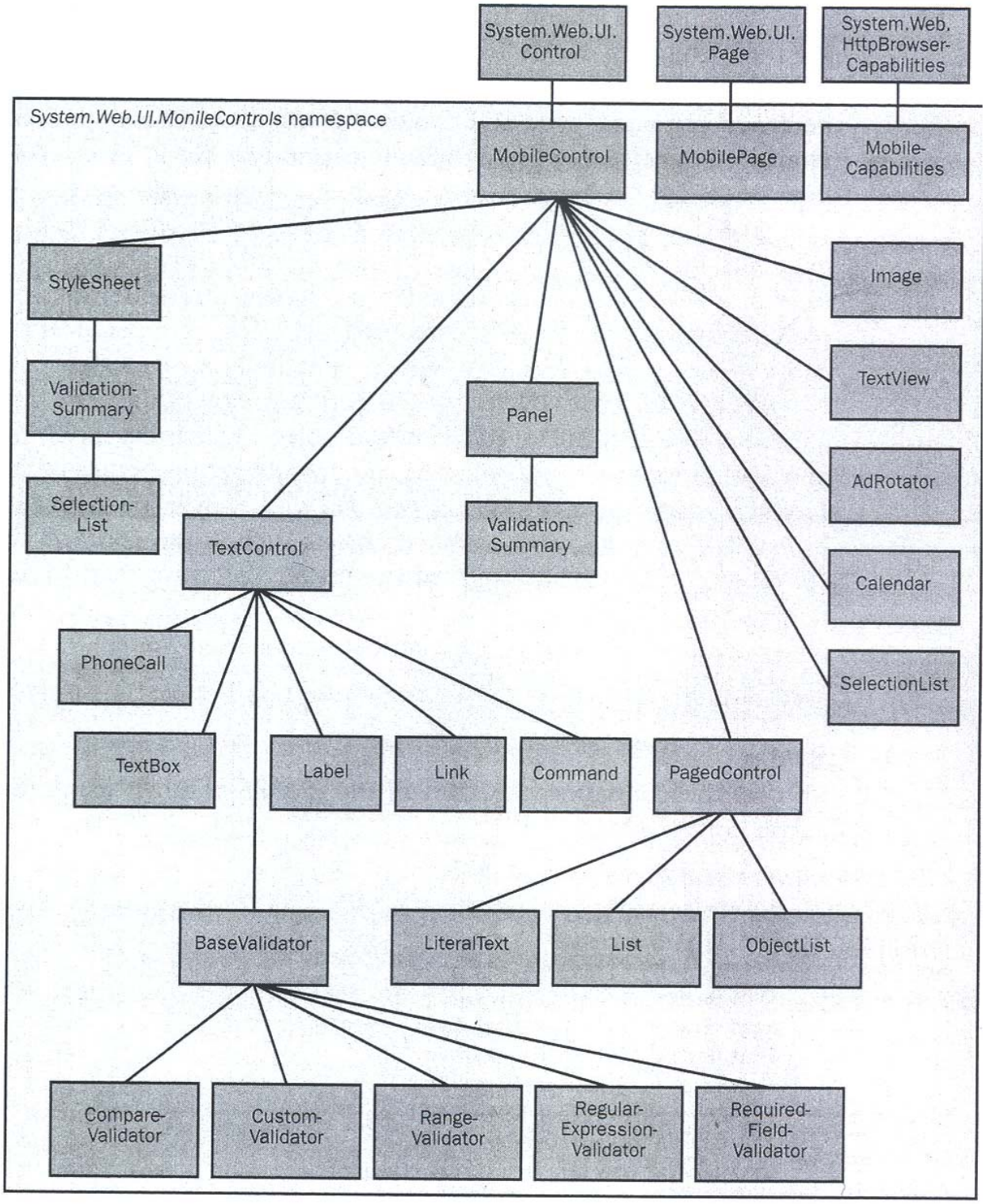

Figure 4-1 The top-level hierarchy of the mobile controls

You are creating an ASP.NET application that accesses a SQL Server 2000 database to provide production supervisors with planning data. Supervisors use this application to determine workload and to schedule necessary personnel.

At the top of the main page, the application needs to display the total number of orders received for the day. You create a SqlConnection object and open the connection. Then, you create a SqlCommand object, named cmdDailyOrders, and set the CommandText property to:

SELECT COUNT(*) FROM Order WHERE YEAR(Order.OrderDate) = YEAR(GETDATE()) AND
MONTH(Order.OrderDate) = MONTH(GETDATE()) AND
DAY(Order.OrderDate) = DAY(GETDATE())

You want to return the result to the page using the most efficient approach.

Which method should you use?

Read

ExecuteReader

ExecuteScalar

ExecuteNonQuery

Explanation:
You should use the ExecuteScalar method. You can use the ExecuteScalar method of a SqlCommand object to execute the underlying command and return a single value. The first column in the first row of the query result is returned. If the query result includes additional rows or columns, they are ignored. This would be the most efficient approach because in this scenario you only return one value.

You should not use the Read method because the SqlCommand object that you have created does not support a Read method. The Read method is supported by other objects, such as the SqlDataReader, OleDbDataReader, StreamReader, and TextReader.

You should not use the ExecuteReader method. The ExecuteReader method accepts a connection and a command and creates a DataReader, either SqlDataReader or OleDbDataReader. In this scenario, you have not defined a DataReader.

You should not use the ExecuteNonQuery method. The ExecuteNonQuery method can be used to execute Transact-SQL statements, such as DML or SET statements, which do not return a result. You could use this method to execute an INSERT, UPDATE, or DELETE statement.

Your company is creating a new online presence for itself. You are part of the team developing the ASP.NET application. The layout for most pages includes a standard page header with a corporate logo, a single-level menu, and some simple graphics. The header is being designed by a graphic arts consultant. You will incorporate the header into most of your pages. Your project sponsor has requested the ability to easily change the look of the header (e.g., for special time-of-year graphics such as Christmas trees and candy canes in December) with as little programmer, webmaster, and system administrator involvement as possible.

The header is essentially static content to be included in many different .aspx pages and would be a good candidate for fragment caching.

How should your team plan to incorporate the header when the graphic designer passes it off to you?

as an XML Web service

as a Web user control in an .ascx file

as clear text in a SQL Server database

as a data blob in a SQL Server database

as a Web custom control in an .ascx file

Explanation:
Your team should implement the header as a Web user control in an .ascx file. Because the header needs to be easily changed in the future, you should implement the header so that the content is quickly retrievable and updateable. An .ascx file can be retrieved through a simple file copy, and an .ascx file that contains no server-side code is no more complex than a standard HTML document for a graphic designer to modify. Using an .ascx is the only option that would not require any C# coding. It would require only two lines in each .aspx file: a @ Register directive and a server control instance (a tag using the tag prefix and name specified in the @ Register directive). You could also use the Visual Studio .NET IDE to drag a glyph representing the control onto the Design view of each .aspx page with no coding at all. The scenario also states that the header is a good candidate for fragment caching, which can only be accomplished using .ascx files. You could also use the Cache object if retrieving the header content programmatically from a database or Web service, but this would also increase the number of lines of code required in each .aspx file.

While your team could implement the header as an XML Web service, as a data blob in a SQL Server database, or as clear text in a SQL Server database, this would add additional complexity and would not be the best choice. In addition, none of the options would allow you to use fragment caching with the header.

Your team should not implement the header as a Web custom control in an .ascx file. An .ascx file defines a Web user control, not a Web custom control. A Web user control is an XML document that represents a UserControl class, which may or may not be derived from a code-behind class. A custom control is a component that derives from a server control and is defined programmatically.

Your Web application is intended to be used concurrently on multiple servers. When a user visits the application's home Web form, the user will submit his name and address. This information must be available to whichever instance of the application the user is connected to when he navigates to other Web forms. You would like all of this information stored on a server, and you are very concerned with security.

Which is the best method to store this information?

Use cookies to store the information.

Use ViewState to store the information.

Store the information as Session state.

Store the information as Application state.

Explanation:
You should store this information as Session state. Session information is stored separately from the application and can be accessed by all of the various instances of the application.

You should not use ViewState or cookies to store the information. Both of these are client-side techniques and this scenario requires server-side storage.

You should not store the information as Application state. Application state is not shared across a Web farm or garden.

You are getting ready to deploy your Web application to the server, and you will be using the XCOPY method. You must ensure that the correct files are placed into the correct folders. Your application will use its own configuration information, so you will be including a web.config file with your application.

To which folder should this web.config file be copied?

- the wwwroot folder
- **the application's root folder**
- the bin folder under the wwwroot folder
- the bin folder under the application's root folder

Explanation:
The web.config file should be copied to the application's root folder. For your application to have its own configuration settings that will override those set on the server, you must include a web.config file with the application. This web.config file must be placed in the application's root folder.

You should not copy the web.config file to the wwwroot folder. If you did, the settings would be applied to all Web applications on the same server.

You should not copy the web.config file to the bin folder under the wwwroot folder or the bin folder under the application's root folder. The .NET Framework will look in the application's root folder, not either of these locations.

You recently developed and are beta testing a new ASP.NET application. The application contains a page that uses a SqlDataAdapter to manage data retrieved from the Product table in your Microsoft SQL Server 2000 database. One of the columns in the underlying Product table, ProdType, contains a NOT NULL constraint.

Your page contains a DataGrid that allows users to display and update multiple rows of data at a time. The user can update these fields in the DataGrid: ProdDescription, ProdType, Price, and Category. The user makes the appropriate updates in the grid and then submits the page.

While testing the page, you notice that if you leave the ProdType field blank an error occurs. You would like to be able to identify the errors that occur, but you do not want the updates that are valid to be interrupted. Instead, you would like all updates that are valid to be performed.

What should you do before calling the Update method?

Set the ContinueUpdateOnError property of the SqlDataAdapter to true.

Set the MissingSchemaAction property of the SqlDataAdapter to MissingSchemaAction.Ignore.

Set the MissingMappingAction property of the SqlDataAdapter to MissingMappingAction.Ignore.

Set the MissingMappingAction property of the SqlDataAdapter to MissingMappingAction.Passthrough.

Explanation:
Before you call the Update method of the SqlDataAdatper, you should set the ContinueUpdateOnError property of the SqlDataAdapter to true. When set to true, the ContinueUpdateOnError property of the DataAdapter specifies that when an error occurs during a row update that no exception will be thrown. If the update is valid, it is performed, but if the update is not valid, it is bypassed and the error information is recorded in the RowError property of the row that contains the error. This will allow all valid updates to process successfully. All errors can then be reviewed and handled accordingly. If ContinueUpdateOnError is set to false, an exception is immediately thrown each time an error occurs during update.

You should not set the MissingSchemaAction property of the SqlDataAdapter to MissingSchemaAction.Ignore. The MissingSchemaAction property of a DataAdapter specifies the action to perform when incoming data does not match the DataSet schema. Setting this property to Ignore would ignore schema differences.

You should not set the MissingMappingAction property of the SqlDataAdapter to MissingMappingAction.Ignore or MissingMappingAction.Passthrough. The MissingMappingAction property of the DataAdapter specifies the action to perform when incoming data does not have a corresponding table or column.