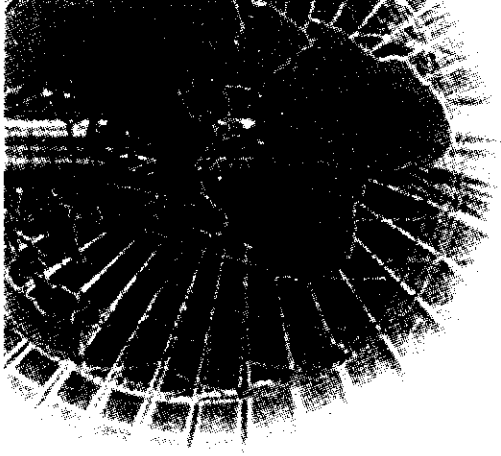


20

Introducing Web Parts Controls

If you need information on:	*See page:*
The WebPartManager Control	750
Using the WebPartManager Class	[illegible]
The ProxyWebPartManager Control	761
The WebPartZone Control	[illegible]
Creating the WebPartZone Control in Code	765
The CatalogZone Control	[illegible]
Creating the CatalogZone Control in Code	769
The DeclarativeCatalogPart Control	[illegible]
The PageCatalogPart Control	773
Creating the PageCatalogPart Control in Code	[illegible]
The ImportCatalogPart Control	776
The EditorZone Control	[illegible]
Creating the EditorZone Control in Code	781
The LayoutEditorPart Control	[illegible]
The AppearanceEditorPart Control	784
The PropertyGridEditorPart Control	[illegible]
Creating the PropertyGridEditorPart Control in Code	787
The BehaviorEditorPart Control	[illegible]
The ConnectionsZone Control	791
Creating the ConnectionsZone Control in Code	[illegible]

Web Parts controls enable you to build flexible websites that can be customized dynamically at run time. Web Parts are objects of a Web form on which users can perform multiple functions, such as open, close, minimize, or modify the content, behavior, or appearance of a Web Parts control, according to the requirements. The `Portal` framework is used to create Web pages with multiple Web Parts controls.

In this chapter, we focus on Web Parts controls and their classes that form an integral part of the Portal framework. This chapter also describes how to use Web Parts controls in Visual Studio 2008 to create customizable websites. In addition, you learn how to declare controls in Visual Basic. This chapter also covers the members (properties, methods, and events) of various Web Parts control classes. These members are essential to develop a customizable website.

This chapter mainly deals with the following Web Parts controls:

- WebPartManager
- ProxyWebPartManager
- WebPartZone
- CatalogZone
- DeclarativeCatalogPart
- PageCatalogPart
- ImportCatalogPart
- EditorZone
- LayoutEditorPart
- AppearanceEditorPart
- PropertyGridEditorPart
- BehaviorEditorPart
- ConnectionsZone

Now, let's discuss these controls one by one.

The WebPartManager Control

The `WebPartManager` control acts like a centralized hub that manages all the other Web Parts controls on a Web page. This control coordinates the interaction between Web Parts and Web Parts zones. The `WebPartManager` control manages the personalization states of Web Parts controls on a Web page. Personalization allows storing of user information and all the runtime customizations (defined by the user) in a persistent storage (typically SQL Server database). These customizations are loaded when the user visits the website the next time. If you create a Web page that uses Web Parts controls, ensure that the page contains a `WebPartManager` control. There must be only one instance of the `WebPartManager` control on each Web page that uses the control, and it must be placed before other Web Parts Zone controls are placed on the Web page. The `WebPartManager` control is an object of the `WebPartManager` class.

The `WebPartManager` control performs the following tasks to control the functionality of a Web page:

- Keeping track of different controls on a Web page.
- Inserting and removing controls on a Web page.
- Establishing, monitoring, and managing connections between various Web Parts controls.
- Enabling you to customize the appearance of a Web page by dragging various controls to different locations on the page.
- Providing various views, which you can use to change and personalize the properties and behavior of controls.
- Enabling you to toggle between different views of a Web page, thereby simplifying certain tasks, such as modifying page layout and editing controls.

- Enabling you to define and raise the lifecycle events associated with a Web Parts control. Lifecycle events keep track of the Web Parts control to determine when the control is inserted, moved, connected, or removed.

You can insert a `WebPartManager` control by dragging it from the Toolbox and dropping it on the Design view of an `.aspx` page. Figure 20.1 shows the `WebPartManager` control in the Design view:

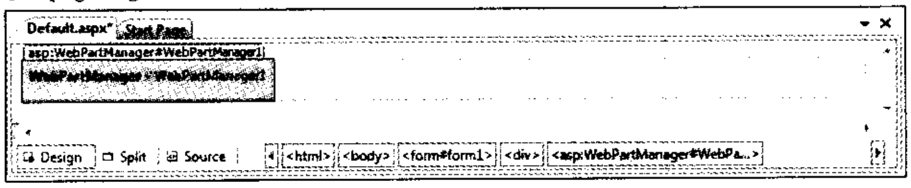

Default.aspx*
Start Page
asp:WebPartManager#WebPartManager1
WebPartManager - WebPartManager1
Design
Split
Source
<html>
<body>
<form#form1>
<div>
<asp:WebPartManager#WebPa...>

Figure 20.1: The WebPartManager Control

You can also add the `WebPartManager` control to an application by using the following code in the `<form>` element of the `.aspx` page:

```
<form id="form1" runat="server">
<asp:WebPartManager ID="WebPartManager1" runat="server">
  </asp:WebPartManager>
  </form>
```

NOTE

To use Web Parts controls on each page of a website, a separate WebPartManager control has to be used.

If, however, you want to use a single WebPartManager control on a multi-paged website containing Web Parts controls, place the WebPartManager control on the master page of the website.

The `WebPartManager` control is created by using the `WebPartManager` class of the .NET Framework class library. This class is available in the `System.Web.UI.WebControls.WebParts` namespace, which resides in the `System.Web` assembly. The inheritance hierarchy of the `WebPartManager` class is:

```
System.Object
  System.Web.UI.Control
        System.Web.UI.WebControls.WebParts.WebPartManager
```

To use the `WebPartManager` control, you need to declare it in the source code of the Web page as:

```
Dim <object name> As WebPartManager
```

The `WebPartManager` class is declared as:

```
Public Class WebPartManager
  Inherits Control
        Implements INamingContainer, IPersonalizable
```

Noteworthy methods of the `WebPartManager` class are listed in Table 20.1:

Table 20.1: Noteworthy Methods of the WebPartManager Class

Method	Description
`ActivateConnections`	Activates all currently inactive connections on a Web page
`AddWebPart`	Adds a WebPart control to a Web page
`BeginWebPartConnecting`	Begins the process of connecting two Web Parts controls
`BeginWebPartEditing`	Begins the process of modifying a Web Parts control
`BuildProfileTree`	Collects data about a server control and sends it to the `Trace` property, which is displayed on a Web page when tracing is enabled for the page
`CanConnectWebParts`	Checks Web Parts controls involved in a connection to determine if the controls are capable of being connected

Table 20.1: Noteworthy Methods of the WebPartManager Class

[illegible]	[illegible]
`CheckRenderClientScript`	Checks the ability of the browser to request for a Web page. This method also checks the value of the `EnableClientScript` property, which specifies whether to render the client script of the Web page or not
`ClearChildControlState`	Removes control-state information about the child controls of a server control. This method is inherited from the `Control` class
`ClearChildState`	Removes both view-state and control-state information about the child controls of a server control. This method is inherited from the `Control` class
`ClearChildViewState`	Removes view-state information about the child controls of a server control. This method is inherited from the `Control` class
`CloseWebPart`	Closes a Web Parts control so that it is not rendered on a Web page, but can be reopened on the Web page
`ConnectWebParts`	Connects two Web Parts controls that exist in a `WebPartZoneBase` zone
`CopyWebPart`	Creates a copy of a Web Parts or a server control and inserts the control in a Web page. This method is used by a Web Parts control set. A Web Parts control set is a group of components, which enables users to modify the appearance and behavior of Web pages directly from the browser
`CreateAvailableTransform ers`	Creates a group of transformers specified in a configuration file of a website. These transformers are then added to a collection of transformers, which is referenced by the `AvailableTransformers` property. Transformers are used to create connections between Web Parts controls
`CreateChildControls`	Determines the server control using composition-based implementation to create child controls for posting back or rendering
`CreateControlCollection`	Gets a collection of all the controls placed on a Web page that are managed by using the `WebPartManager` control
`CreateDisplayModes`	Groups the display modes related to a website that has Web Parts controls
`CreateDynamicConnectionI D`	Returns a unique identification number for a dynamic connection
`CreateDynamicWebPartID`	Creates a unique ID value for a dynamic Web Parts control
`CreateErrorWebPart`	Creates and inserts a special control into a Web page. This control is displayed only when an error occurs while loading or creating a dynamic Web Parts control
`CreatePersonalization`	Creates a personalization object that stores the personalization data of the user of a particular Web page
`CreateWebPart`	Provides Web Parts control functionality to a server control that is not a Web Parts control
`DeleteWebPart`	Deletes a dynamic instance of a WebPart control
`DisconnectWebPart`	Deletes a Web Parts or a server control, which has been closed or removed from the connections in which it is participating
`DisconnectWebParts`	Starts the process of disconnecting two WebPart or server controls on a Web page
`Dispose`	Disposes off a server control completely before releasing it from memory
`EndWebPartConnecting`	Completes the process of connecting two or more Web Parts controls
`EndWebPartEditing`	Completes the editing process of a Web Parts control

Table 20.1: Noteworthy Methods of the WebPartManager Class

[illegible]	[illegible]
`EnsureChildControls`	Identifies whether or not a server control contains child controls. This method also creates child controls for the server control if there are no child controls
`EnsureID`	Creates an identifier for a Web Parts or server control if the control does not have an identifier assigned to it
`Equals`	Determines whether a specified object is equal to the current object or not
`ExportWebPart`	Creates the XML file containing state and property information about the server control
`Finalize`	Enables an object to free resources and perform cleanup operations before the object is retrieved for garbage collection
`Focus`	Sets and removes the focus to or from a `WebPartManger` control
`GetConsumerConnectionPoints`	Get a collection of `ConsumerConnectionPoint` objects, which act as connection points from a server control
`GetCurrentWebPartManager`	Obtains the current instance of a `WebPartManager` control on a Web page
`GetDesignModeState`	Retrieves design-time data related to a `WebPartManager` control
`GetDisplayTitle`	Retrieves the string that contains the value for the `DisplayTitle` property of a Web Parts control
`GetExportUrl`	Retrieves the relative virtual path and the query string when a user tries to export a Web Parts control
`GetGenericWebPart`	Obtains a reference to the instance of the `GenericWebPart` control, which contains a server control
`GetHashCode`	Acts as a hash function for the object of the `WebPartManager` class. A hash function is used to encrypt and decrypt data to make it secure while transferring it over the Internet
`GetProviderConnectionPoints`	Obtains a collection of `ProviderConnectionPoint` objects that serve as connection points from a server control that is acting as a provider within a Web Parts connection
`GetType`	Returns a Type value for the current object
`HasControls`	Verifies whether a server control contains child controls or not
`HasEvents`	Obtains a value to specify whether events are registered for a Web Parts control or child controls
`ImportWebPart`	Imports an XML description file containing the state and property data for a Web Parts control
`IsAuthorized`	Determines if a Web Parts or server control can be added to a Web page
`IsLiteralContent`	Decides whether a server control stores only literal content. (Literal content is content that is in plain text only, and not in the form of images, controls, and so on
`LoadControlState`	Stores state data of a server control saved from a previous page request and return the data on a successive request
`LoadCustomPersonalizationState`	Stores custom personalization data, which is passed by personalization objects to a `WebPartManager` control. Personalization data is used at the time of the initialization process. Personalization objects are used to personalize Web Parts controls
`LoadViewState`	Returns view-state information requested by a previous page. The request is saved by the `SaveViewState` method

Table 20.1: Noteworthy Methods of the WebPartManager Class

[illegible]	[illegible]
`MapPathSecure`	Returns the physical path of a control, which is mapped by a virtual path. The virtual path can be either absolute or relative
`MemberwiseClone`	Constructs a copy of the current object
`MoveWebPart`	Changes the location of a server control from one `WebPartZoneBase` zone to another zone or to a new location within the same zone
`OnAuthorizeWebPart`	Invokes the `AuthorizeWebPart` event and its handler
`OnBubbleEvent`	Determines whether the event for a server control is passed up the UI server control hierarchy of a Web page
`OnConnectionsActivated`	Invokes the `ConnectionActivated` event to specify whether a Web page and its controls are loaded in the browser. It also indicates whether the connections on the Web page are activated to share data
`OnConnectionsActivating`	Invokes the `ConnectionActivating` event to indicate that a Web page and its controls are loaded in the browser and the process of activating connections on the Web page can begin
`OnDataBinding`	Invokes the `DataBinding` event
`OnDisplayModeChanged`	Invokes the `DisplayModeChanged` event to specify that the process of changing the display mode on a Web page has been finished by the `WebPartManager` control
`OnDisplayModeChanging`	Invokes the `DisplayModeChanging` event to specify that the `WebPartManager` control is in the process of changing the display mode on a Web page
`OnInit`	Invokes the `Init` event. This is the first event in the lifecycle of the `WebPartManager` control
`OnLoad`	Invokes the `Load` event
`OnPreRender`	Generates the `PreRender` event before a `WebPartManager` control is rendered on a Web page. This method notifies the `WebPartManager` control to perform any necessary pre-rendering steps prior to saving view-state and rendering content
`OnSelectedWebPartChanged`	Generates the `SelectedWebPartChanged` event after a Web Parts control is selected or its selection cleared
`OnSelectedWebPartChangin g`	Generates the `SelectedWebPartChanging` event while changing a selected `WebPart` control
`OnUnload`	Generates the base `Unload` event to remove a `WebPartManager` instance from a Web page
`OnWebPartAdded`	Generates the `WebPartAdded` event after a Web Parts control is added to a Web page
`OnWebPartAdding`	Generates the `WebPartAdding` event when a Web Parts control is added to a `WebPartZoneBase` zone
`OnWebPartClosed`	Generates the `WebPartClosed` event to indicate the removal of a Web Parts control from a Web page
`OnWebPartClosing`	Generates the `WebPartClosing` event when a Web Parts or server control is removed from a Web page
`OnWebPartDeleted`	Generates the `WebPartDeleted` event after a Web Parts control is permanently deleted from a Web page
`OnWebPartDeleting`	Indicates that a dynamic Web Parts control is being deleted

Table 20.1: Noteworthy Methods of the WebPartManager Class

[illegible]	[illegible]
`OnWebPartMoved`	Generates the `WebPartMoved` event after a Web Parts control is moved to another location on a Web page
`OnWebPartMoving`	Generates the `WebPartMoving` event to indicate that a Web Parts control in a `WebPartZoneBase` zone is in the process of being moved
`OnWebPartsConnected`	Generates the `WebPartsConnected` event after a connection is established between Web Parts controls
`OnWebPartsConnecting`	Generates the `WebPartsConnecting` event while establishing a connection between two Web Parts controls
`OnWebPartsDisconnected`	Generates the `WebPartsDisconnected` event after a connection between Web Parts controls terminates
`OnWebPartsDisconnecting`	Generates the `WebPartsDisconnecting` event, which indicates that two Web Parts controls in a `WebPartZoneBase` zone are going to terminate a connection
`OpenFile`	Retrieves a stream to read a file. The file path is specified as a parameter
`RaiseBubbleEvent`	Provides the sources as well as information of the event of a child control to its parent control
`RegisterClientScript`	Allows a `WebPartManager` control to provide client-side script used for personalization features, such as dragging Web Parts controls in a Web page
`RemovedControl`	Called when a server control is removed from the Controls collection of a Control object
`Render`	Overrides the `Control.Render` (HtmlTextWriter) method so that the `WebPartManager` control is prevented from rendering any content
`RenderChildren`	Provides the content of the child control of a server control to the `HtmlTextWriter` object, which writes the content to be rendered on the client
`RenderControl`	Displays the content of a server control and stores information related to tracing, if tracing is enabled
`ResolveAdapter`	Retrieves the control adapter that renders a specified control
`ResolveClientUrl`	Obtains a URL used by the browser in which you want to run your website
`ResolveUrl`	Converts a URL so that it is usable on the requesting client
`SaveControlState`	Stores state-related data for a `WebPartManager` control. You can restore the data by requesting the Web page containing the `WebPartManager` control
`SaveCustomPersonalizationState`	Stores custom personalization state data maintained by a `WebPartManager` control for a Web page, so that this data is reloaded whenever the Web page is reloaded
`SaveViewState`	Stores the changes in a server control view-state. This method saves only those changes that occur when a Web page is posted back to the server
`SetDesignModeState`	Sets design-time data for a control
`SetPersonalizationDirty`	Sets a flag, which specifies whether the custom personalization data for a `WebPartManager` control has changed or not
`SetRenderMethodDelegate`	Allocates an event handler delegate to provide the content of a server control to the parent control
`SetSelectedWebPart`	Sets the values for the `SelectedWebPart` property so that it is equal to the value of the currently selected Web Part control

Table 20.1: Noteworthy Methods of the WebPartManager Class

[illegible]	[illegible]
`ToString`	Returns a `String` value, which represents the current object
`TrackViewState`	Enables tracking of the changes in the view-state data of a `WebPartManager` control. To enable tracking, the method calls the base method, that is, `Control.TrackViewState()`

Noteworthy properties of the `WebPartManager` class are listed in Table 20.2:

Table 20.2: Noteworthy Properties of the WebPartManager Class

Property	Description
`Adapter`	Retrieves the browser-specific adapter for a `WebPartManager` control
`AppRelativeTemplateSourceDirectory`	Retrieves or specifies the application-relative virtual directory of a Web page or object that contains a `WebPartManager` control
`AvailableTransformers`	Retrieves a collection of `WebPartTransformer` objects available for creating Web Parts connections between server controls
`BindingContainer`	Retrieves a server control containing a `WebPartManager` control's data-binding properties
`ChildControlsCreated`	Retrieves a value that indicates whether or not the child controls of a server control are created
`ClientID`	Enables you to use a server control identifier generated by ASP.NET
`ClientIDSeparator`	Retrieves a character value that represents the separator character used in the `ClientID` property
`CloseProviderWarning`	Retrieves and specifies a warning message displayed if a user closes a server control acting as a provider to the other server controls in a connection
`Connections`	Retrieves a reference to a set of current connections on a Web page
`Context`	Retrieves the `HttpContext` object for the current Web request. The `HttpContext` object encapsulates request information, such as the user and machine accessing the website
`Controls`	Retrieves a collection of all WebPart, server, or user controls contained in the `WebPartZoneBase` zones on a Web page
`DeleteWarning`	Retrieves and specifies a custom warning message, displayed when a server control is deleted
`DesignMode`	Retrieves a value that indicates if a server control is used on a design surface
`DisplayMode`	Retrieves or specifies the display mode for a Web page containing Web Parts controls
`DisplayModes`	Retrieves a set of display modes in the read-only mode associated with a `WebPartManager` control
`DynamicConnections`	Retrieves a collection of dynamic connections that exist on a Web page
`EnableClientScript`	Retrieves or sets a value that specifies whether or not client-side scripting is enabled on a Web page containing `WebPartManager` control
`EnableTheming`	Retrieves a value that determines whether or not a theme is enabled on a Web page
`EnableViewState`	Retrieves or specifies a value to indicate whether a server control persists its

Table 20.2: Noteworthy Properties of the WebPartManager Class

	view-state, and the view-state of any child controls it contains, to the requesting client
`Events`	Retrieves a list of event handlers for a `WebPartManager` control
`ExportSensitiveDataWarning`	Retrieves or sets the text for a warning message, which is displayed when a user exports sensitive data from a WebPart control
`HasChildViewState`	Retrieves a value that indicates whether or not the child controls of the current server control have saved view-state settings
`ID`	Retrieves or sets the identifier of a Web server control
`IdSeparator`	Retrieves the character used for separating control identifiers for the child controls of the `WebPartManager` control
`Internals`	Retrieves a reference to the `WebPartManagerInternals` class, which is used to combine and separate a set of methods implemented in the `WebPartManager` class
`IsChildControlStateCleared`	Retrieves a value indicating whether the child controls of the `WebPartManager` control possess control-state. Control-state holds information specific to a control, which is essential to send to the server at the time of postback. Control-state cannot be disabled as in the case of view-state
`IsCustomPersonalizationStateDirty`	Retrieves a value that indicates whether any personalization changes have been made that affect page-level personalization details of a `WebPartManager` control
`IsTrackingViewState`	Retrieves a value indicating that a server control is saving the changes to its view-state
`IsViewStateEnabled`	Retrieves a value that indicates whether the view-state is enabled for a `WebPartManager` control
`LoadViewStateByID`	Retrieves a value that indicates whether the view-state of a `WebPartManager` control can be loaded by `ID` instead of index
`NamingContainer`	Retrieves the reference to the naming container of the Web server control to create a unique namespace for every Web server control
`Page`	Retrieves a reference to the Web page instance that contains a `WebPartManager` control
`Parent`	Retrieves a reference to the parent control of a server control in the control hierarchy of a Web page
`Personalization`	Retrieves a reference to the object containing the data needed to personalize a Web page
`SelectedWebPart`	Returns a reference to the currently selected Web Parts or server control
`Site`	Returns information about the container control, which stores the current control when it is modified in the Design view
`SkinID`	Gets or sets an empty string to ensure that a user cannot apply any skin to a `WebPartManager` control
`StaticConnections`	Returns a reference to the collection of all `WebPartConnection` objects that are defined as static connections on a Web page
`SupportedDisplayModes`	Retrieves a set of all display modes available on a particular Web page. This is a read-only property

Table 20.2: Noteworthy Properties of the WebPartManager Class

`TemplateControl`	Gets or sets the reference to the template containing the current control
`TemplateSourceDirectory`	Returns the virtual directory of the Web page containing the `WebPartManager` control. This property is inherited from the `Control` class
`UniqueID`	Returns a unique identifier for a server control. This property is inherited from the `Control` class
`ViewState`	Retrieves a dictionary of state information that allows users to save and restore the view-state of a server control across multiple requests for the same Web page
`ViewStateIgnoresCase`	Gets a value that indicates whether the `StateBag` object is case-sensitive
`Visible`	Retrieves a value to make the child controls of a `WebPartManager` control visible
`WebParts`	Returns a reference to all WebPart controls on a Web page that are monitored by a `WebPartManager` control
`Zones`	Returns a reference to a collection of all `WebPartZoneBase` zones included on a Web page

Noteworthy events of the `WebPartManager` class are listed in Table 20.3:

Table 20.3: Noteworthy Events of the WebPartManager Class

`AuthorizeWebPart`	Initiated when the `IsAuthorized` method is invoked. This event determines whether a WebPart or server control can be added to a Web page.
`ConnectionsActivated`	Initiated after all Web Parts connections specified on a Web page connect and start sharing data between consumer and provider controls. Consumer controls are those controls that receive data while provider controls are those that send data.
`ConnectionsActivating`	Initiated during the process of activating all established Web Parts connections on a Web page.
`DataBinding`	Initiated after a server control binds to a data source. This event is inherited from the `Control` class.
`DisplayModeChanged`	Initiated when the display mode of a Web page is changed.
`DisplayModeChanging`	Initiated after a user clicks a verb (a menu item displayed by a Web Part) on a Web page that begins the process of changing the display mode of Web page.
`Disposed`	Initiated when a server control of a Web page is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested.
`Init`	Initiated when a server control is initialized.
`Load`	Initiated when a server control is loaded into the Web page.
`PreRender`	Initiated when a server control is loaded but not yet rendered.
`SelectedWebPartChanged`	Initiated after moving or changing a selected WebPart or server control to another control.
`SelectedWebPartChanging`	Initiated during the process of changing the selected Web Parts or a server control.
`Unload`	Initiated when a server control is unloaded from memory.
`WebPartAdded`	Initiated when a dynamic Web Parts or server control is added to a

Table 20.3: Noteworthy Events of the WebPartManager Class

	`WebPartZoneBase` zone. This event indicates that the control is added successfully.
`WebPartAdding`	Initiated when a dynamic Web Parts or server control is added to a `WebPartZoneBase` zone at run time. This event indicates that the control is added successfully.
`WebPartClosed`	Initiated when a WebPart control is deleted from a Web page.
`WebPartClosing`	Initiated during the process of deleting a WebPart control from a Web page.
`WebPartDeleted`	Initiated after a Web Parts or server control is deleted from a `WebPartZoneBase` zone.
`WebPartDeleting`	Initiated when permanently deleting an instance of a dynamic WebPart control from a `WebPartZoneBase` zone.
`WebPartMoved`	Initiated when the location of a Web Parts or server control on a Web page has changed.
`WebPartMoving`	Initiated when a Web Parts or server control contained in the `WebPartZoneBase` zone is in the process of moving.
`WebPartsConnected`	Initiated after a connection between WebPart controls has been established.
`WebPartsConnecting`	Initiated during the process of establishing a connection between WebPart controls in a `WebPartZoneBase` zone.
`WebPartsDisconnected`	Initiated during the termination of the connection between two WebPart or server controls.
`WebPartsDisconnecting`	Initiated during the process of terminating the connection between WebPart or server controls.

Using the **WebPartManager** Class

The `WebPartManager` class allows users to toggle between various display modes. These display modes help the user to perform different actions on the Web page, such as changing the layout of a Web page or editing controls, and the rest. The `WebPartManager` class has the following five display modes:

- `BrowseDisplayMode`—Specifies the default display mode for pages that contain Web Parts controls
- `CatalogDisplayMode`—Specifies the display mode in which a catalog of controls is visible. A user can add controls to a Web page from the catalog
- `ConnectDisplayMode`—Represents the display mode in which the connection UI is visible and where users can manage the connections between the controls of Web page
- `DesignDisplayMode`—Represents the display mode in which the user can modify the layout of a Web page
- `EditDisplayMode`—Specifies the display mode in which the user can change the appearance, properties, and behavior of server controls

*Creating the **WebPartManager** Control in Code*

As discussed in this chapter, you can create a `WebPartManager` control directly by using its class hierarchy. The `WebPartManager` control does not have any visual appearance in a Web page; therefore, the display mode property of the `WebPartManager` control is set to a Label control's Text property to see the corresponding default mode.

To learn how to create the `WebPartManager` control through the control's class hierarchy, let's create an application named `WebPartManagerControlAppVB`. You can find the code of `WebPartManagerControlAppVB` application in the Code\ASP.NET\Chapter

20\WebPartManagerControlAppVB folder on the CD. Listing 20.1 shows the complete code for the Default.aspx page of WebPartManagerControlAppVB website:

Listing 20.1: Showing the Code for Default.aspx Page

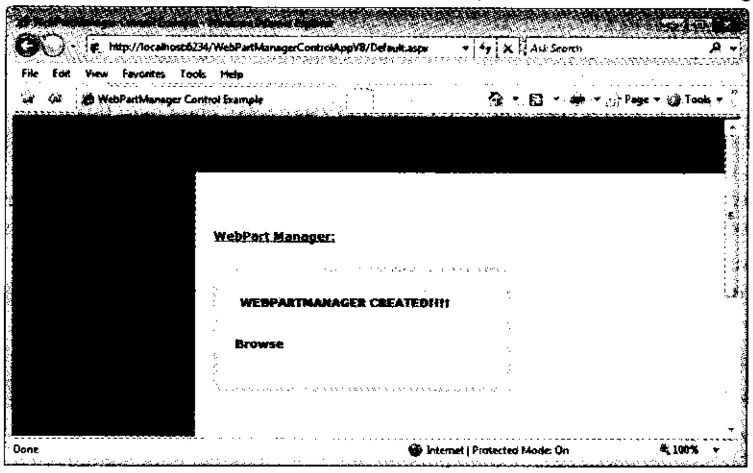
```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>WebPartManager Control Example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
        <div>
        <div id="header">
        </div>
        <div id="sidebar">
        <div id="nav">
        &nbsp;
        </div>
        </div>
        <div id="content">
        <p>
        &nbsp;</p>
        <div class="itemContent">
        <asp:Label ID="Label3" runat="server" Font-Bold="True" Font-
        Underline="True" ForeColor="#0000C0"
        Text="WebPart Manager:"></asp:Label>
        <br />
        <br />
        <asp:Panel ID="Panel1" runat="server" BackColor="#FFE0C0"
        BorderColor="#FFC080" BorderStyle="Double"
        Height="129px" Width="320px">
        &nbsp; &nbsp; &nbsp;<br />
        &nbsp;&nbsp; &nbsp;<asp:Label ID="Label1" runat="server" Font-
        Bold="True" ForeColor="Navy"
        Text="WEBPARTMANAGER CREATED!!!!"
        Visible="False"></asp:Label><br />
        <br />
        <asp:Button ID="Button2" runat="server" BackColor="#FFC080" Font-
        Bold="True" ForeColor="#0000C0" Text="DisplayMode" Width="213px" />
        &nbsp; &nbsp;
        <asp:Label ID="Label2" runat="server" Font-Bold="True"
        ForeColor="#0000C0" Visible="False"></asp:Label><br />
        &nbsp; &nbsp;</asp:Panel>
        &nbsp;
        </div>
        </div>
        <div id="footer">
        <p class="left">
        All content copyright &copy; Kogent Solutions Inc.</p>
        </div>
        </div>
    </form>
</body>
</html>
```

Listing 20.2 shows the complete code for the code-behind file of WebPartManagerControlAppVB website:

Listing 20.2: Showing the Code for the Code-Behind File:

```
Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Button2_Click(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles Button2.Click
        Label2.Visible = True
        Button2.Visible = False
        Dim wpm As New WebPartManager
        Dim str As String
        Str = wpm.DisplayMode.Name
        Label2.Text = str
        Label1.Visible = True
    End Sub
End Class
```

Now, run the website and click the DisplayMode Button; the output will be as shown in, Figure 20.2:


http://localhost6234/WebPartManagerControlAppV8/Default.aspx
Ask Search
File Edit View Favorites Tools Help
WebPartManager Control Example
Page Tools
Done
Internet | Protected Mode: On
100%

WebPart Manager:

WEBPARTMANAGER CREATED!!!!

Browse

Figure 20.2: The WebPartManager Control Example

The ProxyWebPartManager Control

As you know, a Web Parts application (an application using Web Parts controls) can contain a single `WebPartManager` control for each Web page, which manages all the Web Parts controls on that page. You also know that if the Web Parts application contains a master page and the `WebPartManager` control is placed in the master page, all the content pages (Web pages inherited from the master page) use the `WebPartManager` control of the master page to manage their Web Parts controls. To do this, each content page declares a static link to connect to the `WebPartManager` control of the master page. To declare a static link in the content page, you need to add the `ProxyWebPartManager` control and the `<asp:WebPartConnection>` element.

The `<asp:WebPartConnection>` element is derived from the `<staticconnections>` element, which is further derived from the `<asp:WebPartManager>` element. In this way, the `ProxyWebPartManager` control imitates the `WebPartManager` control of the master page in the content page. However, remember that, because the `WebPartManager` control is already declared in the master page once, you cannot declare additional `WebPartManager` controls in the content pages. If you do want to declare more `WebPartManager` controls, you have to use the `ProxyWebPartManager` control. This is how a `ProxyWebPartManager` control imitates a `WebPartManager` control residing in a master page.

You can add a `ProxyWebPartManager` control to a Web page by dragging and dropping the control from the Toolbox to the Design view. Figure 20.3 shows the `ProxyWebPartManager` control in the Design view:

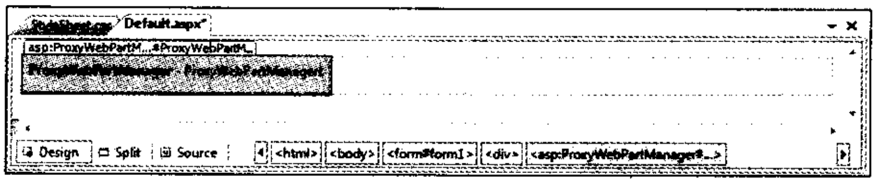

Default.aspx*
asp:ProxyWebPartM...#ProxyWebPartM...
Design
Split
Source
<html>
<body>
<form#form1>
<div>
<asp:ProxyWebPartManager#...>

Figure 20.3: The ProxyWebPartManager Control

Although the `ProxyWebPartManager` control is used to replace the `WebPartManager` control for declaring static connections, the `ProxyWebPartManager` class is not derived from the `WebPartManager` class. Rather, it is derived from the `Control` class.

The `ProxyWebPartManager` control is an object of the `ProxyWebPartManager` class. You can also create the `ProxyWebPartManager` control by using the `ProxyWebPartManager` class, which belongs to the `System.Web` assembly. The namespace hierarchy of the `ProxyWebPartManager` class is:

```
System.Object
  System.Web.UI.Control
      System.Web.UI.WebControls.WebParts.ProxyWebPartManager
```

The `ProxyWebPartManager` class inherits various properties from the `Control` class, such as `EnableTheming`, `Visible` and so on. The `ProxyWebPartManager` class uses these properties by overriding them. In addition to various inherited properties, the `ProxyWebPartManager` class also contains a non-inherited property, called `StaticConnections`. The `StaticConnections` property gets a collection of static connections declared within the `<asp:ProxyWebPartManager>` element on a content page. Similar to properties, the `ProxyWebPartManager` class also inherits various events and methods from the `Control` class.

The syntax to declare the `ProxyWebPartManager` control is:

```
Dim <object name> As ProxyWebPartManager
```

The `ProxyWebPartManager` class is declared as:

```
Public Class ProxyWebPartManager
Inherits Control
```

The WebPartZone Control

Technically, the `WebPartZone` control is used to create a region on the Web page where Web Parts or server controls can be relocated, maximized, or minimized according to the requirements specified by a user.

The `WebPartZone` control is an object of the `WebPartZone` class that permits you to create zones on a Web page. After the zone is created, you can place any server control in the zone.

`WebPartZone` controls are used to design the user interface (UI) of Web Parts applications. These controls enable you to host ASP.NET-based server-side controls on a Web page. At runtime, these server-side controls function as Web Parts controls.

The `WebPartZone` control is an object of the `WebPartZone` class, which allows you to create zones in Web pages.

To add a `WebPartZone` control to a Web page, drag and drop the control from the Toolbox to the Design view of an `.aspx` page. Figure 20.4 shows the `WebPartZone` control in the Design view:

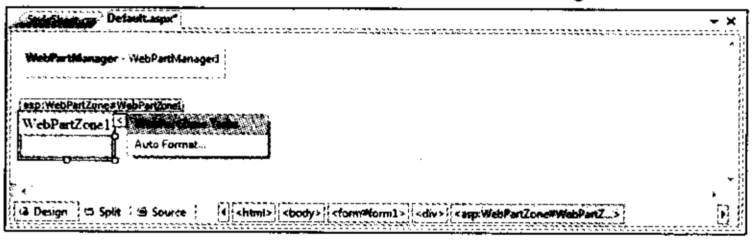

Default.aspx*
WebPartManager - WebPartManager1
asp:WebPartZone#WebPartZone1
WebPartZone1
Auto Format...
Design
Split
Source
<html>
<body>
<form#form1>
<div>
<asp:WebPartZone#WebPartZ...>

Figure 20.4: The WebPartZone Control

You can also add the WebPartZone control by inserting the following code in the <form> element of the .aspx page:

```
<asp:WebPartZone ID="WebPartZone1" runat="server">
   </asp:WebPartZone>
```

NOTE

The WebPartZone control also contains the <ZoneTemplate> element, which is used for storing other server controls, such as Label or Textbox.

The WebPartZone control is created by using the WebPartZone class of the .NET Framework class library. The WebPartZone class derives most of its functionalities from the base WebZone and WebPartZoneBase classes. The WebPartZone class is available in the System.Web.UI.WebControls.WebParts namespace. The namespace hierarchy of the WebPartZone class is:

```
System.Object
  System.Web.UI.Control
        System.Web.UI.WebControls.WebControl
                System.Web.UI.WebControls.CompositeControl
                        System.Web.UI.WebControls.WebParts.WebZone
                                System.Web.UI.WebControls.WebParts.WebPartZoneBase
                                        System.Web.UI.WebControls.WebParts.WebPartZone
```

You can use the various properties and methods of the WebPartZone class to use the WebPartZone control. The properties of the WebPartZone class allow you to set the appearance and behavior of the WebPartZone control on a Web page. The WebPartZone control is the primary control to add Web Parts controls on the Web page. You can add server controls in the WebPartZone control, which can be added to the Web page.

The syntax for creating a WebPartZone control is:

```
Dim <object name> As WebPartZone
```

Using the .NET Framework class library, the WebPartZone class is declared as:

```
Public Class WebPartZone
  Inherits WebPartZoneBase
```

Noteworthy methods of the WebPartZone class are listed in Table 20.4:

Table 20.4: Noteworthy Methods of the WebPartZone Class

[illegible]	[illegible]
CreateWebPartChrome	Enables derived zones to substitute a custom WebPartChrome object to the appearance of Web Parts controls in a zone
EditWebPart	Initiates the process of editing a selected Web Parts control in a zone
GetInitialWebParts	Overrides the abstract base method and retrieves the initial set of values of static WebPart controls contained within a zone's template
MinimizeWebPart	Minimizes a selected Web Parts control in a zone
OnCreateVerbs	Raises the CreateVerbs event
RenderDropCue	Provides UI elements to indicate where a Web Parts control is dragged and dropped within a zone
RestoreWebPart	Restores a selected minimized Web Parts control to the normal state

Noteworthy properties of the `WebPartZone` class are listed in Table 20.5:

Table 20.5: Noteworthy Properties of the WebPartZone Class

[illegible]	[illegible]
`AllowLayoutChange`	Determines whether or not the layout of a Web Parts control can be modified. This property also allows you to specify a layout for the Web Parts control
`DragDropEnabled`	Retrieves a value indicating whether or not Web Parts controls can be dragged and dropped into or out of a zone
`EditVerb`	Obtains a reference to the `WebPartVerb` class object to change a Web Parts control into the `WebPartsZone` control
`ExportVerb`	Obtains a reference to the `WebPartVerb` class object to export an XML definition for a Web Parts control
`HelpVerb`	Obtains a reference to the `WebPartVerb` class object to access help content for using Web Parts controls in a `WebPartZone` control
`LayoutOrientation`	Determines whether the controls in a `WebPartZone` control are aligned horizontally or vertically. This property also enables you to specify the alignment of the Web Parts controls in a `WebPartZone` control
`MenuCheckImageStyle`	Determines the image style used as a check mark for a verbs menu of a Web Parts control in a `WebPartZone` control
`MenuCheckImageUrl`	Determines or specifies a URL for the image used as a check mark for the verbs menu of a Web Parts control in a `WebPartZone` control
`MenuLabelHoverStyle`	Obtains or specifies the styles used to change the appearance of the label of a verbs menu when the mouse pointer is placed over it
`MenuLabelStyle`	Obtains or specifies the style applied to the label of a verbs menu that is displayed in the title bar of a Web Parts control in a `WebPartZone` control
`MenuLabelText`	Obtains or specifies the label for the verbs menu displayed in the title bar of a Web Parts control in a `WebPartZone` control
`MenuPopupImageUrl`	Obtains or specifies the URL of the image that represents the verbs menu in the title bar of a Web Parts control in the `WebPartZone` control
`MenuPopupStyle`	Obtains the style applied to the verbs menu of a Web Parts control in a `WebPartzone` control
`MenuVerbHoverStyle`	Obtains the style applied to change the appearance of the verb displayed in a verbs menu when the mouse pointer is placed over the verb
`MenuVerbStyle`	Obtains the style for the verb in a verbs menu of a Web Parts control in a `WebPartzone` control
`MinimizeVerb`	Obtains a reference to the `WebPartVerb` class object to minimize a `WebParts` control in a `WebPartsZone` control
`RestoreVerb`	Obtains a reference to the `WebPartVerb` class object to set the size of a `WebParts` control in a `WebPartZone` control to normal
`SelectedPartChromeStyle`	Obtains the style for a Web Parts control in a `WebPartZone` control
`ShowTitleIcons`	Determines the icons displayed in the title bar of a Web Part control in `WebPartsZone` control
`TitleBarVerbButtonType`	Determines the button used for the verbs in the title bar of a Web Parts control

Table 20.5: Noteworthy Properties of the WebPartZone Class

Property	Description
`TitleBarVerbStyle`	Obtains the style for the verbs displayed in the title bar of a Web Parts control
`VerbButtonType`	Retrieves or sets the type of button associated with the verbs in the title bar of a Web Parts control
`WebPartChrome`	Retrieves a reference to a `WebPartChrome` control that determines the peripheral rendering of a Web Parts control in a zone
`WebParts`	Retrieves a collection of Web Parts controls within a zone
`WebPartVerbRenderMode`	Retrieves or sets a value indicating how the verbs should be rendered on Web Parts controls in a zone
`ZoneTemplate`	Retrieves or sets the template reference containing controls in a Web page

Noteworthy event of the `WebPartZone` class is listed in Table 20.6:

Table 20.6: Noteworthy Event of WebPartZone Class

Event	Description
`CreateVerbs`	Initiated when the verbs are created for a zone derived from the `WebPartZoneBase` class

Creating the WebPartZone Control in Code

As learned earlier, the `WebPartZone` control is used to create a region where Web Parts or server controls can be relocated, maximized, or minimized by a user. Now, let's work with the `WebPartZone` control programmatically. After creating the various properties of a `WebPartZone` control, you can use the `WebPartZone` class to access and change these properties without using the UI at all.

Now, let's create a website named WebPartZoneAppVB and learn how to use the `WebPartZone` control in the application. You can find the code of WebPartZoneAppVB application in the Code\ASP.NET\Chapter 20\WebPartZoneAppVB folder on the CD. Listing 20.3 shows the complete code for the `Default.aspx` page of the WebPartZoneAppVB website:

Listing 20.3: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>WebPartZone control Example</title>
  <link href="Stylesheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
      <div>
      <div id="header">
      </div>
      <div id="sidebar">
      <div id="nav">
      &nbsp;
      </div>
      </div>
      <div id="content">
      <p>
      &nbsp;
```

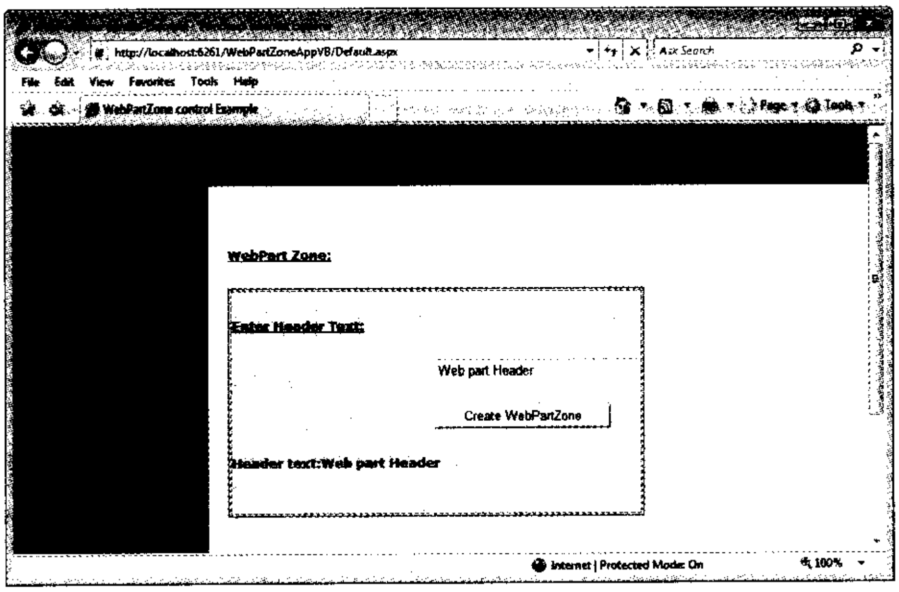
```
            </p>
            <div class="itemContent">
            <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-
            Underline="True" ForeColor="Navy" Text="WebPart Zone:">
            </asp:Label>
            <br />
            <br />
            <asp:Panel ID="Panel1" runat="server" BackColor="#FFE0C0"
            BorderColor="#FF8000" BorderStyle="Double"
            Height="226px" Width="422px">
            <br />
            <asp:Label ID="Label2" runat="server" Font-Bold="True" Font-
            Underline="True" ForeColor="Navy"
            Text="Enter Header Text:">
            </asp:Label>
            <br />
            <br />
            <br />
            <asp:TextBox ID="TextBox1" runat="server"
            Width="230px">
            </asp:TextBox>
            <br />
            <br />
            <asp:Button ID="Button1" runat="server" BackColor="#FFFFC0"
            Text="Create WebPartZone" />
            <br />
            <br />
            <asp:Label ID="Label3" runat="server" Font-Bold="True">
            </asp:Label>
            </asp:Panel>
            </div>
            </div>
            <div id="footer">
            <p class="left">
            All content copyright &copy; Kogent Solutions Inc.
            </p>
            </div>
            </div>
      </form>
    </body>

    </html>
```

Listing 20.4 shows the complete code for the code-behind file of the `WebPartZoneAppVB` website:

Listing 20.4: Showing the Code for the Code-Behind File:

```
  Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles Button1.Click
          Dim wpm As New WebPartManager
          Dim wpz As New WebPartZone
          Dim str As String
          wpz.MenuLabelText = TextBox1.Text
          str = wpz.MenuLabelText
          Label3.Text = "Header text:" + str
    End Sub
  End Class
```

Now, run the application, enter Web Part Header in the Enter Header Text TextBox, and click the Create WebPartZone button. You will find the output as shown in Figure 20. 5:



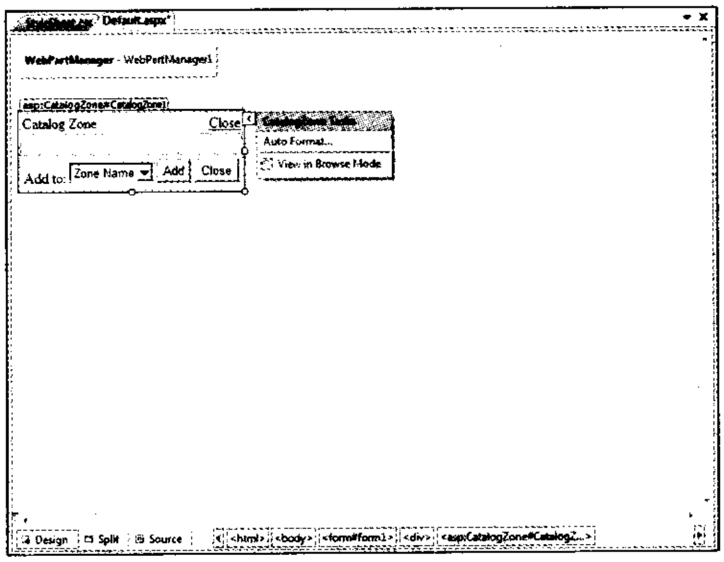
http://localhost:6261/WebPartZoneAppVB/Default.aspx
File Edit View Favorites Tools Help
WebPartZone control Example
WebPart Zone:
Enter Header Text:
Web part Header
Create WebPartZone
Header text:Web part Header
Internet | Protected Mode: On
100%

Figure 20.5: Output of the WebPartZone Control Example

The CatalogZone Control

You can personalize a website by adding the catalog functionality. The catalog functionality allows you to manage Web Parts controls efficiently. You can create a catalog-enabled Web page by adding a `CatalogZone` control to it.

A `CatalogZone` control helps to create a catalog of Web Parts controls. This allows a user to add or update the controls easily on a Web page. To add a `CatalogZone` to a Web page, simply drag and drop the `CatalogZone` control from the Toolbox to the Design view of an `.aspx` page. Figure 20.6 shows the `CatalogZone` control in the Design view:



Default.aspx*
WebPartManager - WebPartManager1
asp:CatalogZone#CatalogZone1
Catalog Zone
Close
Add to:
Zone Name
Add
Close
Auto Format...
View in Browse Mode
Design
Split
Source
<html> <body> <form#form1> <div> <asp:CatalogZone#CatalogZ...>

Figure 20.6: The CatalogZone Control

You can also add the `CatalogZone` control by inserting the following code in the `<form>` element of the `.aspx` page:

```
<%@ Page Language="VB"%>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>CatalogZone Example</title>
</head>
<body>
  <form id="form1" runat="server">
  <asp:WebPartManager ID="WebPartManager1" runat="server">
        </asp:WebPartManager>
  <asp:CatalogZone ID="CatalogZone1" runat="server">
        </asp:CatalogZone>
  </form>
</body>
</html>
```

When you add a `CatalogZone` control to a Web page, a new mode, called the `Catalog` mode, appears in the Select mode list on the Web page. The `CatalogZone` control is displayed when the Web page is in `Catalog` mode allowing users to add controls on the Web page contained in it. The `CatalogZone` control is a template control. Therefore, the control needs a template section, which is represented as the `<ZoneTemplate>` element inside the `CatalogZone` section. Inside the template section, you can place several `CatalogPart` controls, where every `CatalogPart` control works as a container control for server controls. These server controls can be added on a Web page. The `CatalogZone` control is an object of the `CatalogZone` class. Remember that you can view a `CatalogZone` control on a Web page only after changing the display mode of the Web page to the `CatalogDisplayMode` mode.

The `CatalogZone` control is created by using the `CatalogZone` class of the .NET Framework class library. This class is available in the `System.Web.UI.WebControls.WebParts` namespace.

The inheritance hierarchy of the `CatalogZone` class is:

```
System.Object
  System.Web.UI.Control
      System.Web.UI.WebControls.WebControl
         System.Web.UI.WebControls.CompositeControl
            System.Web.UI.WebControls.WebParts.WebZone
               System.Web.UI.WebControls.WebParts.ToolZone
                  System.Web.UI.WebControls.WebParts.CatalogZoneBase
                     System.Web.UI.WebControls.WebParts.CatalogZone
```

Noteworthy methods of the `CatalogZone` class are listed in Table 20.7:

Table 20.7: Noteworthy Methods of the CatalogZone Class

Method	Description
`CreateCatalogPartChrome`	Creates the `CatalogPartChrome` object used to render the UI elements of `CatalogPart` controls in a zone.
`CreateCatalogParts`	Creates instances of all `CatalogPart` controls declared in a CatalogZone control.
`InvalidateCatalogParts`	Deletes all `CatalogPart` controls associated with a `CatalogZoneBase` zone.
`LoadPostData`	Determines the Web Parts controls that are selected in the `CatalogZone` control when a Web page is posted back to a Web server. The selected Web Parts controls are added to the Web page.
`RenderCatalogPartLinks`	Creates the links to every `CatalogPart` control in a `CatalogZoneBase` zone.

Noteworthy properties of the `CatalogZone` class are listed in Table 20.8:

Table 20.8: Noteworthy Properties of the CatalogZone Class

Property	Description
`SelectedCatalogPartID`	Retrieves or initializes a string to identify the currently selected `CatalogPart` control in a zone
`SelectedPartLinkStyle`	Retrieves an object that contains style attributes for the currently selected `CatalogPart` control in a zone
`SelectTargetZoneText`	Retrieves or sets the text next to a server control in the catalog UI. This allows a user to select the zone for adding selected controls
`ShowCatalogIcons`	Retrieves or sets a value to indicate whether server controls in a `CatalogZone` control display their associated icons or not

Creating the CatalogZone Control in Code

As discussed in this chapter, the `CatalogZone` control is used to create a catalog of Web Parts controls. This control also allows the user to add or update the controls on a Web page. Similar to other Web Parts controls, you can also create the `CatalogZone` control by using classes. You can set most of the properties of the `CatalogZone` control by declaring the `CatalogZone` class.

To understand the use of the `CatalogZone` control, let's use it in an application. For this, let's create an application named `CatalogZoneAppVB`. You can find the code of `CatalogZoneAppVB` application in the Code\ASP.NET\Chapter 20\`CatalogZoneAppVB` folder on the CD. In this application, we add three label controls to display the `ZoneSelect`, `Mode` and `EmptyZoneText` controls on the click event of the `Button` control. Listing 20.5 shows the complete code for the `Default.aspx` page of the `CatalogZoneAppVB` website:

Listing 20.5: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>CatalogZone Example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
       <div>
       <div id="header">

       </div>
       <div id="sidebar">
       <div id="nav">
       &nbsp;
       </div>
       </div>
       <div id="content">
       <p>
       &nbsp;</p>
       <div class="itemContent">
       <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-
       Underline="True" ForeColor="Navy"
       Text="WebPart CatalogZone:"></asp:Label>
       <br />
```

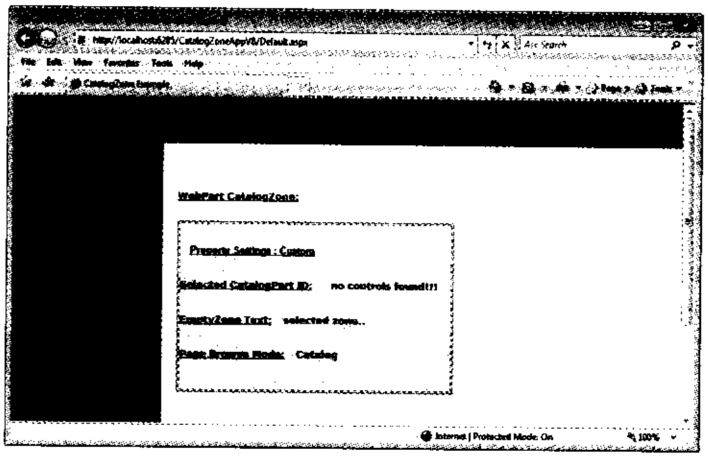
```
            <br />
            <asp:Panel ID="Panel1" runat="server" BackColor="#FFE0C0"
            BorderColor="#FF8000" BorderStyle="Double"
            Height=" 221px " Width="364px">
            <br />
            &nbsp;<asp:Button ID="Button1" runat="server" BackColor="#FFE0C0"
            BorderStyle="None"
            Font-Bold="True" Font-Underline="True" ForeColor="Blue"
            Text="Property Settings : Custom" Width="174px" /><br />
            <br />
            <asp:Label ID="Label3" runat="server" Font-Bold="True" Font-
            Underline="True" ForeColor="Navy"
            Text="Selected CatalogPart ID:"></asp:Label>&nbsp;&nbsp;
            <asp:Label ID="Label4" runat="server" Font-Bold="True" Text="Label"
            Visible="False"></asp:Label><br />
            <br />
            <asp:Label ID="Label5" runat="server" Font-Bold="True" Font-
            Underline="True" ForeColor="Navy"
            Text="EmptyZone Text:"></asp:Label>&nbsp;
            <asp:Label ID="Label6" runat="server" Font-Bold="True" Text="Label"
            Visible="False"></asp:Label><br />
            <br />
            <asp:Label ID="Label7" runat="server" Font-Bold="True" Font-
            Underline="True" ForeColor="Navy"
            Text="Page Browse Mode:"></asp:Label>&nbsp;
            <asp:Label ID="Label8" runat="server" Font-Bold="True" Text="Label"
            Visible="False"></asp:Label><br />
            <br />
            </asp:Panel>
            </div>
            </div>
            <div id="footer">
            <p class="left">
            All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
            </div>
    </form>
  </body>
  </html>
```

Listing 20.6 shows the complete code for the code-behind file of the `CatalogZoneAppVB` website:

Listing 20.6: Showing the Code for the Code-Behind File:

```
Partial Class _Default
   Inherits System.Web.UI.Page
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
   System.EventArgs) Handles Button1.Click
         Dim wpm As New WebPartManager
         Dim cz As New CatalogZone
         Dim str As String
         cz.EmptyZoneText = "no controls found!!!"
         Label4.Text = cz.EmptyZoneText
         cz.SelectedCatalogPartID = "selected zone.."
         Label6.Text = cz.SelectedCatalogPartID
         str = WebPartManager.CatalogDisplayMode.Name
         Label8.Text = str
         Label4.Visible = True
         Label6.Visible = True
         Label8.Visible = True
   End Sub
End Class
```

Now, run the application and click the Property Settings: Custom link. You will find the output as shown in Figure 20.7:



WebPart CatalogZone:

Property Settings : Custom

Selected CatalogPart ID: no controls found!!!

EmptyZone Text: selected zone..

Page Browse Mode: Catalog

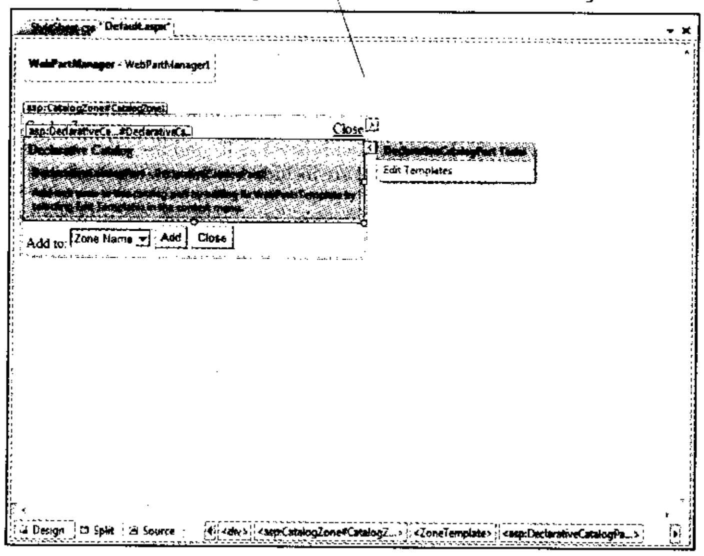
Figure 20.7: Output of the CatalogZone Control Example

The DeclarativeCatalogPart Control

The `DeclarativeCatalogPart` control is one of the three controls (the other controls are `PageCatalogPart` and `ImportCatalogPart`) used together with the `CatalogZone` control. The `DeclarativeCatalogPart` control is used to modify the functionality of a Web page. This control displays a list of Web Parts controls and other server controls, when the website is in the `CatalogDisplayMode` mode. A user can add controls from this catalog onto the Web page.

The `DeclarativeCatalogPart` control is an object of the `DeclarativeCatalogPart` class. You can add a `DeclarativeCatalogPart` control to an `.aspx` page by dragging and dropping the control from the Toolbox to the `CatalogZone` control in the Design view of an `.aspx` page. While adding a `DeclarativeCatalogPart` control to a Web page, ensure that the Web page contains the `CatalogZone` control.

Figure 20.8 shows the `DeclartiveCatalogPart` control inside the `CatalogZone` control in the Design view:



Default.aspx*
WebPartManager - WebPartManager1
asp:CatalogZone#CatalogZone1
asp:DeclarativeCa...#DeclarativeCa...
Close
Declarative Catalog
Edit Templates
Add to: Zone Name Add Close
Design Split Source
<div> <asp:CatalogZone#CatalogZ...> <ZoneTemplate> <asp:DeclarativeCatalogPa...>

Figure 20.8: The DeclarativeCatalogPart Control

You can also add the `DeclarativeCatalogPart` control by using the following code in the `<form>` element of the `.aspx` page:

```
<asp:CatalogZone ID="CatalogZone1" runat="server">

  <ZoneTemplate>
  <asp:DeclarativeCatalogPart   ID="DeclarativeCatalogPartI"
  runat="server">
  </asp:DeclarativeCatalogPart>
  </ZoneTemplate>

</asp:CatalogZone>
```

The `DeclarativeCatalogPart` control is created by using the `DeclarativeCatalogPart` class of the .NET Framework class library. This class is available in the `System.Web.UI.WebControls.WebParts`namespace.

The inheritance hierarchy of this class is:

```
System.Object
  System.Web.UI.Control
     System.Web.UI.WebControls.WebControl
        System.Web.UI.WebControls.Panel
           System.Web.UI.WebControls.WebParts.Part
              System.Web.UI.WebControls.WebParts.CatalogPart
                 System.Web.UI.WebControls.WebParts.DeclarativeCatalogPart
```

Now, let's identify the syntax for declaring the `DeclarativeCatalogPart` control by using the `DeclarativeCatalogPart` class.

The syntax for declaring the `DeclarativeCatalogPart` control is:

```
Dim control_ DeclarativeCatalogPart as DeclarativeCatalogPart
```

You can create a user control made of one or more server controls and add it to the `DeclarativeCatalogPart` control. To do so, you have to set the `WebPartsListUserControlPath` property of the `DeclarativeCatalogPart` class to the path of the user control. A user control is preferred to adding server controls, because you can display the catalog of Web Parts controls on multiple pages of a website by using the same user control. When the Web Parts controls in a user control are modified, all the catalogs associated with the user control are updated automatically by the `DeclarativeCatalogPart` control.

Noteworthy methods of the `DeclarativeCatalogPart` class are listed in Table 20.9:

Table 20.9: Noteworthy Methods of the DeclarativeCatalogPart Class

Method	Description
`ApplyStyle`	Applies the specified style to the style elements of a WebPart control
`GetAvailableWebpartsDescriptions`	Provides details (such as ID and the value of other properties) of a WebPart control that are included in the catalog displayed on a Web page
`CopyBaseAttributes`	Copies the `AccessKey`, `Enabled`, `ToolTip`, `TabIndex`, and `Attributes` properties from the specified server control to the server control from which this method is called. These properties and their values are specified as a parameter
`RenderBeginTag`	Provides the beginning tag for a `Panel` control to add a panel on which server controls can be added
`RenderEndTag`	Provides the ending tag for a `Panel` control indicating the end of the panel
`GetWebpart`	Provides a reference to a Web Parts control based on the description passed to the `GetWebpart` method
`MergeStyle`	Adds the style elements passed as parameters to a Web Parts control

Noteworthy properties of the `DeclarativeCatalogPart` class are listed in Table 20.10:

Table 20.10: Noteworthy Properties of the DeclarativeCatalogPart Class

[illegible]	[illegible]
`AccessKey`	Represents a keyboard shortcut key with which you can access a `DeclarativeCalatlogPart` control
`BackColor`	Specifies or determines the background color of a WebPart control
`ChromeType`	Determines or specifies the border of a WebPart control
`ChromeState`	Determines whether a WebPart control is in the minimized or normal state
`ControlStyle`	Determines the style of a WebPart control
`ControlStyleCreated`	Determines whether a style has been indicated for a WebPart control by using the `ControlStyle` property
`HasAttributes`	Determines whether a WebPart control has attributes set or not
`WebPartsListUserControlPath`	Determines or specifies the path of a user control that contains WebPart controls for a catalog
`WebPartsTemplate`	Specifies or determines a template reference for containing WebPart controls of a catalog
`BorderWidth`	Specifies or determines the width of the border of a WebPart control
`CssClass`	Specifies or determines the Cascading Style Sheet (`CSS`) class for a WebPart control
`Font`	Determines the font of a WebPart control
`ForeColor`	Specifies the foreground color of a WebPart control
`GroupingText`	Specifies a caption for the controls contained in a control
`Height`	Specifies the height of a WebPart control
`HorizontalAlign`	Determines the horizontal alignment of the content in a `Panel` control
`ScrollBars`	Determines and specifies the setting for the scroll bars, such as their appearance and position
`TabIndex`	Determines the tab index of a WebPart control
`ToolTip`	Specifies the text that is displayed when the mouse pointer is placed on a WebPart control
`Width`	Specifies the width of a WebPart control
`Wrap`	Determines whether the text displayed in a control should wrap or not
`BackImageUrl`	Specifies the URL of the image displayed in a control

NOTE

These properties are not used by the Web Parts control set when rendering a DeclarativeCatalogPart control. The property is overridden only for the purpose of preventing it from appearing in Microsoft Visual Studio 2008 designer tools.

The PageCatalogPart Control

The `PageCatalogPart` control, along with the `DeclarativeCatalogPart` and `PageCatalogPart` controls, is used with the `CatalogZone` control to add a specific functionality to the catalog section of a Web page. The `PageCatalogPart` control allows users to restore previously deleted WebPart controls of a Web page. This is the only way users have to restore the deleted Web Parts controls. The `PageCatalogPart` control adds a list of check boxes to the Catalog zone controls corresponding to each deleted Web Parts control. To restore these controls on the Web page, users simply need to select the required check box and a Web part zone where the Web Parts control is to be added, and click the Add button.

To add the `PageCatalogPart` control to the `CatalogZone` control, drag and drop the `PageCatalogPart` control from the Toolbox to the `CatalogZone` control in the Design view of an `.aspx` page. Figure 20.9 shows the `PageCatalogPart` control inside the `CatalogZone` control in the Design view:

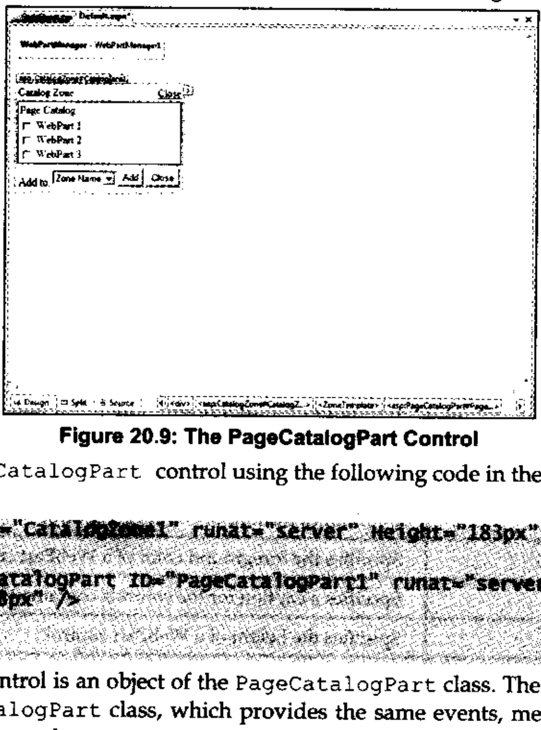


Figure 20.9: The PageCatalogPart Control

You can also add the `PageCatalogPart` control using the following code in the `<form>` element of the `.aspx` page:

```
<asp:CatalogZone ID="CatalogZone1" runat="server" Height="183px" Width="318px">
  <ZoneTemplate>
      <asp:PageCatalogPart ID="PageCatalogPart1" runat="server"
      Height="190px" />
  </ZoneTemplate>
</asp:CatalogZone>
```

The `PageCatalogPart` control is an object of the `PageCatalogPart` class. The `PageCatalogPart` control is associated to the `PageCatalogPart` class, which provides the same events, methods, and properties as those provided by the `CatalogZone` class.

The inheritance hierarchy of the `PageCatalogPart` control is:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.WebControls.WebControl
      System.Web.UI.WebControls.Panel
        System.Web.UI.WebControls.WebParts.Part
          System.Web.UI.WebControls.WebParts.CatalogPart
             System.Web.UI.WebControls.WebParts.PageCatalogPart
```

You can declare the `PageCatalogPart` class to add a `PageCatalogPart` control. The syntax to declare the `PageCatalogPart` class is:

```
Dim pagecatalogpart as PageCatalogPart
```

Creating the **PageCatalogPart** Control in Code

As learned earlier, the `PageCatalogPart` control is used to restore previously deleted Web Parts controls of a Web page. Now, let's learn how to create a `PageCatalogPart` control. For this, create an application named PageCatalogPartAppVB and add four label controls to display the `ChromeState`, `ChromeType`, `title` and `view-state` properties of the `PageCatalogPart` class, when a user clicks the `Button` control. You can find the code of PageCatalogPartAppVB application in the Code\ASP.NET\Chapter 20\PageCatalogPartAppVB folder on the CD. Listing 20.7 shows the complete code for the `Default.aspx` page of `PageCatalogPartAppVB` website:

Listing 20.7: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>PageCatalogPart Control Example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
        <div>
        <div id="header">

        </div>
        <div id="sidebar">
        <div id="nav">
        &nbsp;
        </div>
        </div>
        <div id="content">
        <p>
        &nbsp;</p>
        <div class="itemContent">
        <br />
        &nbsp;<asp:Label ID="Label1" runat="server" Font-Bold="True" Font-
        Underline="True"
        ForeColor="#0000C0" Text="Page Catalog
        Webpart"></asp:Label>&nbsp;<br />
        <br />
        <asp:Button ID="Button1" runat="server" BackColor="#FFE0C0"
        BorderColor="#FF8000"
        ForeColor="#0000C0" Text="Click" /><br />
        <asp:Panel ID="Panel1" runat="server" BackColor="#FFE0C0"
        BorderColor="#FF8000" BorderStyle="Double"
        Height="185px" Width="288px">
        <asp:Label ID="Label5" runat="server" Font-Bold="True" Font-
        Underline="True" ForeColor="#0000C0"
        Text="Ch[illegible]<asp:Label
        ID="Label[illegible]"server"
        Font-Bold="True" Text="Label"></asp:Label>&nbsp;&nbsp;
        <br />
        <br />
        <asp:Label ID="Label7" runat="server" Font-Bold="True" Font-
        Underline="True" ForeColor="#0000C0"
        Text="Chrome Type:"></asp:Label>&nbsp; &nbsp; &nbsp;<asp:Label
        ID="Label3" runat="server"
        Font-Bold="True" Text="Label"></asp:Label>&nbsp; &nbsp;
        <br />
        <br />
        <asp:Label ID="Label8" runat="server" Font-Bold="True" Font-
        Underline="True" ForeColor="#0000C0"
        Text="Title:"></asp:Label>&nbsp; &nbsp; &nbsp;<asp:Label
        ID="Label4" runat="server"
        Font-Bold="True" Text="Label"></asp:Label>&nbsp; &nbsp;
        <br />
        <br />
        <asp:Label ID="Label9" runat="server" Font-Bold="True" Font-
        Underline="True" ForeColor="#0000C0"
        Text="Enable View State:"></asp:Label>&nbsp; &nbsp;<asp:Label
        ID="Label6" runat="server"
        Font-Bold="True" Text="Label"></asp:Label></asp:Panel>
```

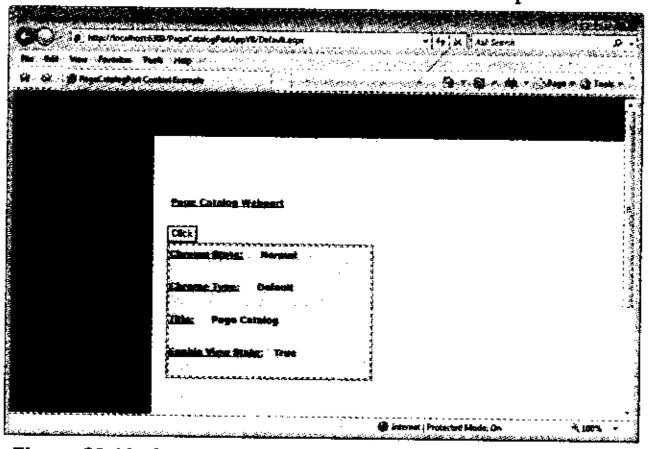
```
            <br />
            &nbsp; &nbsp;<br />
            <br />
            &nbsp; &nbsp;<br />
            &nbsp; &nbsp;
            <br />
            &nbsp; &nbsp; &nbsp; &nbsp;<br />
            <br />
            &nbsp; &nbsp; &nbsp;&nbsp;
            </div>
            </div>
            <div id="footer">
            <p class="left">
            All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
            </div>
    </form>
  </body>
  </html>
```

Listing 20.8 shows the complete code for the code-behind file of `PageCatalogPartAppVB` website:

Listing 20.8: Showing the Code for the Code-Behind File:

```
Partial Class _Default
  Inherits System.Web.UI.Page
  Dim wpm1 As New WebPartManager
  Dim pcp As New PageCatalogPart
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
  System.EventArgs) Handles Button1.Click
        Label2.Text = pcp.ChromeState .ToString
        Label3.Text = pcp.ChromeType .ToString
        Label4.Text = pcp.Title
        Label6.Text = pcp.EnableViewState
  End Sub
End Class
```

Now, run the application and click the Click Button. You will find the output as shown in Figure 20.10:



Page Catalog Webpart

Chrome State: Normal

Chrome Type: Default

Title: Page Catalog

Enable View State: True

Figure 20.10: Output of the PageCatalogPart Control Example

The ImportCatalogPart Control

The `ImportCatalogPart` control imports the description file of a Web Parts control or server control that is used as a Web Parts control. This enables the server control to be added to a Web page with pre-assigned settings. The description file enables users to share the settings of Web Parts controls. The

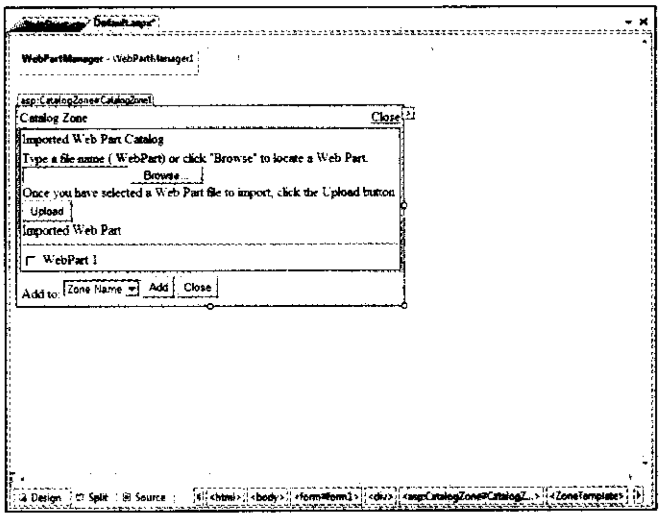
ImportCatalogPart control is the third control (the other two being DeclarativeCatalogPart and PageCatalogPart), which is used with the CatalogZone control to add a specific functionality to the catalog section of a Web page. This control provides you with options that allow you to browse and upload a description file required for importing a WebPart control.

The ImportCatalogPart control is an object of the ImportCatalogPart class.

NOTE

A description file is an XML file with a .WebPart extension. It contains property values, state data, and reference to the assembly or source file. If the properties of a Web Part control are customized, the description file contains values for these properties as well.

You can add an ImportCatalogPart control to a Web page by dragging and dropping it from the Toolbox to the CatalogZone control in the Design view of an .aspx page. Figure 20.11 shows the ImportCatalogPart control inside the CatalogZone control in the Design view:



WebPartManager - WebPartManager1

asp:CatalogZone#CatalogZone1

Catalog Zone Close

Imported Web Part Catalog

Type a file name (.WebPart) or click "Browse" to locate a Web Part.

Once you have selected a Web Part file to import, click the Upload button

Imported Web Part

WebPart 1

Add to: Zone Name Add Close

Figure 20.11: The ImportCatalogPart Control

You can also add the ImportCatalogPart control by using the following code in the <form> element of the .aspx page:

```
<asp:CatalogZone ID="CatalogZone1" runat="server">
   <ZoneTemplate>
      <asp:ImportCatalogPart ID="ImportCatalogPart1" runat="server" />
   </ZoneTemplate>
</asp:CatalogZone>
```

The ImportCatalogPart control is created by using the ImportCatalogPart class of the .NET Framework class library. This class is available in the System.Web.UI.WebControls.WebParts namespace.

The inheritance hierarchy of the ImportCatalogPart class is:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.WebControls.WebControl
      System.Web.UI.WebControls.Panel
        System.Web.UI.WebControls.WebParts.Part
          System.Web.UI.WebControls.WebParts.CatalogPart
            System.Web.UI.WebControls.WebParts.ImportCatalogPart
```

The syntax of the ImportCatalogPart control is:

```
Dim ImportCatalogPart1 as New ImportCatalogPart
```

You can use the various properties and methods of the `ImportCatalogPart` class to handle the `ImportCatalogPart` control. The properties of the `ImportCatalogPart` class allow you to specify the appearance and behavior of a zone on a Web page. The properties of the `ImportCatalogPart` control cannot be inherited by the derived classes of a base class. Noteworthy methods of the `ImportCatalogPart` class are listed in Table 20.11:

Table 20.11: Noteworthy Methods of the ImportCatalogPart Class

[illegible]	[illegible]
GetAvailableWebPartDescriptions	Retrieves a description of WebPart controls of a catalog
GetWebPart	Retrieves a WebPart control based on a description passed as a parameter

Noteworthy properties of the `ImportCatalogPart` class are listed in Table 20.12:

Table 20.12: Noteworthy Properties of the ImportCatalogPart Class

[illegible]	[illegible]
BrowseHelpText	Retrieves or sets a text message that informs users to browse to the location of a description file
ImportedPartLabelText	Retrieves or sets the text displayed after a user imports a description file to represent or describe an imported control in the catalog of imported controls
PartImportErrorLabelText	Retrieves or sets an error message that is displayed if an error occurs during the import process
UploadButtonText	Retrieves or sets the text for the `Button` control that starts the upload process of a description file
UploadHelpText	Retrieves or sets the text of a message that enables the user to upload a description file

The EditorZone Control

Till now, you must have got some idea of important Web Parts controls. Now, suppose you want to change the appearance of a Web page. To do so, you will obviously go to the properties of the Web page to make the changes at design time. However, instead of this traditional approach, you can use a Web Parts control called `EditorZone`. The `EditorZone` control is one of the primary editing controls used to change the appearance, format, and structure of Web pages containing Web Parts controls, at run time. In addition, you can use the `EditorZone` control to change the behavior and content of Web Parts controls. The `EditorZone` control is visible in the edit mode of the Web page. This control contains various `EditorPart` controls, such as `AppearanceEditorPart`, `BehaviorEditorPart`, which are used to customize the Web page. The `EditorZone` control is an object of the `EditorZone` class.

To add an `EditorZone` control to the Web page, drag and drop the `EditorZone` tool from the Toolbox to the Design view of an `.aspx` page. Figure 20.12 shows the `EditorZone` control in the Design view:

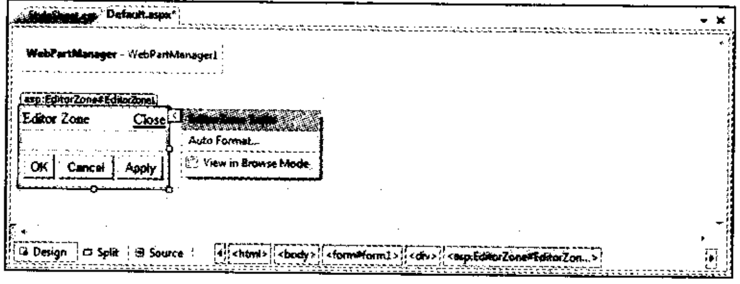


Figure 20.12: The EditorZone Control

You can also add an `EditorZone` control by using the following code in the `<form>` element of the.`aspx` page:

```
<asp:EditorZone ID="EditorZone1" runat="server">
</asp:EditorZone>
```

NOTE

You have to use the <ZoneTemplate> element inside the EditorZone control for the proper working of the EditorZone control.

The `EditorZone` control is created by using the `EditorZone` class of the .NET Framework class library. This class is available in the `System.Web.UI.WebControls.WebParts` namespace.

The inheritance hierarchy of the `EditorZone` class is:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.WebControls.WebControl
      System.Web.UI.WebControls.CompositeControl
        System.Web.UI.WebControls.WebParts.WebZone
          System.Web.UI.WebControls.WebParts.ToolZone
            System.Web.UI.WebControls.WebParts.EditorZoneBase
              System.Web.UI.WebControls.WebParts.EditorZone
```

The syntax to declare the `EditorZone` control is:

```
Dim editorA as EditorZone
```

Noteworthy methods of the `EditorZone` class are listed in Table 20.13:

Table 20.13: Noteworthy Methods of the EditorZone Class

`AddAttributesToRender`	Adds HTML attributes and styles provided to the specified `System.Web.UI.HtmlTextWriter` class
`CreateControlStyle`	Creates the style object, used by the `WebControl` class to execute all the properties related to the style
`CreateEditorPartChrome`	Retrieves a reference to an `EditorPartChrome` object, which provides the peripheral UI elements (such as a border, title, and icons) residing around an `EditorZoneBase` zone
`CreateEditorParts`	Creates all `EditorPart` controls specified in a zone template in page persistence format
`InvalidateEditorParts`	Removes all `EditorPart` controls associated with an `EditorZoneBase` zone
`RaisePostBackEvent`	Performs actions related to the verbs menu of an editor zone, or invokes an event that responds back to the server
`RenderBody`	Overrides the base method to leave the body area of a zone, which is obtained from the `EditorZoneBase` class
`RenderContents`	Provides the whole content of a zone control between the beginning and ending tags of the specified `HtmlTextWriter` class
`RenderFooter`	Provides verbs in the footer of a `ToolZone` control by overriding the base method
`RenderHeader`	Provides specialized rendering for the header area required by `ToolZone` controls, by overriding the base method
`Close`	Ends the process of modifying Web Parts controls and returns the browse mode of a Web page
`RenderVerb`	Provides a verb to a `ToolZone` control
`RenderVerbs`	Provides verbs to apply at the zone level

Noteworthy properties of the `EditorZone` class are listed in Table 20.14:

Table 20.14: Noteworthy Properties of the EditorZone Class

`ApplyVerb`	Retrieves a reference to a `WebPartVerb` object, which allows you to modify a Web Parts control in the edit mode
`AssociatedDisplayModes`	Retrieves the objects of the `WebPartDispalyMode` class activated when a `ToolZone` control is present on a Web page
`CancelVerb`	Retrieves a reference to a `WebPartVerb` object, which allows you to cancel the modification you made to a control in the edit mode
`EditorPartChrome`	Retrieves a reference to an instance of the `EditorPartChrome` class related to an `EditorZoneBase` zone
`EditorParts`	Retrieves all `EditorPart` controls contained in an `EditorZoneBase` zone
`EditUIStyle`	Retrieves the style attributes for the controls contained in a `ToolZone` control
`EmptyZoneText`	Sets or retrieves a message text, which appears when a zone is empty and has no controls
`EmptyZoneTextStyle`	Retrieves the style attributes for text in an empty zone
`ErrorStyle`	Retrieves the style attributes of an `EditorZone` control for displaying an error message if the control is not loaded or created
`ErrorText`	Retrieves or sets the text to display an error message in the editing UI
`FooterStyle`	Retrieves the style attributes for the footer area of a zone
`HeaderCloseVerb`	Retrieves a reference to a `WebPartVerb` object in the header of a `ToolZone` control to close the control
`HeaderStyle`	Retrieves the style attributes for the header area of a zone
`HeaderText`	Retrieves or sets the text for the header area of a zone
`HeaderVerbStyle`	Retrieves the style attributes of all the header verbs displayed in a `ToolZone` control
`InstructionText`	Retrieves or sets the text to instruct users how to use a `ToolZone` control
`InstructionTextStyle`	Retrieves the attributes for the text displayed at the top of a `ToolZone` control
`LabelStyle`	Retrieves the style attributes for the labels that appear in the editing controls of a `ToolZone` control
`OKVerb`	Retrieves a reference to a `WebPartVerb` object to apply editing changes to a control and hide the editing UI
`Padding`	Retrieves or sets the attributes related to cell padding in a table containing Web Parts controls
`PartChromeStyle`	Retrieves the style attributes for the borders of Web Parts controls contained in a zone
`PartChromeType`	Retrieves or sets the border type of Web Parts controls
`PartStyle`	Retrieves the style attributes for the border and content of Web Parts controls
`PartTitleStyle`	Retrieves the style attributes for the title bar of Web Parts controls
`VerbButtonType`	Retrieves or sets the button to represent the verbs in a zone
`VerbStyle`	Retrieves style characteristics for the UI of verbs of Web Parts controls
`ZoneTemplate`	Provides a template to add child controls to an `EditorZone` control

Creating the **EditorZone** Control in Code

As you know, the `EditorZone` control is used to change the appearance, format, and structure of Web pages containing Web Parts controls at runtime. The control can be created by simple drag and drop operations. In the following example, we show another way of creating this control, that is, the class declaration method. For this, create an application named `EditorZoneAppVB`. You can find the code of `EditorZoneAppVB` application in the Code\ASP.NET\Chapter 20\`EditorZoneAppVB` folder on the CD. Listing 20.9 shows the complete code for the `Default.aspx` page of `EditorZoneAppVB` website:

Listing 20.9: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!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>EditorZone control Example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
        <div>
        <div id="header">

        </div>
        <div id="sidebar">
        <div id="nav">
        &nbsp;
        </div>
        </div>
        <div id="content">
        <p>
        &nbsp;</p>
        <div Class="itemContent">
        &nbsp; &nbsp;
        <asp:Label ID="Label7" runat="server" Font-Bold="True" Font-
        Underline="True" ForeColor="#0000C0"
        Text="EditorZone Webpart"></asp:Label><br />
        <br />
        </div>
        <asp:Panel ID="Panel1" runat="server" BackColor="#FFFFC0"
        BorderStyle="Outset" Height="235px" Width="466px">
        &nbsp; &nbsp;<asp:Button ID="Button1" runat="server" BackColor="#FFC0C0"
        Font-Bold="True" Text="Default Values" Width="221px" />
        &nbsp;<br />
        &nbsp; &nbsp;<br />
        &nbsp;<asp:Label ID="Label4" runat="server" Font-Bold="True"
        Font-Underline="True"
        ForeColor="Navy" Text="Browse Mode:"></asp:Label>&nbsp;
        &nbsp;<asp:Label ID="Label1"
        runat="server" Font-Bold="True" Text="Label"></asp:Label><br />
        <br />
        <br />
        <asp:Label ID="Label5" runat="server" Font-Bold="True" Font-Underline="True"
        ForeColor="Navy"
        Text="Empty Zone Text:"></asp:Label>&nbsp; &nbsp;
        &nbsp;<asp:Label ID="Label2" runat="server"
        Font-Bold="True" Text="Label"></asp:Label>&nbsp; &nbsp;<br />
        <br />
        &nbsp; &nbsp;
```

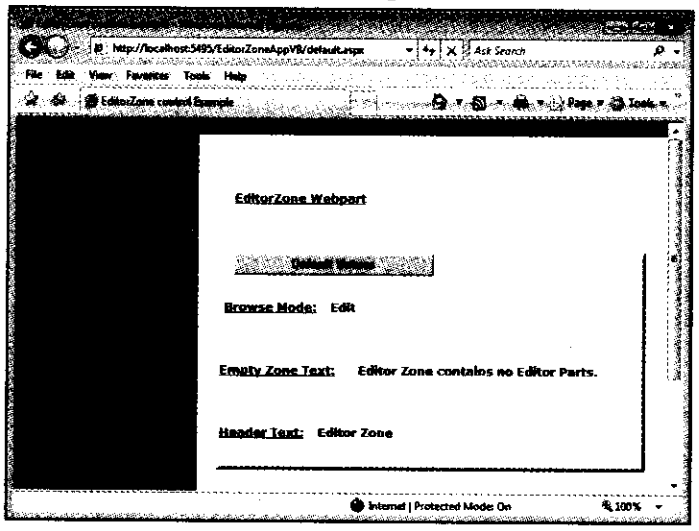
```
            <br />
            <asp:Label ID="Label6" runat="server" Font-Bold="True" Font-
            Underline="True" ForeColor="Navy"
            Text="Header Text:"></asp:Label>&nbsp; &nbsp;<asp:Label
            ID="Label3" runat="server"
            Font-Bold="True" Text="Label"></asp:Label><br />
            &nbsp; &nbsp;</asp:Panel>
            </div>
            <div id="footer">
            <p class="left">
            All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
            </div>
        </form>
    </body>
    </html>
```

Listing 20.10 shows the complete code for the code-behind file of `EditorZoneAppVB` website:

Listing 20.10: Showing the Code for the Code-Behind File:

```
Partial Class _Default
  Inherits System.Web.UI.Page
  Dim wpm As New WebPartManager
  Dim ez As New EditorZone
  Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
  System.EventArgs) Handles Button1.Click
        Dim str As String
        Dim str1 As String
        str = ez.EmptyZoneText
        Label2.Text = str
        str1 = ez.HeaderText
        Label3.Text = str1
  End Sub
  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles Me.Load
        Label1.Text = WebPartManager.EditDisplayMode.Name
  End Sub
End Class
```

The output of the `EditorZoneApp` website is shown in Figure 20.13:


http://localhost:5495/EditorZoneAppVB/default.aspx
Ask Search
File Edit View Favorites Tools Help
EditorZone control Example
Page
Tools
EditorZone Webpart
Default Values
Internet | Protected Mode: On
100%

Browse Mode: Edit

Empty Zone Text: Editor Zone contains no Editor Parts.

Header Text: Editor Zone

Figure 20.13: Output of the EditorZone Control Example

The LayoutEditorPart Control

You can use the `LayoutEditorPart` control to create a full-fledged layout editor for Web Parts controls. Web Parts controls can have their respective `LayoutEditorPart` controls associated with them. To change the layout of a Web Parts control, simply make the required changes in the `LayoutEditorPart` control associated with the Web Parts control.

The `LayoutEditorPart` control is visible only when a Web page is in the edit mode. In this mode, you can select and change the properties of a Web Part control. The `LayoutEditorPart` control offers the following three options to modify a Web Parts control:

- **ChromeState**—Gets or sets the current state of a Web Parts control, which can be normal or minimized.
- **Zone**—Sets the `WebPartZoneBase` zone, which consists of the Web Parts control you want to modify.
- **ZoneIndex**—Sets the index position of Web Parts controls that reside in a `WebPartZone` control. This option allows you to determine the specific position of a Web Parts control within the `WebPartZone` control.

You must use the `LayoutEditorPart` control in page persistence format. This format enables the `LayoutEditorPart` control to retain its property values between browser sessions. To do this; however, you need to declare the `<asp:LayoutEditorPart>` element inside the `<ZoneTemplate>` element.

To add a `LayoutEditorPart` control to a Web page, drag and drop a `LayoutEditorPart` control from the Toolbox to the Design view of an `.aspx` page. Figure 20.14 shows the `LayoutEditorPart` control inside an `EditorZone` control in the Design view:

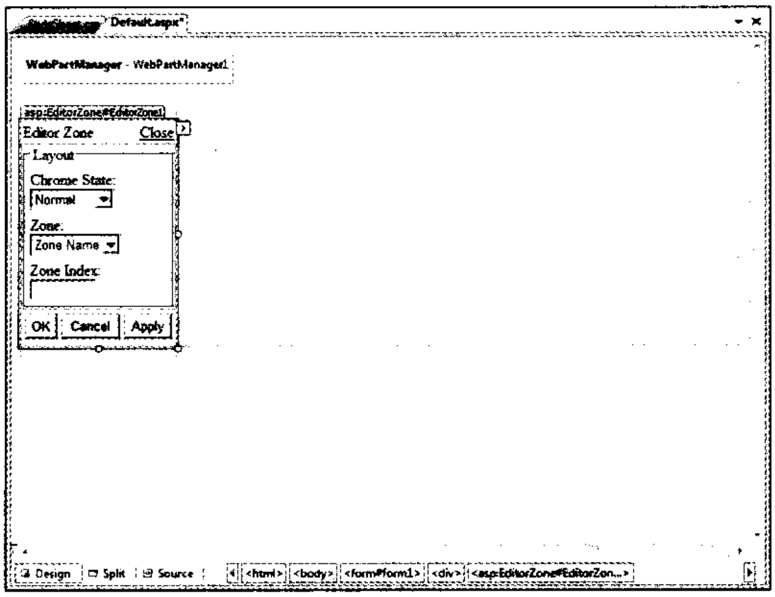


Figure 20.14: The LayoutEditorPart Control

You can also add the `LayoutEditorPart` control by using the following code in the `<form>` element of the `.aspx` page:

```
<asp:EditorZone ID="EditorZone1" runat="server">
  <ZoneTemplate>
      <asp:LayoutEditorPart ID="LayoutEditorPart1" runat="server" />
  </ZoneTemplate>
</asp:EditorZone>
```

The `LayoutEditorPart` control is an object of the `LayoutEditorPart` class.

NOTE

If you place LayoutEditorPart controls in any other zone except the EditorZone control, the designer generates an error.

The `LayoutEditorPart` control is created by using the `LayoutEditorPart` class of the .NET Framework class library. The `LayoutEditorPart` class allows you to modify the layout of the associated Web Parts control. The `LayoutEditorPart` class is available in the `System.Web.UI.WebControls.WebParts` namespace. The inheritance hierarchy of the `LayoutEditorPart` class is:

```
System.Object
   System.Web.UI.Control
      System.Web.UI.WebControls.WebControl
         System.Web.UI.WebControls.Panel
            System.Web.UI.WebControls.WebParts.Part
               System.Web.UI.WebControls.WebParts.EditorPart
                  System.Web.UI.WebControls.WebParts.LayoutEditorPart
```

The `LayoutEditorPart` control is associated with the `LayoutEditorPart` class. This class employs various methods to perform tasks, such as applying changes and style properties to a Web Parts control and determining if two object instances are equal. Noteworthy methods of the `LayoutEditorPart` class are listed in Table 20.15:

Table 20.15: Noteworthy Methods of the LayoutEditorPart Class

[illegible]	[illegible]
`ApplyChanges`	Updates the associated WebPart control properties with the changes made in a `LayoutEditorPart` control
`SyncChanges`	Gets the property values from a WebPart control, which is then assigned to the associated `LayoutEditorPart` control

Now, let's take a look at the properties of the `LayoutEditorPart` class. This class is inherited from the `EditorPart` class. The `EditorPart` class acts as a base class that provides a set of properties and methods to the `LayoutEditorPart` class. The `LayoutEditorPart` class also inherits some of the properties from other classes, such as `Control`, `Part`, and `WebControl`. The various properties inherited by the `LayoutEditorPart` class are `Font`, `Enabled`, `Page`, and `Style`. Noteworthy properties of the `LayoutEditorPart` class are listed in Table 20.16:

Table 20.16: Noteworthy Propeties of the LayoutEditorPart Class

[illegible]	[illegible]
DefaultButton	Retrieves or specify the ID of default button included in a `Panel` control
Display	Retrieves a value that specifies whether the `LayoutEditorPart` control should be displayed when the Web Parts control associated with it is in the edit mode
Title	Retrieves or specify a string used as a title for the `LayoutEditorPart` control

The AppearanceEditorPart Control

The `AppearanceEditorPart` control allows users to edit the properties associated with the appearance of a Web Parts control on the Web page. The `AppearanceEditorPart` control resides in an `EditorZone` zone and is visible only when a Web Parts control is selected for editing. The following options are displayed on the `AppearanceEditorPart` control:

- **Title**—Specifies a string value, which appears as a title of the `AppearanceEditorPart` control
- **Height**—Specifies the height of the `AppearanceEditorPart` control
- **Width**—Specifies the width of the `AppearanceEditorPart` control
- **ChromeType**—Selects the title and border type (dotted, single line, double line, and so on) of the `AppearanceEditorPart` control by using the `DropDownList` control

- **Direction**—Selects the direction (horizontal or vertical) of content flow inside the AppearanceEditorPart control by using the DropDownList control
- **Hidden**—Hides or displays the AppearanceEditorPart control by using the CheckBox control

To add the AppearanceEditorPart control to the EditorZone control, drag and drop the control from the Toolbox to the EditorZone control in the Design view of an .aspx page. Figure 20.15 shows the AppearanceEditorPart control inside the EditorZone control in the Design view:

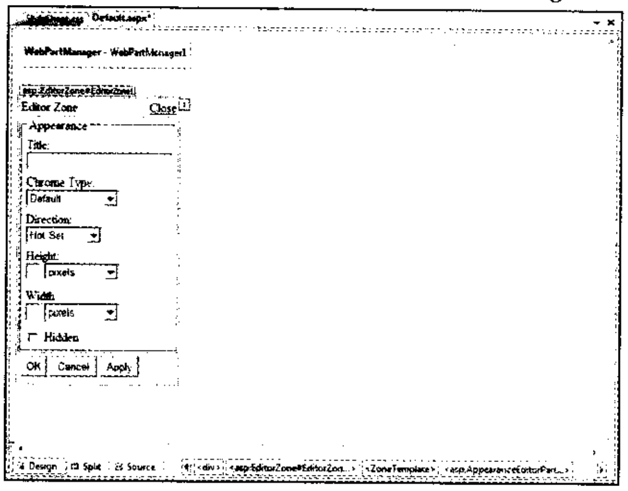


Figure 20.15: The AppearanceEditorPart Control

You can also add the AppearanceEditorPart control by using the following code in the <form> element of the .aspx page:

```
<asp:EditorZone ID="EditorZone1" runat="server">
    <ZoneTemplate>
        <asp:AppearanceEditorPart          ID="AppearanceEditorPart1"
        runat="server" />
    </ZoneTemplate>
</asp:EditorZone>
```

The AppearanceEditorPart control is created by using the AppearanceEditorPart class of the .NET Framework class library. The AppearanceEditorPart class provides an editor with control to modify the UI properties of the associated Web Parts control. The AppearanceEditorPart class is available in the System.Web.UI.WebControls.WebParts namespace. The inheritance hierarchy of the AppearanceEditorPart class is:

```
System.Object
   System.Web.UI.Control
      System.Web.UI.WebControls.WebControl
         System.Web.UI.WebControls.Panel
            System.Web.UI.WebControls.WebParts.Part
               System.Web.UI.WebControls.WebParts.EditorPart
                  System.Web.UI.WebControls.WebParts.AppearanceEditorPart
```

The AppearanceEditorPart class contains methods and properties that are used to work with the AppearanceEditorPart control at runtime. Noteworthy methods of AppearanceEditorPart class are listed in Table 20.17:

Table 20.17: Noteworthy Methods of the AppearanceEditorPart Class

[illegible]	[illegible]
ApplyChanges	Updates the associated WebPart control properties with the changes made in a AppearanceEditorPart control

Table 20.17: Noteworthy Methods of the AppearanceEditorPart Class

[illegible]	[illegible]
`SyncChanges`	Gets the property values from a WebPart control, which is then assigned to the associated `AppearanceEditorPart` control

Noteworthy properties of `AppearanceEditorPart` class are listed in Table 20.18:

Table 20.18: Noteworthy Propeties of the AppearanceEditorPart Class

[illegible]	[illegible]
`DefaultButton`	Retrieves or specify the ID of default button included in a `Panel` control
`Title`	Retrieves or specify a string used as a title for the `AppearanceEditorPart` control

The PropertyGridEditorPart Control

The `PropertyGridEditorPart` control provides a UI that allows users to edit the custom properties of a Web Parts control on a Web page. This control resides in the `EditorZone` control on the Web page. The `PropertyGridEditorPart` control is visible only when the Web page is in the edit mode or when the user selects a Web Parts control for editing.

The `PropertyGridEditorPart` control enables users to edit the properties that are marked with the `WebBrowsableattribute` attribute in the source code. This attribute specifies whether or not the property should be displayed in the `PropertyGridEditorPart` control. After specifying the editable properties, the `PropertyGridEditorPart` control creates an editing UI based on the type of properties to be edited.

To add the `PropertyGridEditorPart` control to a Web page, first drag and drop an `EditorZone` control from the Toolbox to the Design view of an `.aspx` page, and then drag and drop the `PropertyGridEditorPart` control tool from the Toolbox to the `EditorZone` control in the Design view. Figure 20.16 shows the `PropertyGridEditorPart` control inside the `EditorZone` control in the Design view:

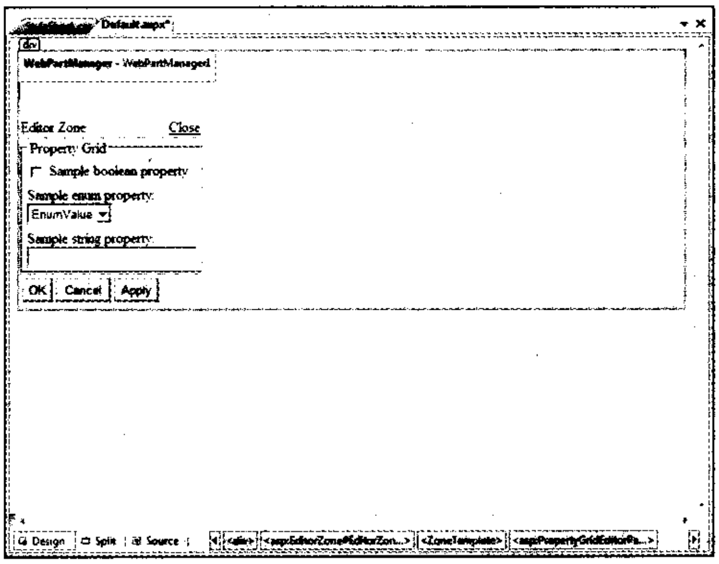


Figure 20.16: The PropertyGridEditorPart Control

You can also add the `PropertyGridEditorPart` control by using the following code in the `<form>` element of the `.aspx` page:

```
<asp:EditorZone ID="EditorZone1" runat="server">
  <ZoneTemplate>
      <asp:PropertyGridEditorPart ID="PropertyGridEditorPart1"
      runat="server" />
  </ZoneTemplate>
</asp:EditorZone>
```

The `PropertyGridEditorPart` control is an object of the `PropertyGridEditorPart` class. This control is created by using the `PropertyGridEditorPart` class of the .NET Framework class library. The `PropertyGridEditorPart` class enables users to modify the custom properties of the associated Web Parts or server control. The `PropertyGridEditorPart` class is available in the `System.Web.UI.WebControls.WebParts` namespace. The inheritance hierarchy of the `PropertyGridEditorPart` class is:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.WebControls.WebControl
      System.Web.UI.WebControls.Panel
        System.Web.UI.WebControls.WebParts.Part
          System.Web.UI.WebControls.WebParts.EditorPart
            System.Web.UI.WebControls.WebParts.PropertyGridEditorPart
```

The syntax of the `PropertyGridEditorPart` control is:

```
Dim propgridA as PropertyGridEditorPart
```

The `PropertyGridEditorPart` class resides in the `System.Web.UI.WebControls.WebParts` namespace and cannot be inherited. The `PropertyGridEditorPart` class consists of the `Title` and `Display` properties. The `Title` property is used to obtain or set a title text for a Web Parts control and the `Display` property is used to determine whether the control is displayed when the Web page is in the edit mode or not.

The two important methods in the `PropertyGridEditorPart` class are `ApplyChanges` and `SyncChanges`. The `ApplyChanges` method saves the value that a user sets on the `PropertyGridEditorPart` control to the corresponding properties of a Web Parts control. The `ApplyChanges` method is called when the user clicks the OK or Apply button on the editing UI. The `ApplyChanges` method returns true if the values from the `PropertyGridEditorPart` control are saved to a `WebPart` control and returns false if the values are not saved in the `WebPart` control. The `SyncChanges` method updates the values of the `PropertyGridEditorPart` control so that users can edit the values of these properties.

Creating the PropertyGridEditorPart Control in Code

As discussed in this chapter, the `PropertyGridEditorPart` control is used to edit the custom properties of a Web Parts control on a Web page. Now, let's learn how to create the `PropertyGridEditorPart` control. For this, we create an application named `PropertyGridEditorPartAppVB`. You can find the code of `PropertyGridEditorPartAppVB` application in the Code\ASP.NET\Chapter 20\`PropertyGridEditorPartAppVB` folder on the CD. You can set the properties of the `propertyGridEditorPart` control and use the methods of the `PropertyGridEditorPart` class after creating the control. Listing 20.11 shows the complete code for the `Default.aspx` page of `PropertyGridEditorPartAppVB` website:

Listing 20.11: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!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>PropertyGridEditorPart control Example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
```

```
          <div>
          <div id="header">

          </div>
          <div id="sidebar">
          <div id="nav">
          &nbsp;
          </div>
          </div>
          <div id="content">
          <p>
          &nbsp;</p>
          <div class="itemContent">
          &nbsp; &nbsp;<br />
          <asp:Label ID="Label7" runat="server" Font-Bold="True" Font-Underline="True"
          ForeColor="#0000C0" Text="Property Grid EditorPart"></asp:Label><br />
          &nbsp; &nbsp;<br />
          <asp:Button ID="Button1" runat="server" BackColor="#FFE0C0" BorderColor="#FF8000"
          Font-Bold="True" ForeColor="#0000C0" Text="CLICK" /><br />
          &nbsp; &nbsp;<asp:Panel ID="Panel1" runat="server" BackColor="#FFE0C0"
          BorderColor="#FF8000" BorderStyle="Double" Height="165px" Width="394px">
          <asp:Label ID="Label6" runat="server" Font-Bold="True" Font-Underline="True"
          ForeColor="#0000C0" Text="Display Text"></asp:Label>&nbsp; &nbsp;
          <asp:Label ID="Label1" runat="server" Font-Bold="True"
          Text="Label"></asp:Label><br />
          <br />
          <br />
          <asp:Label ID="Label5" runat="server" Font-Bold="True" Font-Underline="True"
          ForeColor="#0000C0" Text="Title Text"></asp:Label>&nbsp; &nbsp;
          <asp:Label ID="Label2" runat="server" Font-Bold="True"
          Text="Label"></asp:Label><br />
          <br />
          <br />
          <asp:Label ID="Label4" runat="server" Font-Bold="True" Font-Underline="True"
          ForeColor="#0000C0" Text="Tool Tip"></asp:Label>&nbsp; &nbsp;
          <asp:Label ID="Label3" runat="server" Font-Bold="True"
          Text="Label"></asp:Label></asp:Panel>
          &nbsp;<br />
          <br />
          <br />
          &nbsp;
          </div>
          <div id="footer">
          <p class="left">
          All content copyright &copy; Kogent Solutions Inc.</p>
          </div>
          </div>
          </div>
    </form>
</body>
</html>
```

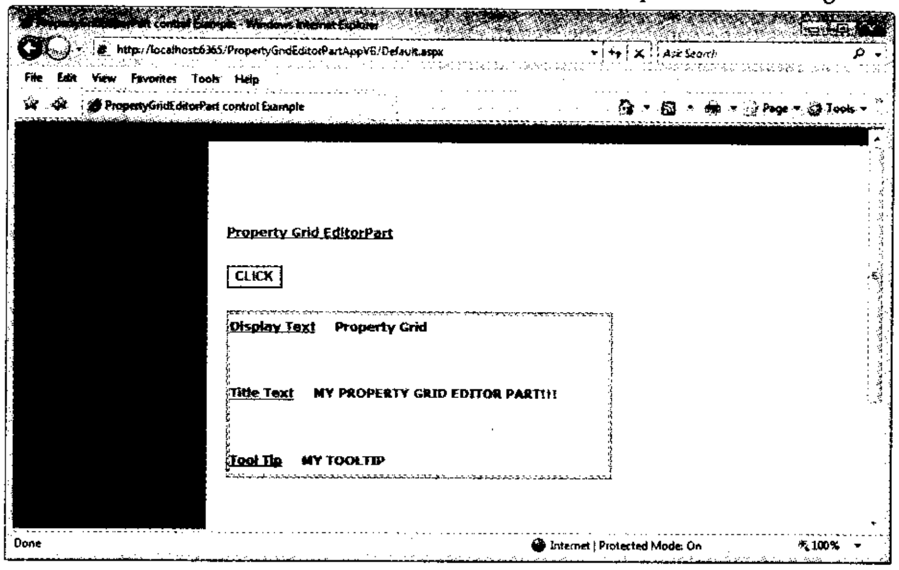
Listing 20.12 shows the complete code for the code-behind file of `PropertyGridEditorPartAppVB` website:

Listing 20.12: Showing the Code for the Code-Behind File

```
Partial Class _Default
    Inherits System.Web.UI.Page
    Dim wpm As New WebPartManager
    Dim propgrid As New PropertyGridEditorPart
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Button1.Click
          Label1.Text = propgrid.DisplayTitle
          propgrid.Title = "MY PROPERTY GRID EDITOR PART!!!"
          Label2.Text = propgrid.Title
          propgrid.ToolTip = "MY TOOLTIP"
```

```
            Label3.Text = propgrid.ToolTip
    End Sub
End Class
```

Now, run the application and click the Click Button. You will find the output as shown in Figure 20.17:



http://localhost6365/PropertyGridEditorPartAppVB/Default.aspx
File Edit View Favorites Tools Help
PropertyGridEditorPart control Example
Page Tools
Done
Internet | Protected Mode: On
100%

Property Grid EditorPart

CLICK

Display Text   Property Grid

Title Text   MY PROPERTY GRID EDITOR PART!!!

Tool Tip   MY TOOLTIP

Figure 20.17: Output of the PropertyGridEditorPart Control Example

The BehaviorEditorPart Control

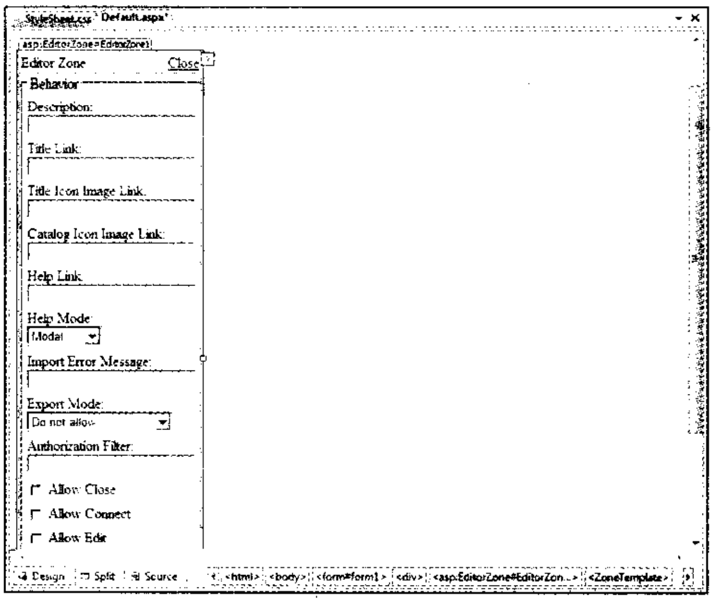
The `BehaviorEditorPart` control, as the name suggests, provides options that enable users to modify the behavior of Web Parts controls. This control usually remains hidden and is visible only when a Web page containing Web Parts controls is in edit mode. Similar to other `EditorPart` controls, the `BehaviorEditorPart` control is also contained inside the `EditorZone` control.

The various options provided by the `BehaviorEditorPart` control are:

- **Description**—Describes the functionality a Web Parts control in brief
- **TitleUrl**—Specifies the link for additional information about a Web Part control
- **TitleIconImageUrl**—Specifies an image, which appears in the title bar of a Web Part control
- **CatalogIconImageUrl**—Specifies an image, which appears as a representative of a Web Parts control in the catalog of controls
- **HelpUrl**—Specifies the link of the help file of a Web Parts control
- **HelpMode**—Specifies the type of UI to display the help content of a Web Part control
- **Hidden**—Hides or displays a Web Parts control
- **ImportErrorMessage**—Retrieves or specifies the error message, which appears when importing a Web Parts control
- **ExportMode**—Exports the properties of a Web Parts control
- **AuthorizationFilter**—Specifies a string value, which indicates whether you can add a Web Parts control to a Web page or not
- **AllowClose**—Specifies a value that allows a user to close a Web Parts control on a Web page
- **AllowEdit**—Specifies a value, which allows a user to modify a Web Parts control on Web page
- **AllowHide**—Specifies a value, which allows a user to hide a Web Parts control
- **AllowMinimize**—Specifies a value to minimize a Web Parts control

- **AllowZoneChange**—Specifies a value, which indicates whether a Web Parts control can be moved from one zone to another

To add the `BehaviorEditorPart` control to a Web page, drag and drop the control from the Toolbox to the `EditorZone` control in the Design view of an `.aspx` page. Figure 20.18 shows the `BehaviorEditorPart` control inside the `EditorZone` control in the Design view:



Description:

Title Link:

Title Icon Image Link:

Catalog Icon Image Link:

Help Link:

Help Mode: Modal

Import Error Message:

Export Mode: Do not allow

Authorization Filter:

☐ Allow Close
☐ Allow Connect
☐ Allow Edit

Figure 20.18: The BehaviorEditorPart Control

You can also add the `BehaviorEditorPart` control by using the following code in the `<zonetemplate>` element of the `EditorZone` control on the `.aspx` page:

```
<ZoneTemplate>
  <asp:BehaviorEditorPart ID="BehaviorEditorPart1" runat="server" />
</ZoneTemplate>
```

The `BehaviorEditorPart` control is an object of the `BehaviorEditorPart` class. The `BehaviorEditorPart` control is used for modifying UI properties, such as `Title`, `Link`, `Description`, `Help Link`, in a Web page. You can add the `BehaviorEditorPart` control by using the `BehaviorEditorPart` class, which belongs to the `System.Web` assembly. The namespace hierarchy of the `BehaviorEditorPart` class is:

```
System.Object
   System.Web.UI.Control
      System.Web.UI.WebControls.WebControl
         System.Web.UI.WebControls.Panel
            System.Web.UI.WebControls.WebParts.Part
               System.Web.UI.WebControls.WebParts.EditorPart
                  System.Web.UI.WebControls.WebParts.BehaviorEditorPart
```

The syntax of the `BehaviorEditorPart` control is:

```
Dim behaviorA as BehaviorEditorPart
```

The `EditorPart` class derives from the `BehaviorEditorPart` class, which is used to modify the properties of the associated Web Parts control. The `BehaviorEditorPart` class inherits various methods, properties, and events from the `Control` class, such as `ApplyStyle`, `Title`, `DataBinding`. Noteworthy properties of the `BehaviorEditorPart` class are listed in Table 20.19:

Table 20.19: Noteworthy Propeties of the BehaviorEditorPart Class

Method	Description
`DefaultButton`	Retrieves or specify the ID of default button included in a `Panel` control
`Display`	Retrieves a value that specifies whether the `BehaviorEditorPart` control should be displayed when the Web Parts control associated with it is in the edit mode
`Title`	Retrieves or specify a string used as a title for the `BehaviorEditorPart` control

The ConnectionsZone Control

You can dynamically connect Web Parts controls with each other that reside in the `WebPartZoneBase` zone with the help of the `ConnectionsZone` control. However, to do this, the Web Parts controls must be enabled for such type of dynamic connection. When you add the `ConnectionsZone` control to a Web page, a new mode, the connection mode, appears in the Select mode list of the Web page. The `ConnectionsZone` control is an object of the `ConnectionsZone` class.

NOTE

While working with the ConnectionsZone control and ConnectionsZone class, you must be aware of two terms, that is, Consumer and Provider. A Web Part control that receives data is called a Consumer and a Web Part control that sends data to other Web Part control(s) is called a Provider.

To add the `ConnectionsZone` control to the Web page, drag and drop the `ConnectionsZone` control from the Toolbox to the Design view of an `.aspx` page. The Design view of `ConnectionsZone` control is shown in Figure 20.19:

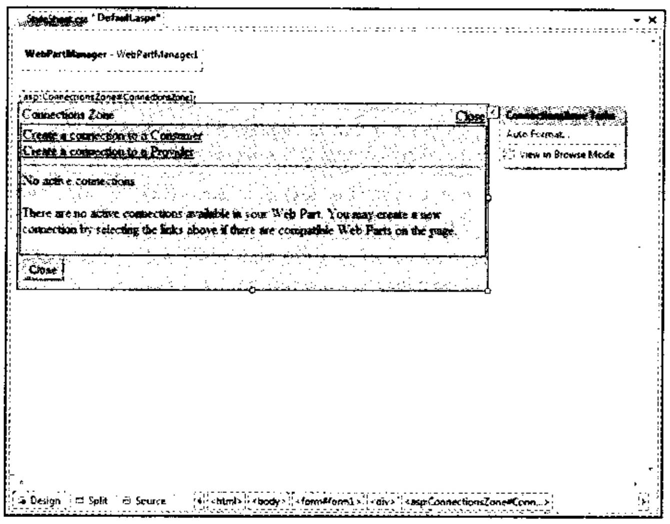

StyleSheet.css
Default.aspx
WebPartManager - WebPartManager1
asp:ConnectionsZone#ConnectionsZone1
Connections Zone
Close
Create a connection to a Consumer
Create a connection to a Provider
No active connections
There are no active connections available in your Web Part. You may create a new connection by selecting the links above if there are compatible Web Parts on the page.
Close
ConnectionsZone Tasks
Auto Format...
View in Browse Mode
Design
Split
Source
<html> <body> <form#form1> <div> <asp:ConnectionsZone#Conn...>

Figure 20.19: The ConnectionsZone Control

You can also add the `ConnectionsZone` control by using the following code in the `<form>` element of the `.aspx` page:

```
<asp:ConnectionsZone ID="ConnectionsZone1" runat="server">
</asp:ConnectionsZone>
```

The `ConnectionsZone` control is created by using the `ConnectionsZone` class of the .NET Framework class library. The `ConnectionsZone` control inherits from the `ToolZone` base class, which is available in the `System.Web.UI.WebControls.WebParts` namespace. The inheritance hierarchy of the `ConnectionsZone` class is:

```
System.Object
   System.Web.UI.Control
      System.Web.UI.WebControls.WebControl
         System.Web.UI.WebControls.CompositeControl
            System.Web.UI.WebControls.WebParts.WebZone
               System.Web.UI.WebControls.WebParts.ToolZone
                  System.Web.UI.WebControls.WebParts.ConnectionsZone
```

The syntax of the ConnectionsZone control is:

```
Dim ConnectionsZone1 as new ConnectionsZone
```

NOTE

To create a connection between Web Parts controls, a user needs to switch to the ConnectionDisplayMode connection display mode and select the connection verb from the Web Part Verb menu.

Noteworthy properties of the ConnectionsZone class are listed Table 20.20:

Table 20.20: Noteworthy Properties of the ConnectionsZone Class

Property	Description
ConfigureConnectionTitle	Gets or sets the title text of a connection UI created by a ConnectionsZone control
ConfigureVerb	Gets a reference to a WebPartVerb object. You can use this reference to open the configuration view in a connection UI
ConnectToConsumerInstructionText	Gets or sets the text displayed in the section of a connection UI where users select the consumer control to connect to the provider control
ConnectToConsumerText	Gets or sets the hyperlink text that users click to select a consumer control for a connection
ConnectToConsumerTitle	Gets or sets the title text of the section in a connection UI where users select the consumer control to connect with
ConnectToProviderInstructionText	Gets or sets the text that is displayed in the section of a connection UI where users select the provider control to connect to the consumer control
ConnectToProviderText	Gets or sets a hyperlink text that users click to open and then select a provider control for a connection
CancelVerb	Retrieves a reference to a WebPartVerb object that enables the user to cancel the process of establishing a connection
CloseVerb	Retrieves a reference to a WebPartVerb object that enables the user to close the connection UI
ConnectToProviderTitle	Gets or sets the title text of the section in the connection UI where users select the specific provider control to connect with
ConnectVerb	Gets a reference to a WebPartVerb object to allow two Web Parts controls to connect with each other
ConsumersInstructionText	Gets or sets the instructional text contained in the consumers section of a connection UI, when a connection already exists
ConsumersTitle	Gets or sets the title text of the consumers section present in a connection UI, when a connection already exists
DisconnectVerb	Gets a reference to a WebPartVerb object to enable the user to disconnect two connected WebPart controls
ExistingConnectionErrorMessage	Gets or sets the message string that is displayed in a connection UI when an error occurs in an existing connection
GetFromText	Gets or sets the text present before the name of provider in the connection UI. This

Table 20.20: Noteworthy Properties of the ConnectionsZone Class

Property	Description
	provider is used by a consumer to retrieve data
`GetText`	Gets or sets the text present before the name of consumer in the connection UI. This consumer retrieves the data from provider
`InstructionTitle`	Gets or sets the description text for the action performed on a consumer or provider control in a connection UI to manage existing connections
`NewConnectionErrorMessage`	Gets or sets a message string that is displayed in a connection UI when an error occurs while creating a new connection
`NoExistingConnectionInstructionText`	Gets or sets the text contained in the body of a connection UI when there is no existing connection associated with a WebPart control
`NoExistingConnectionTitle`	Gets or sets the title text of a connection UI when there is no existing connection associated with a WebPart control
`ProvidersInstructionText`	Gets or sets the instructional text contained in the providers section of a connection UI, when a connection already exists
`ProvidersTitle`	Gets or sets the title text above in the providers section of a connection UI, when a connection already exists
`SendText`	Gets or sets the text present before the name of provider in the connection UI. This provider sends data to consumer
`SendToText`	Gets or sets the text present in the connection UI before the name of the consumer to which a provider will send data

NOTE

All Web Parts controls reside inside the <form> element of an .aspx page. However, there are some Web Parts Controls that cannot be used alone and require some specific control to work with. For example, the LayoutEditorPart or BehaviorEditorPart control cannot be used without the EditorZone control in the <form> element.

Creating the ConnectionsZone Control in Code

We have already discussed how to use the `ConnectionsZone` class to create the `ConnectionsZone` control. Now, let's create the `ConnectionsZone` control with the help of the `ConnectionsZone` class and specify the various properties of the control. For this, we create an application named `ConnectionsZoneAppVB`. You can find the code of `ConnectionsZoneAppVB` application in the Code\ASP.NET\Chapter 20\`ConnectionsZoneAppVB` folder on the CD. Listing 20.13 shows the complete code for the `Default.aspx` page of `ConnectionsZoneAppVB` website:

Listing 20.13: Showing the Code for `Default.aspx` Page

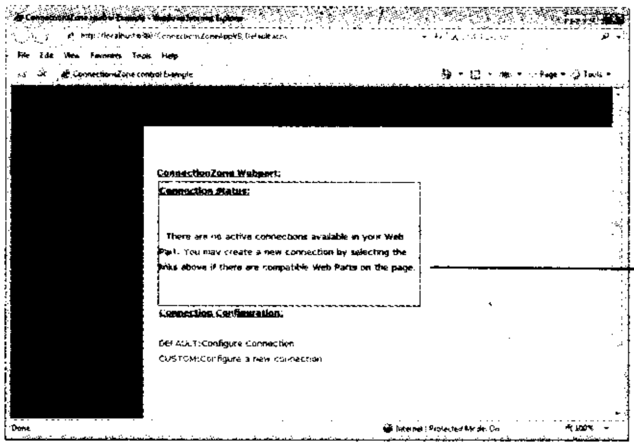
```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>ConnectionsZone control Example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
        <div id="header">

        </div>
        <div id="sidebar">
        <div id="nav">
        &nbsp;
        </div>
        </div>
        <div id="content">
        <p>
        &nbsp;</p>
        <div class="itemContent">
        <asp:Label ID="Label2" runat="server" Font-Bold="True" Font-Underline="True"
        ForeColor="Navy" Text="ConnectionZone Webpart:"></asp:Label>
        <asp:Panel ID="Panel1" runat="server" BackColor="#FFE0C0"
        BorderStyle="Ridge" Height="183px" Width="622px">
        <asp:Label ID="Label1" runat="server" Text="Connection Status:"
        Width="622px" Font-Bold="True" Font-Underline="True"
        ForeColor="Navy"></asp:Label><br />
        <br />
        &nbsp;<asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>
        <br />
        <br />
        <asp:Label ID="Label5" runat="server" Font-Bold="True" Font-Underline="True"
        ForeColor="Navy" Text="Connection Configuration:"></asp:Label><br />
        <br />
        <asp:Label ID="Label4" runat="server" Text="Label"></asp:Label>&nbsp;
        <br />
        <asp:Label ID="Label6" runat="server" Text="Label"></asp:Label></asp:Panel>
        </div>
        <div id="footer">
        <p class="left">
        All content copyright &copy; Kogent Solutions Inc.</p>
        </div>
        </div>
    </form>

  </body>

  </html>
```

Listing 20.14 shows the complete code for the code-behind file of `ConnectionsZoneAppVB` website:

Listing 20.14: Showing the Code for the Code-Behind File

```
Partial Class _Default
    Inherits System.Web.UI.Page
    Dim wpm As New WebPartManager
    Dim conn As New ConnectionsZone

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Me.Load
        Dim value As String
        value = conn.NoExistingConnectionInstructionText
        conn.NoExistingConnectionInstructionText = value
        Label3.Text = conn.NoExistingConnectionInstructionText
        Label4.Text = "DEFAULT:" & conn.ConfigureConnectionTitle
        conn.ConfigureConnectionTitle = _
        "Configure a new connection"
        Label6.Text = "CUSTOM:" & conn.ConfigureConnectionTitle
    End Sub

End Class
```

Figure 20.20 shows the output of the `ConnectionsZoneAppVB` website:


ConnectionZone Webpart:

Connection Status:

There are no active connections available in your Web Part. You may create a new connection by selecting the links above if there are compatible Web Parts on the page.

Connection Configuration:

DEFAULT:Configure Connection

CUSTOM:Configure a new connection

ConnectionZone control Example
File Edit View Favorites Tools Help
Done
Internet | Protected Mode: On
100%
splaying a message,
ı active connection
available.

Figure 20.20: The ConnectionZone Control in Action

Summary

This chapter described the Web Parts controls available with Visual Studio 2008. These Web Parts controls enable you to create a website that can be personalized at runtime directly through the browser. In addition to a thorough discussion of the Web Parts controls, which includes a detailed description of their methods, properties and events, this chapter also shows the use of these controls through sample Web applications.

In the next chapter, we describe how to work with databases in ASP.NET.

Quick Revise

Q1. **What are WebParts Controls?**

Ans: The WebParts controls are objects of WebParts on which the end-user can perform multiple functions, such as open, close, or minimize a WebPart, according to the requirement of the end-user. The WebParts controls enable you to add rich and personalized content and layout to your website and also edit that content and layout at runtime. A single Web page can have multiple WebParts controls. It is essential to remember that a Web page containing WebParts is called a WebParts page. There is no other difference between the simple Web page and WebParts page.

Q2. **Which WebPart control manages all the WebParts of an ASP.NET website? Enlist all the functions of this control.**

Ans: The WebPartManager control manages all other WebParts controls on the Web page. If you create a Web page by using the WebParts controls, then you need to ensure that the Web page contains a WebPartManager control before placing any other WebParts control on the Web page. The WebPartManager control performs the following tasks to control the functionality of the Web page:

- Tracks the controls that provide Web Part features to the web page including such as the WebPartZone control and ConnectionsZone control.
- Provide methods to insert and remove different types of controls on a Web page.
- Establishes, monitors, and manages connections between different types of controls.
- Enables you to drag controls to different locations on a Web page to customize its appearance.
- Provides various views that you can use to change and personalize the properties and behavior of controls.
- Enables you to toggle between different views of a Web page. Toggling between views simplifies certain tasks, such as modifying page layout and editing controls.
- Enables you to define and raise the lifecycle events associated with a WebParts control. The lifecycle events keep track of a control to determine when it is inserted, moved, connected, or removed from a Web page.

Q3. Which WebPart control is used to change the appearance of the Web page at runtime?

Ans: After designing a Web page, you may sometimes feel the need to modify its appearance, such as its color or style. For this purpose, you can use the EditorZone control, which is one of the primary controls used for editing the appearance, format, and structure of the WebParts pages. You can also use an EditorZone control to change the behavior and content of the WebParts controls. The EditorZone control contains various EditorPart controls, such as AppearanceEditorPart and BehaviorEditorPart, which are used to customize the WebParts pages.

Q4. Which WebPart control is used to create a region on a Web page?

Ans: The WebPartZone control is used to describe a region on a WebParts page. You can place other WebParts or standard controls such as Label, Textbox, and Image controls in the region created by the WebPartZone control. Once the controls are placed in the region they can be relocated, maximized, or minimized according to the requirements of the end-user.

Q5. How can you connect two WebParts controls to share data between them?

Ans: The ConnectionsZone control allows you to connect the WebParts controls dynamically with each other. However, the webParts you want to connect, using the ConnectionsZone control, must be enabled for such type of connections. The ConnectionsZone control provides a user interface to connect WebParts controls at run time. The WebPart that receives the data is called Consumer control and the WebPart that sends data is called Provider control.

Q6. How can you restore the previously deleted WebParts control on the Web page?

Ans: The PageCatalogPart control allows users to restore previously deleted WebParts controls of a Web page. This control is used along with the CatalogZone control and this is the only way to restore the deleted WebParts control. The PageCatalogPart control adds a list of check boxes to the CatalogZone control corresponding to each deleted WebParts control. The users simply need to select the required check box and a Web part zone, where the Web Parts control is to be added, and click the Add button to restore these controls on the Web page.

Q7. Why the ImportCatalogPart control is used?

Ans: The ImportCatalogPart control is used to import the description file for a `WebParts` control (or other ASP.NET server control that is used as a `WebParts` control). You can use this imported description file to add the control to a Web page with pre-assigned settings to a specified zone.

Q8. What are three editor controls used with the EditorZone control? Give their brief description.

Ans: The three editor controls used along with the EditorZone control are as follows:

- **AppearanceEditorPart**—Allows you to edit or change the characteristics of the WebParts controls displayed on a Web page.
- **PropertyGridEditorPart**—Allows the end-users to edit the custom properties of a WebParts control.
- **BehaviorEditorPart**—Enables end-users to modify the Behavior properties of the WebParts controls.

Q9. Why ProxyWebPartManager control is used?

Ans: The ProxyWebPartManager control is used to allow developers to specify static connections in a content page when the WebPartManager control has been already placed in the master page.

Q10. List any six Web parts controls used in ASP.NET 3.5?

Ans: The six Web Parts controls are as follows:

- WebPartManager
- ProxyWebPartManager
- WebPartZone
- LayoutEditorPart
- AppearanceEditorPart
- BehaviorEditorPart