Lecture 6

Handling the *back-button* problem

If the user clicks the *Backbutton* in the browser window to return to a previous ASP.NET form and then posts the form, the application's session state may not correspond to that form. In some cases, this can result in a propblem that we refer to as the ***back-button problem***.

Supposed the content of the user's shopping cart are stored in session state and displayed on the page. The user then deletes one of the two items, which changes the data in session state. At that point, the user changes his mind and clicks the *Back* button, which displays both items again, even though session state only includes one item.

If the user now proceeds to check out, the order is likely to show one item when the user thinks he has ordered two items. But that depends upon how the application is coded. In the worth cases, the back-button problem may cause an application to crash. In the case, clicking on the *Back* button won't cause a problem at all.

In general, there are two ways to handle the back-button problem. The first is to try to prevent pages from being saved in the browser's cache. Then, when the user clicks the *Back* button, the old page can't be retrived. ASP.NET provides four methods for doing that, but they don't work if the user's browser ignores the page cache settings that are sent with a response.

The second way is to code critical web forms so they detect when the user attempt to post a page that isn't current. To do that, a form can use timespamps or random numbers to track the users of pages. Because there's no relible way to prevent a page from being cached and retrieved via the *Back* button, you should use this second technique wherever possible.

How to disable browser page caching

You can limit the effect of the *Back* button by directing the browser to not cache pages that contain state-sensitive data. Then, when the user attempts to return to a page using the *Back* button, the warning message is displayed.

You can place the code to disable browser page caching in the event handler for the *Load* event of the page. This code should be executed each time the page is loaded.

Methods that set page caching options

Method	Description
Response.Cache.SetCacheability()	Indicates how the page should be cached. Specify *HttpCacheability.NoCache* to suppress caching.
Response.Cache.SetExpires()	Specifies when the cached page should expire. Specify *Now().AddSeconds(-1)* to mark the page as already expired.
Response.Cache.SetNoStore()	Specifies that the browser should not cache the page.
Response.AppendHeader()	Adds a header to the HTTP response object. Specifying "Pragma" for the key and *"no-cache"* for the value disables caching.

Code that disables caching for a page

```
Response.Cache.SetCacheability(HttpCacheability.NoCache);

Response.Cache.SetExpires(DateTime.Now.AddSeconds(-1));

Response.Cache.SetNoStore();

Response.AppendHeader("Pragma", 'no-cache');
```

Unfortunately, the technique described above doesn't ensure that the user's browser won't cache the page because the user's browser may ignore the page cache settings. Still, it's not a bad idea to add the code, descibed above, to the *Load_Page* event handler of any ASP.NET page that gets important data like customer or product information.

Using timestamps to avoid the back-button problem

We suggest the most relaible way to avoid the back-button problem. Below you can see the code for a web page that uses timestamps to determine whether the posted page is current.

A page that checks timestamps to avoid the back-button problem

```
public partial class Cart : System.Web.UI.Page
{
        private sortList cart;

        protected void Page_Load(object sender, EventArgs e);

        if (IsExpired())
                Response.Redirect("Expired.aspx");
        else
                this.SaveTimeStamps();

        this.GetCart();

        if (!IsPostBack)
                this.DisplayCart();

        private bool IsExpired()
        {
                if (Session["Cart_TimeStamp"] == null)
                        return false;
                else if (ViewState["TimeStamp"].ToString());
                        return false;
                else if (ViewState["TimeStamp"].To string() == Session["Cart_TimeStamp"].ToString())
                        return false;
```

```
            else
                return true;
        }

        private void SaveTimeStamps()
        {
            DateTime dim = DateTime.Now;

            ViewState.Add(“TimeStamp”, dtm);

            Session.Add(“Cart_TimeStamp”, dtm);
        }
        .
        .
}
```

The basic technique is to record a timestamp in two places when a page is posted: viwe state and session state. Then, the view state stamp is sent back to the browser and cached along with the rest of the information on the page, while the session state stamp is saved on the server.

Later, when the user posts a form for the second time, the *Page_Load* event handler calls a private method named *IsExpired*. This method retrieves the timestamps from view state and session state and compare them. If they are identical, the page is current and *IsExpired* retirns false. But if they are different, it indicates that the user posted that page that was retrived from the browser’s cache via the *Back* button. In that case, the *IsExpired* method returns true. Then, the *Page_Load* event handler redirects to a page named *Expired.aspx*, which in turn displays a message indicating that the page is out of data and can’t be posted. Notice, that before comparing the timestamp item in session state and view state, the *IsExpired* method checks that both of them items exist. If not, the method returns false so that current timestamps can be saved in both sessions state and view state.

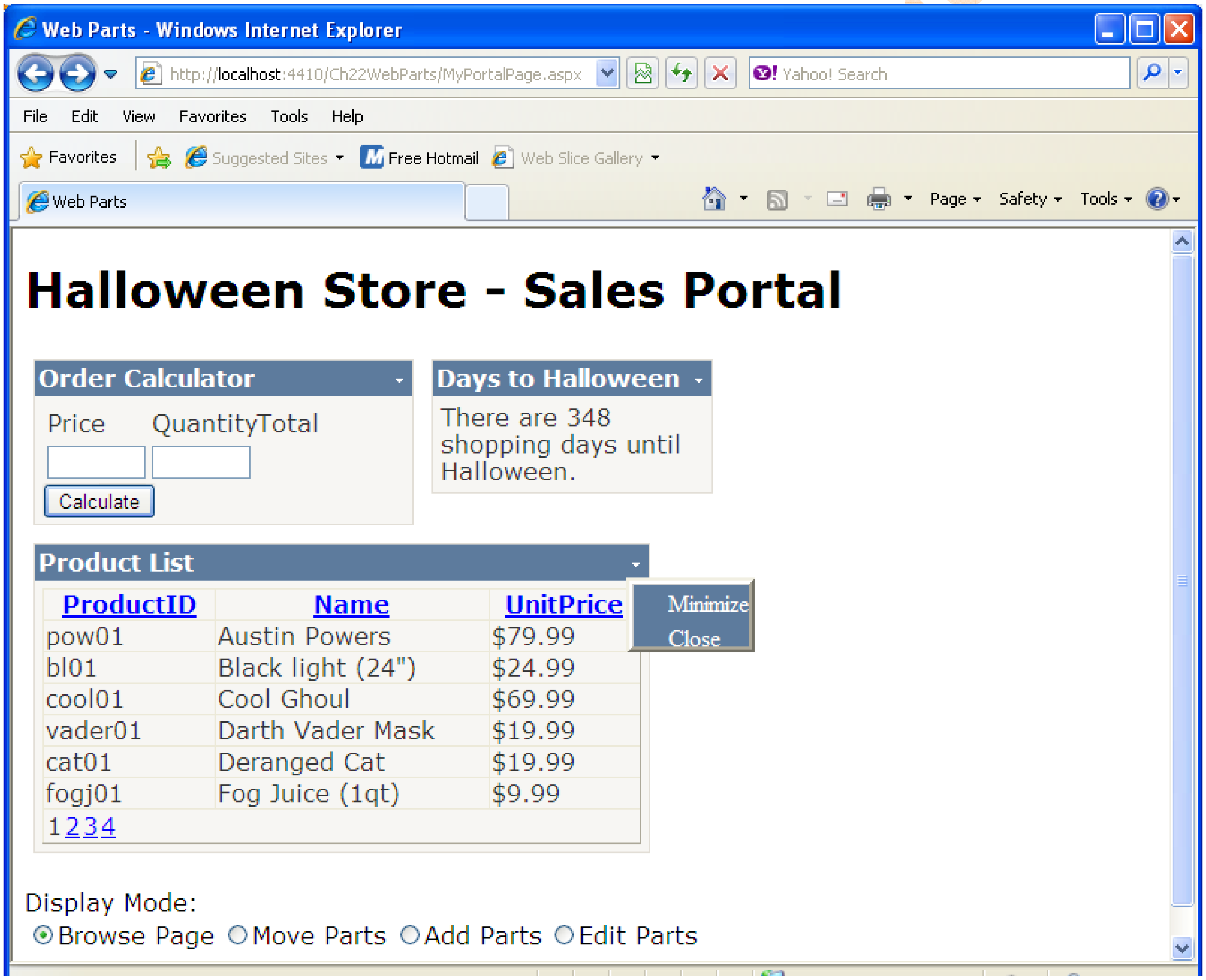
Incidentically, this technique can also be used to deal with problems that occur when the user clicks the *Refresh* button. This posts the page to the server and gets a new response, which refreshes the page, so it has nothing to do with the browser’s cache. However, this can cause problems like a user ordering a product twice without realizing it. Because most users tend to click the *Back* button far more than the *Refresh* button, though, the *Refresh* button causes far fewer errors. That’s why most web developers ignore this problem.

<u>**Using web parts to build portals**</u>

Portals are web pages that display modular content that can be customized by the user. ASP.NET 2.0 provides a built-in portal framework that's conceptually similar to the framework that's used by Miscrosoft SharePoint. Visaul Studio 2005 provides full support for bulding portals, and the .NET Framework exposes an API that can be used to work with portals.

We are going to use an *Ch22WebParts* web application that illustrate some of the feutures for a portal that might be used by a salesperson employed by the company that runs the Holloween Store web site.

Each portal page contains one or more web parts. In our example, the *Sales Portal* page contains three web parts: *Order Calculator, Days to Halloween*, and *Product List*.


Web Parts - Windows Internet Explorer
http://localhost:4410/Ch22WebParts/MyPortalPage.aspx
Yahoo! Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Parts
Page Safety Tools
Calculate
Minimize
Close

Halloween Store - Sales Portal

Order Calculator

Price QuantityTotal

Days to Halloween

There are 348 shopping days until Halloween.

Product List

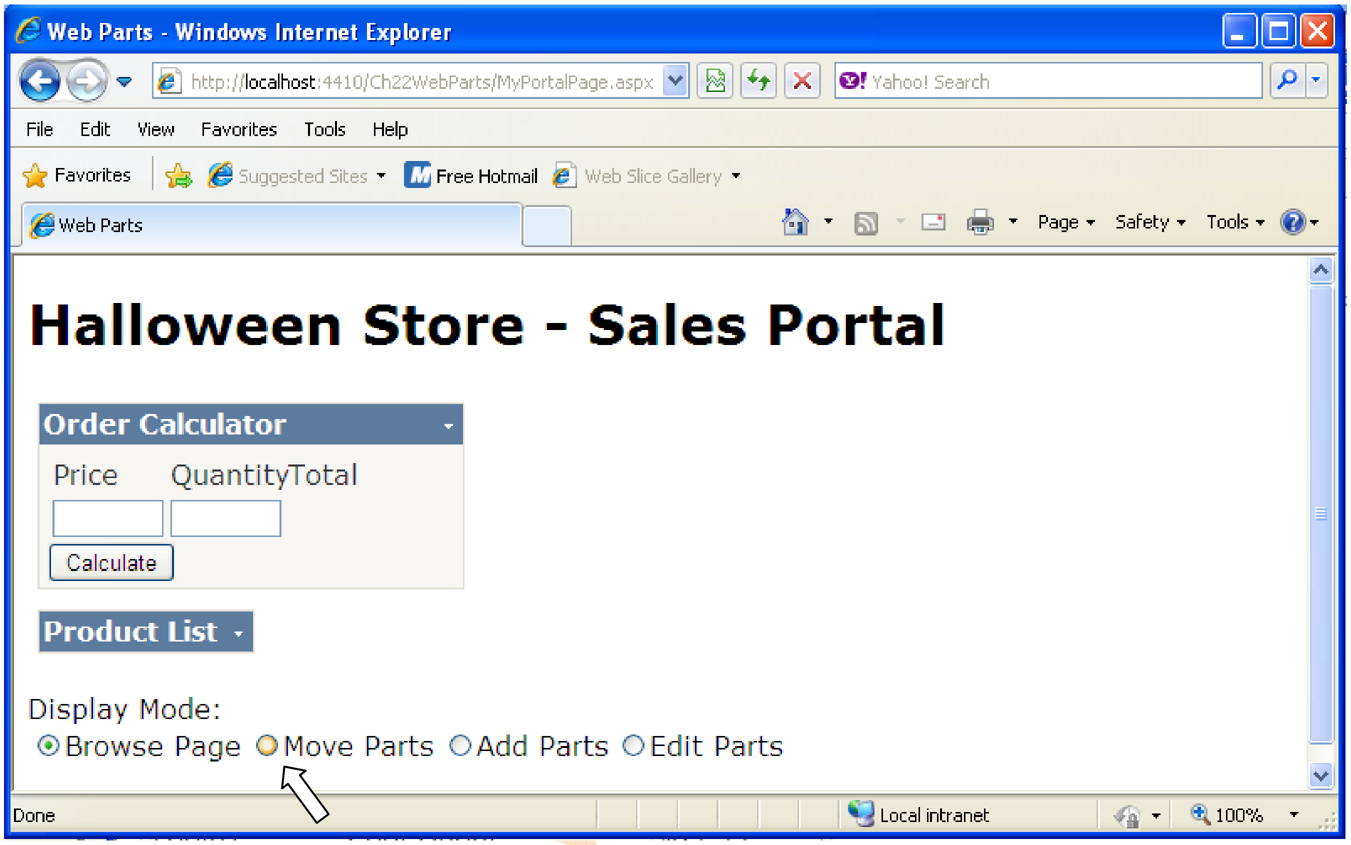
ProductID	Name	UnitPrice
pow01	Austin Powers	$79.99
bl01	Black light (24")	$24.99
cool01	Cool Ghoul	$69.99
vader01	Darth Vader Mask	$19.99
cat01	Deranged Cat	$19.99
fogj01	Fog Juice (1qt)	$9.99
1234		

Display Mode:

[✓ Browse Page] Move Parts Add Parts Edit Parts

In that screen, the user has pulled down the control menu for the *Product List* part and is about to minimize it.

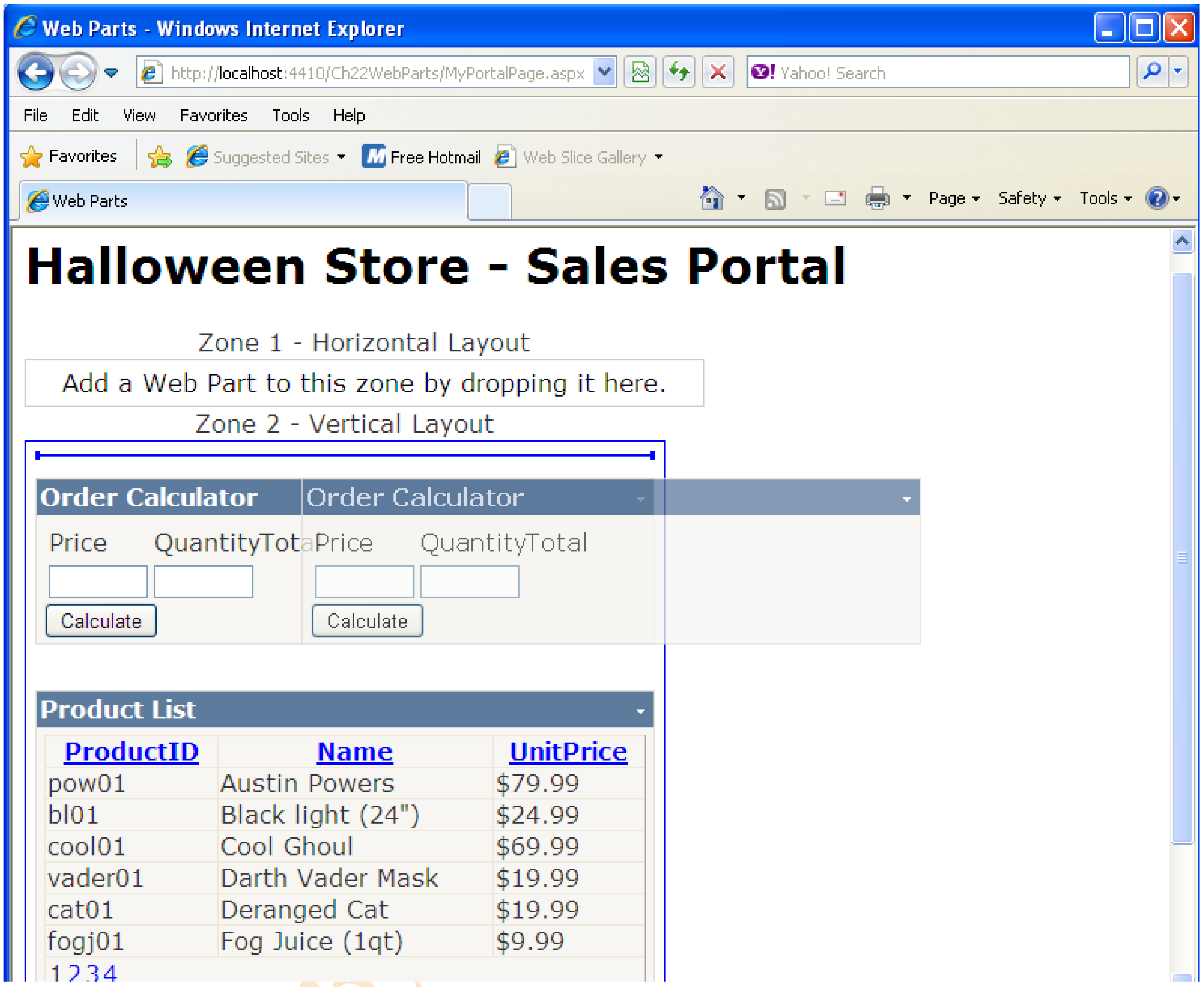
In the second screen, the *Product List* has been minimized. As a result, only its title bar is displayed. In addition, the user is about to click on a radio button to switch to another display mode that allows the user to move web parts.


Web Parts - Windows Internet Explorer
http://localhost:4410/Ch22WebParts/MyPortalPage.aspx
Yahoo! Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Parts
Page Safety Tools
Halloween Store - Sales Portal
Order Calculator
Price QuantityTotal
Calculate
Product List
Display Mode:
Browse Page Move Parts Add Parts Edit Parts
Done
Local intranet
100%

In the next picture (see below), the user has switched to the mode that allows the web parts to be moved between different areas of the page, This cause the header text and borders for these areas, which are known as ***web zones***, to be displayed. For example, the header text for the first web part zone is *"Zone 1 – Horizontal Layout"*. That's because this zone lays out web parts horizontally from left to right. In contrast, the header text for the second web part zone is *"Zone 2 – Vertical Layout".* This is because this zone lays out the web parts vertically from top to bottom.

Once the user has switched to the mode that allows the web parts to be moved, the user can drag a web part from one zone to another. For instance, in the screen, presented below, the user is dragging *"Order Calculator"* part from the first part zone to the second one.

Although this figure doesn't illustrate it, this portal also lets a user add web parts to a web part zone, and it lets a user edit the appearance and behavior of web parts. If, for example, the user clicks on the *Add Parts* radio button, the page displays a zone that lets the user add a part by selecting it from a catalog that contains all available parts for the page. Or, if the user clicks on the *Edit Parts* radio button, an edit command is added to the control list menu for each web page. Then, the user can select this command to edit the appearance and behavior of this part.


Web Parts - Windows Internet Explorer
http://localhost:4410/Ch22WebParts/MyPortalPage.aspx
Yahoo! Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Parts
Page Safety Tools

Halloween Store - Sales Portal

Zone 1 - Horizontal Layout

Add a Web Part to this zone by dropping it here.

Zone 2 - Vertical Layout

Order Calculator

Price QuantityTotal

Calculate

Product List

ProductID	Name	UnitPrice
pow01	Austin Powers	$79.99
bl01	Black light (24")	$24.99
cool01	Cool Ghoul	$69.99
vader01	Darth Vader Mask	$19.99
cat01	Deranged Cat	$19.99
fogj01	Fog Juice (1qt)	$9.99
1 2 3 4		

How the user configurations are stored

The web parts feature uses a ***personalization provider to*** store the portal configuration for each user in the *AspNetDb.mdf* database. That means that whenever a user moves a web part to a new zone, minimizes a web part, or modifies the portal configuration in some other way, the new configuration is saved in this database. That way, the configuration page can be displayed the next time the user visits the site.

The user who is using this feature must be authenticated. The best way to authenticate to authenticate a user is to use forms-based authentication.

Types of controls that can be used as web parts

Types of controls that can be used as web parts

ASP.NET controls that can be placed in a web part zone

- Standard control (such as the *Label* control, etc.)
- User controls (such as the *ProductList* control)
- Custom server controls

Custom *WebPart* control

- Custom *WebPart* controls (such as the *HalloweenCounter* control)

The *WebPart* hierarchy

Panel
 Part
  WebPart
   GenericWebPart

To start a web application with web parts, you can place any ASP.NET control within a web part zone. This includes standard ASP.NET controls such as *Label* and *Calendar* controls, it includes user controls, and it includes custom server controls. When you add one of these controls to a web part zone, the control will automatically be wrapped by *GenericWebPart* class at runtime. This class allows the control to behave like a web part.

The advantage of this approach is that you can use the ASP.NET skills that you're already familiar with to quickly development control that can be used web parts. The advantage is that the *GenericWebPart* class doesn't let you take advantage of several advanced features of web parts. For example, when you use the *GenericWebPart* class, you can't create a connection between two web parts and share data between them.

If you need to use any of the advanced features, you can develop custom *WebPart* controls that inherit the *WebPartclass*. Keep in mind, though, that custom *WebPart* controls are significantly more difficult to develop than the user controls that are used here.

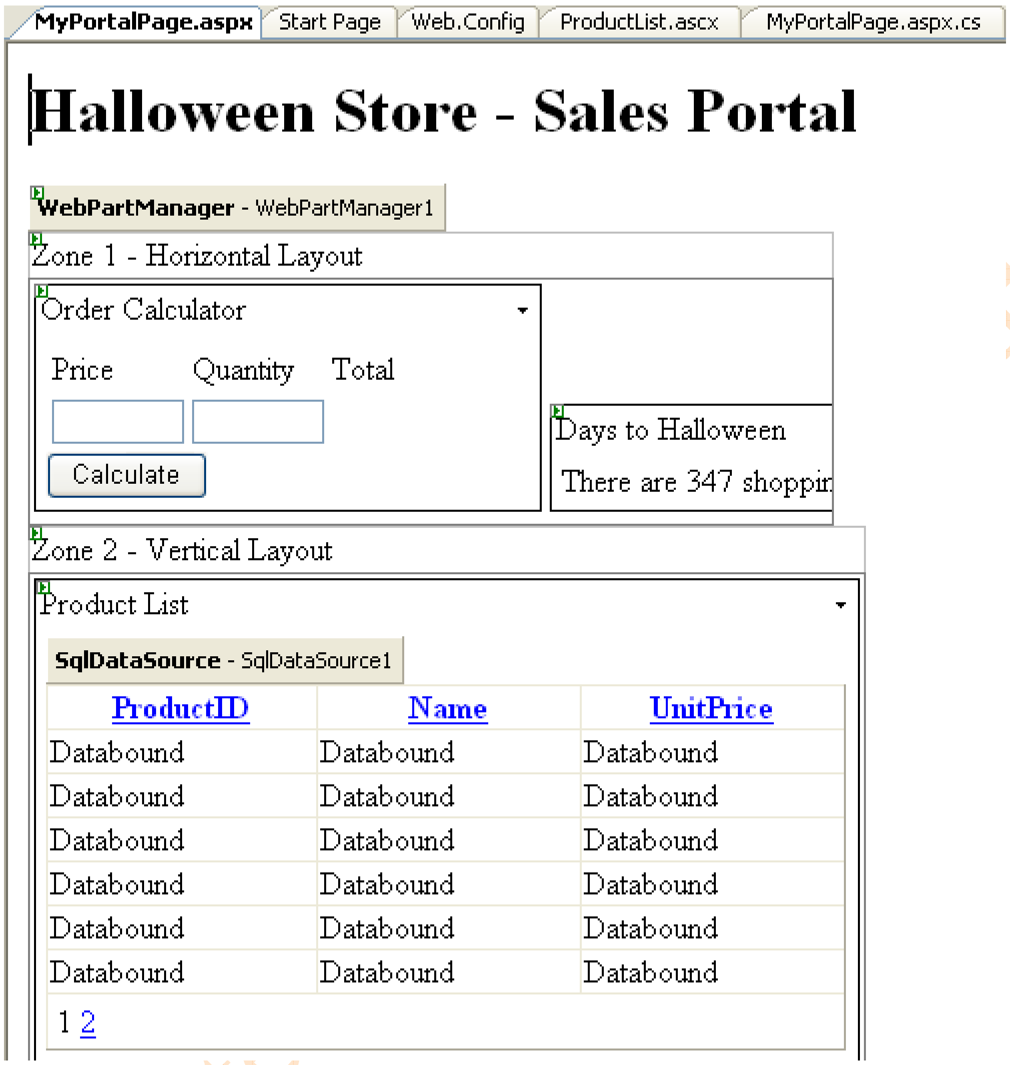
The *WebPart* hierarchy shows that all web parts begin by inheriting the *Panel* class. Then, the *Part* and *WebPart* classes provide additional member that can be used to work with web parts. Finally, the *GenericWebPart* class provides the functionality that's necessary to wrap ASP.NET controls so they can appear and behave as a *WebPart* controls at runtime.

How to use web parts

To start, you open the page in *Design* view, go to the *WebParts* group of the Toolbox, and drag a *WebPartManager* control onto the page. This control manages the web parts and zones, and each portal page must include one of these controls.

Once you've added a *WebPartManager* control to a page, you can add *WebPartZone* controls by dragging them from the Toolbox onto the page. To keep things simple, the page in this figure contains only two zones, but it's common to place several web parts on a page. Often, these web parts zones are added within a table. Then, you can add a web part zone to each cell of the table to give the user more flexibility for how the page can be organized.

Once you've added the *WebPartZone* controls to the page, you can add web parts to the part zones. In the figure below, for example, two *Label* controls have been added to the first web zone, and one *Label* control has been added to the second web part zone. These controls are coded within the *ZoneTemplate* element of the *WebPartZone* controls.


MyPortalPage.aspx | Start Page | Web.Config | ProductList.ascx | MyPortalPage.aspx.cs

Halloween Store - Sales Portal

WebPartManager - WebPartManager1

Zone 1 - Horizontal Layout

Order Calculator

Price Quantity Total

Calculate

Days to Halloween

There are 347 shoppin

Zone 2 - Vertical Layout

Product List

SqlDataSource - SqlDataSource1

ProductID	Name	UnitPrice
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound

1 2

This template specifies the web parts that are displayed the first time a user accesses the portal. However, if the user closes or moves these web parts, the user's changes will be stored in the database, and the changes will be displayed the next time the user access the portal.

Any control that's placed within a *WebPartZone* control is automatically wrapped by the *GenericWebPart* class so it can behave like a web part. As a result, for the *Label* controls in the figure, you can set any attributes that are available to the *GenericWebPart* class. Of these attributes, the *Title* attribute is commonly used to set the title of a part. This attribute is stored in the *Part* class, which is ultimately inherited by the *GenericWebPart* class.

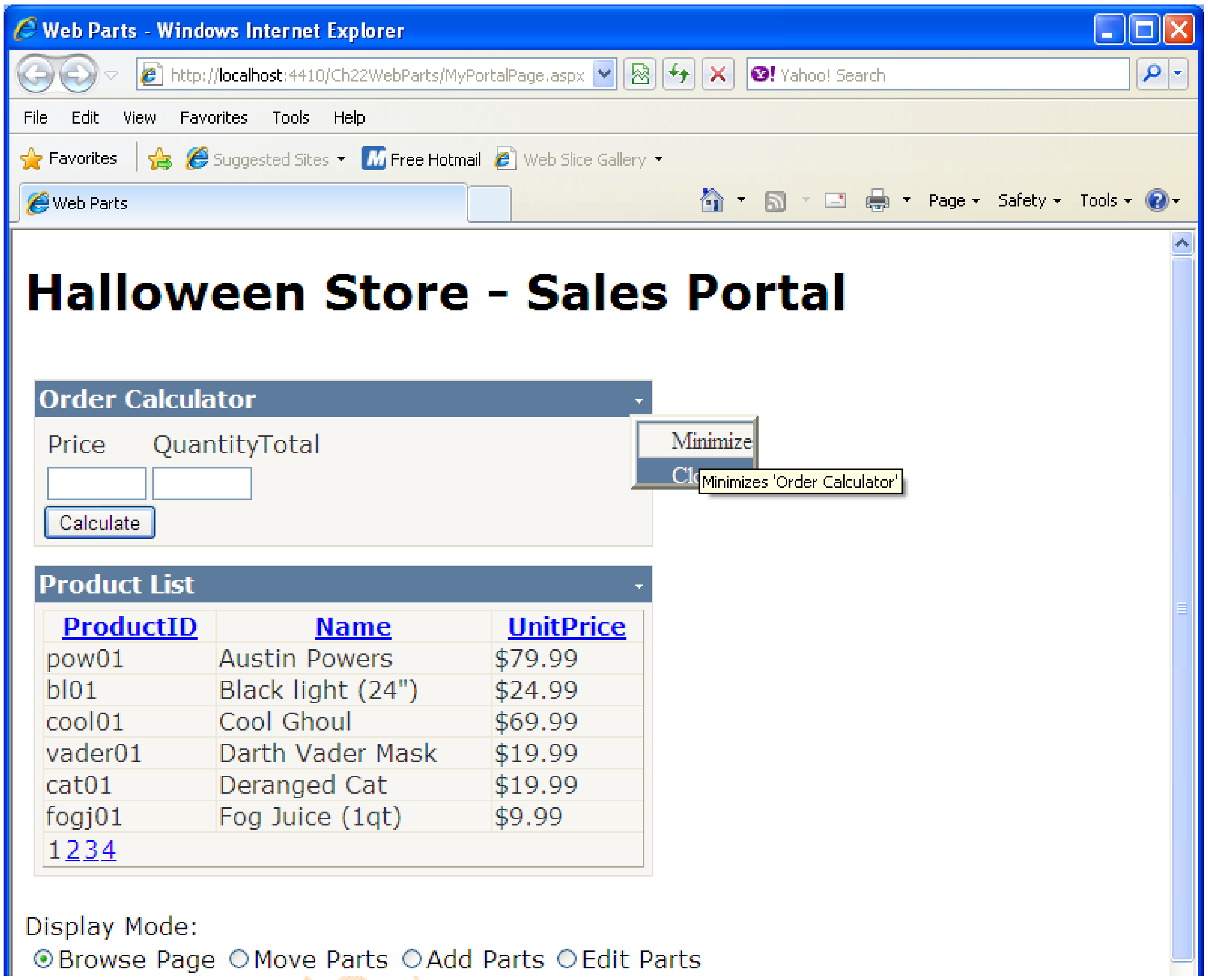
The tags for the page

```
<h1>Halloween Store - Sales Portal</h1>

<asp:WebPartManager ID="WebPartManager1" runat="server">
</asp:WebPartManager>

<asp:WebPartZone ID="WebPartZone1" runat="server"
  HeaderText="Zone 1 - Horizontal Layout"
  LayoutOrientation="Horizontal">
  <ZoneTemplate>
    <uc1:OrderCalculator ID="OrderCalculator1" runat="server"
      Title="Order Calculator"/>
    <mma:HalloweenCounter ID="HalloweenCounter1" runat="server" />
  </ZoneTemplate>
  <EmptyZoneTextStyle />
</asp:WebPartZone>

<asp:WebPartZone ID="WebPartZone2" runat="server"
  HeaderText="Zone 2 - Vertical Layout">
  <ZoneTemplate>
    <uc1:ProductList ID="ProductList1" runat="server"
      Title="Product List">
    </uc1:ProductList>
  </ZoneTemplate>
</asp:WebPartZone>
```

If necessary, you can use the *HeaderText* attribute of a *WebPartZone* control to specify the header text for the web part zone. Then, this text will be displayed above the control in *Design* view, and it will be displayed above the control at runtime whenever the portal enters certain modes such as the mode that allows the user to move web parts between zones.

You may also want to set the *LayoutOrientation* attribute of a *WebPartZone* control to specify the layout orientation. In the picture above, for example, the first web part zone specifies a horizontal orientation, so the web parts are displayed from the left to right. In contrast, the second web part zone doesn't specify a layout orientation, so the default orientation is used to display the web parts vertically.

When you run a page like in the picture below, the default features provide a lot of build-in functionality. For example, the control menu for each web part will contain *Minimize* and *Close* commands for any parts that haven't been minimized and *Restore* and *Close* commands for any parts that have been minimized. So, if you use the *Minimize* command to minimize a web part, you can use the *Restore* command to restore that part. However, if you select the *Close* command, the web part will be removed from the page. In that case, there's no way for the user to add that part to the page unless the page provides for it.


Web Parts - Windows Internet Explorer
http://localhost:4410/Ch22WebParts/MyPortalPage.aspx
Yahoo! Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Parts
Page
Safety
Tools
Minimize
Cl
Minimizes 'Order Calculator'
Calculate

Halloween Store - Sales Portal

Order Calculator

Price QuantityTotal

Product List

ProductID	Name	UnitPrice
pow01	Austin Powers	$79.99
bl01	Black light (24")	$24.99
cool01	Cool Ghoul	$69.99
vader01	Darth Vader Mask	$19.99
cat01	Deranged Cat	$19.99
fogj01	Fog Juice (1qt)	$9.99

1 2 3 4

Display Mode:
[✓ Browse Page] Move Parts Add Parts Edit Parts

How to add a user control to a web part zone

Although figure above shows how to add a standard *Label* control to a web part zone, it's more common to add a user control to a web zone as shown below. Then, this user control will automatically be wrapped by the *GenericWebPart* class so it behaves like a web part.

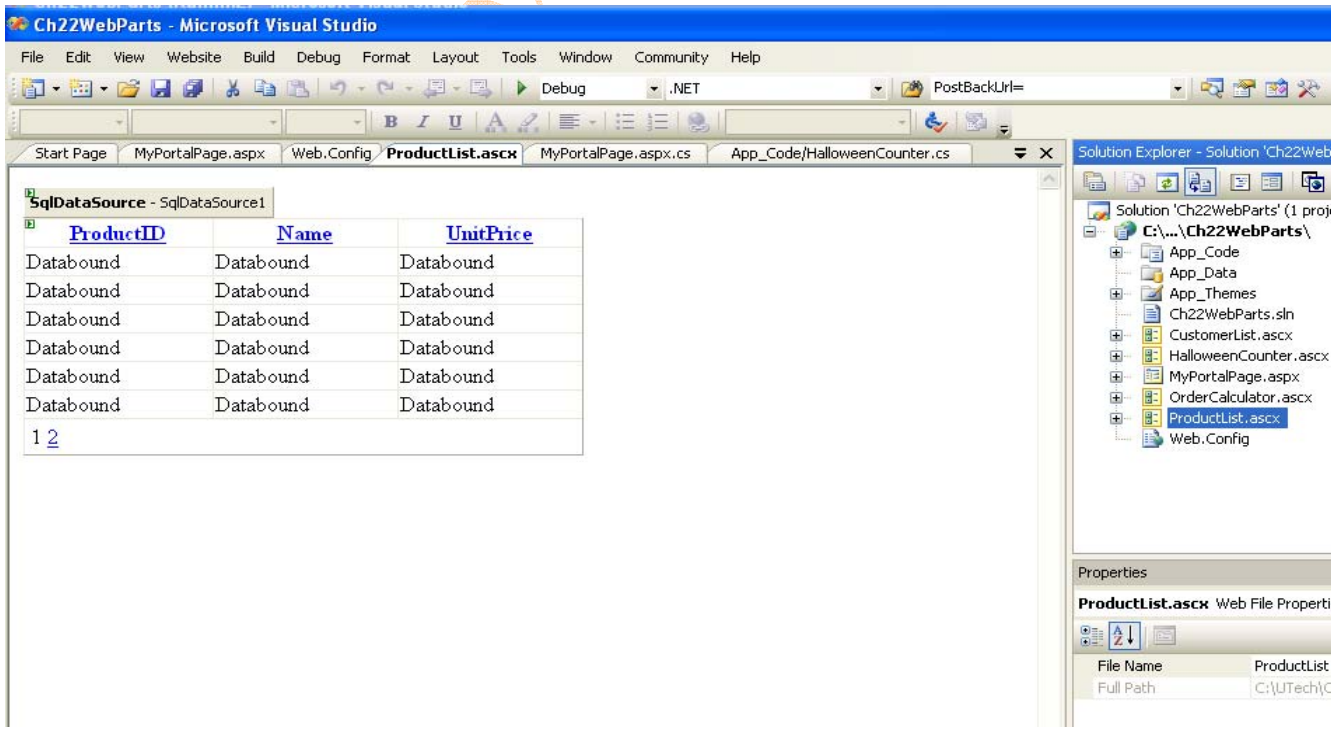
Later in that modular of the "Enterprise Computing II" module, you can learn how to develop user controls. To do that, you:

1. Add a user control (*ascx* file) to the project
2. Use the User Control to design the control
3. Use the Code Editor to write any code that's necessary to get the control to work.

The *ProductList* control in *Source* view

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

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="Data Source=vpougatchev;
    Initial Catalog=Halloween;
    Integrated Security=True;
    MultipleActiveResultSets=False;
    Packet Size=4096;
    Application Name=&quot;
    Microsoft SQL Server Management Studio Express&quot;"
  SelectCommand="SELECT ProductID, Name, UnitPrice
          FROM Products ORDER BY Name">
</asp:SqlDataSource>

<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1"
  AllowPaging="True" AllowSorting="True" AutoGenerateColumns="False"
  PageSize="6" Width="400px">
  <Columns>
    <asp:BoundField DataField="ProductID" HeaderText="ProductID"
      ReadOnly="True" SortExpression="ProductID" />
    <asp:BoundField DataField="Name" HeaderText="Name"
      SortExpression="Name" />
    <asp:BoundField DataField="UnitPrice" HeaderText="UnitPrice"
      SortExpression="UnitPrice"
      DataFormatString="{0:c}" HtmlEncode="False" />
  </Columns>
</asp:GridView>
```


Ch22WebParts - Microsoft Visual Studio
File Edit View Website Build Debug Format Layout Tools Window Community Help
Debug .NET PostBackUrl=
Start Page | MyPortalPage.aspx | Web.Config | ProductList.ascx | MyPortalPage.aspx.cs | App_Code/HalloweenCounter.cs
Solution Explorer - Solution 'Ch22Web
Properties

SqlDataSource - SqlDataSource1

ProductID	Name	UnitPrice
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound

1 2

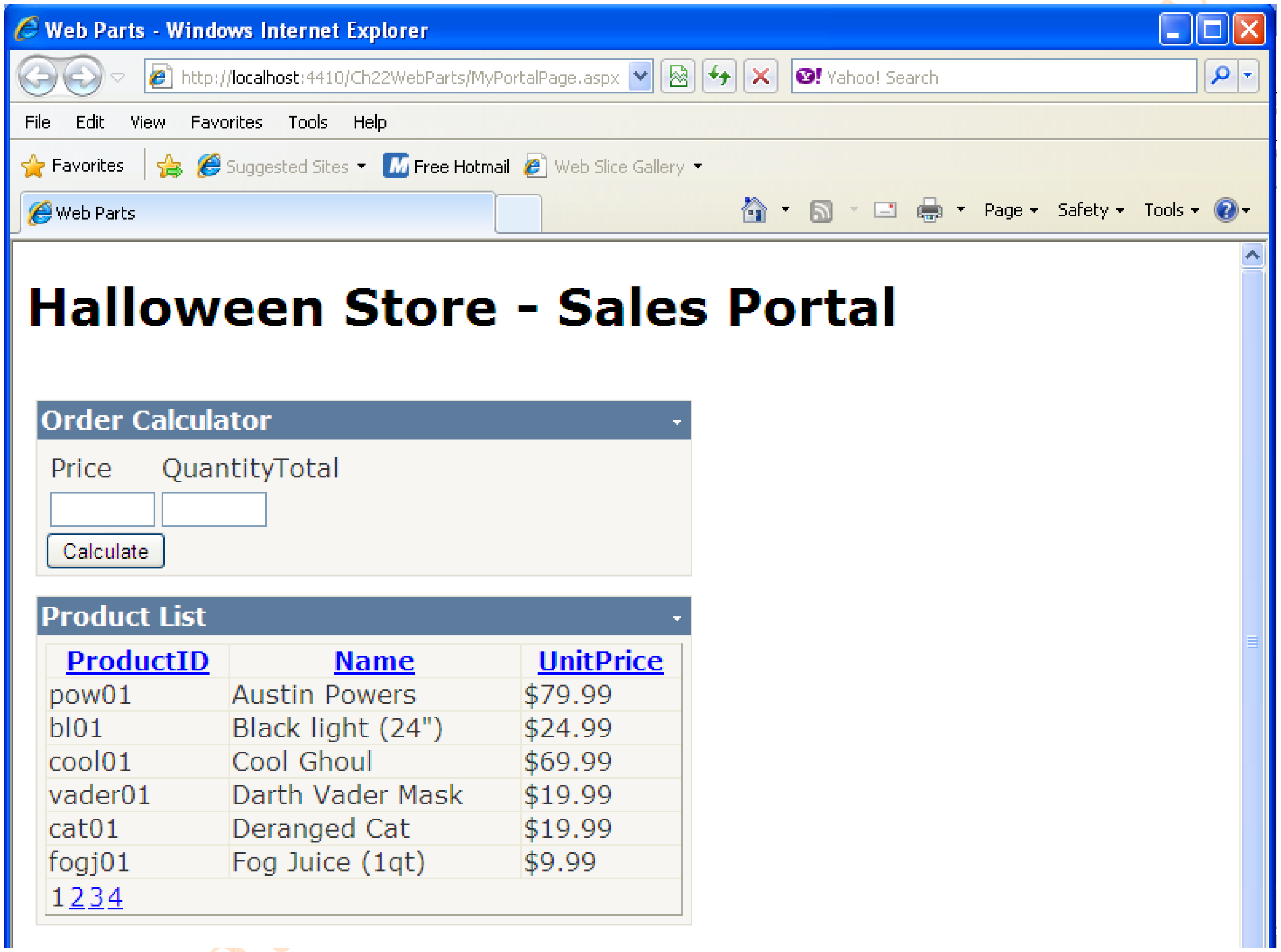
```
Solution 'Ch22WebParts' (1 proj
  C:\...\Ch22WebParts\
    App_Code
    App_Data
    App_Themes
    Ch22WebParts.sln
    CustomerList.ascx
    HalloweenCounter.ascx
    MyPortalPage.aspx
    OrderCalculator.ascx
    ProductList.ascx
    Web.Config
```

ProductList.ascx Web File Properti

File Name	ProductList
Full Path	C:\UTech\C

Once you've created a user control and added it to your project, you can add it to your portal page. The easiest way to do that is to view the page in *Design* view. Then, you can drag the user control from the Solution Explorer onto the page. This automatically generates the *Register* directive that registers the control and the user control element that displayed the control.

Once you add the user control to the web part zone, you can switch to Source view and add a *Title* attribute to the control. In the figure below, for example, the *Title* attribute for the *ProductList* control has been set to *"Product List"*.


Web Parts - Windows Internet Explorer
http://localhost:4410/Ch22WebParts/MyPortalPage.aspx
Yahoo! Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Parts
Page Safety Tools

Halloween Store - Sales Portal

Order Calculator

Price QuantityTotal

Calculate

Product List

ProductID	Name	UnitPrice
pow01	Austin Powers	$79.99
bl01	Black light (24")	$24.99
cool01	Cool Ghoul	$69.99
vader01	Darth Vader Mask	$19.99
cat01	Deranged Cat	$19.99
fogj01	Fog Juice (1qt)	$9.99

1 2 3 4

Although this figure only shows the code for the *ProductList* user control, you can use the same coding techniques to create other user controls such as the *OrderCalculator* and *Days to Halloween* controls. In many ways, these controls are easier to develop than the *ProductList* user control since they don't access a database. Instead, the *OrderCalculator* control allows the user to calculate a total based on quantity and price, and the *Days to Halloween* control calculates the number of days to Halloween and displays the result within a label.

The code that registers a user control for a page

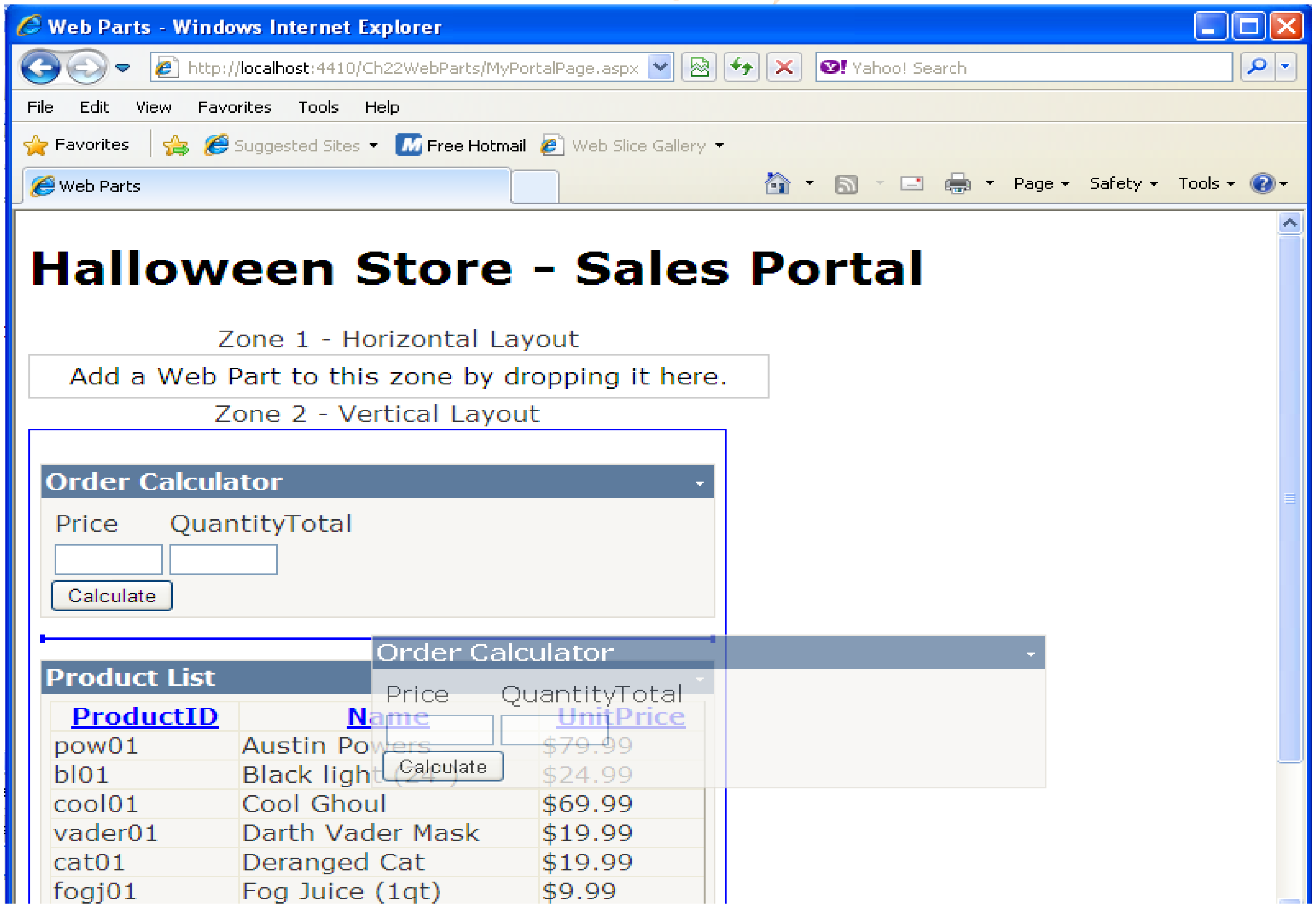
```
<%@ Register Src="ProductList.ascx" TagName="ProductList" TagPrefix="uc1" %>
```

The code that adds a user control to a web zone

```
<asp:WebPartZone ID="WebPartZone1" runat="server"
  HeaderText="Zone 1 - Horizontal Layout"
  LayoutOrientation="Horizontal">
  <ZoneTemplate>
    <uc1:OrderCalculator ID="OrderCalculator1" runat="server"
      Title="Order Calculator"/>
    <mma:HalloweenCounter ID="HalloweenCounter1" runat="server" />
  </ZoneTemplate>
  <EmptyZoneTextStyle />
</asp:WebPartZone>
```

How to let the user move web parts between zones

By default, a page that contains web parts runs in ***browse display mode***. In this mode, you can only minimize, restore, and close web parts. Then, if you want to let the user move web parts between zones, you must switch into ***design display mode***. In this mode, which is shown below, the header text and border appear for each web part zone, and the user can move web parts between zones by dragging them from one zone to another.


Web Parts - Windows Internet Explorer
http://localhost:4410/Ch22WebParts/MyPortalPage.aspx
Yahoo! Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Parts
Page Safety Tools

Halloween Store - Sales Portal

Zone 1 - Horizontal Layout

Add a Web Part to this zone by dropping it here.

Zone 2 - Vertical Layout

Order Calculator

Price QuantityTotal

Calculate

Product List

ProductID	Name	UnitPrice
pow01	Austin Powers	$79.99
bl01	Black light (24")	$24.99
cool01	Cool Ghoul	$69.99
vader01	Darth Vader Mask	$19.99
cat01	Deranged Cat	$19.99
fogj01	Fog Juice (1qt)	$9.99

In this figure, the user is moving the Order *Calculator* web part from the first web part zone to the second web part zone.

To switch between display modes in the code-behind file for a page, you set the *DisplayMode* property of the *WebPartManager* class equal to one of the *DisplayMode* constants that are available from that class. In this example the page includes a *RadioButtonList* control (which is not shown due limit of space in the browser) that lets the user choose between browse display mode and design display mode. If the user chooses the second option, the page enters design display mode so the user can move web parts between zones. Then, the user is done moving web parts, the user can select the first option. This will cause the page to enter browse display mode, which is the optional mode for browsing the data that's displayed on a page.

A radio button list that switches between display modes

```
<asp:RadioButtonList ID="rdoDisplayMode" runat="server" AutoPostBack="True"
  RepeatDirection="Horizontal" Width="455px"
OnSelectedIndexChanged="rdoDisplayMode_SelectedIndexChanged">
    <asp:ListItem Value="Browse Page" Selected="True">Browse Page</asp:ListItem>
    <asp:ListItem Value="Move Parts">Move Parts</asp:ListItem>
    <asp:ListItem Value="Add Parts">Add Parts</asp:ListItem>
    <asp:ListItem>Edit Parts</asp:ListItem>
</asp:RadioButtonList><br />
```

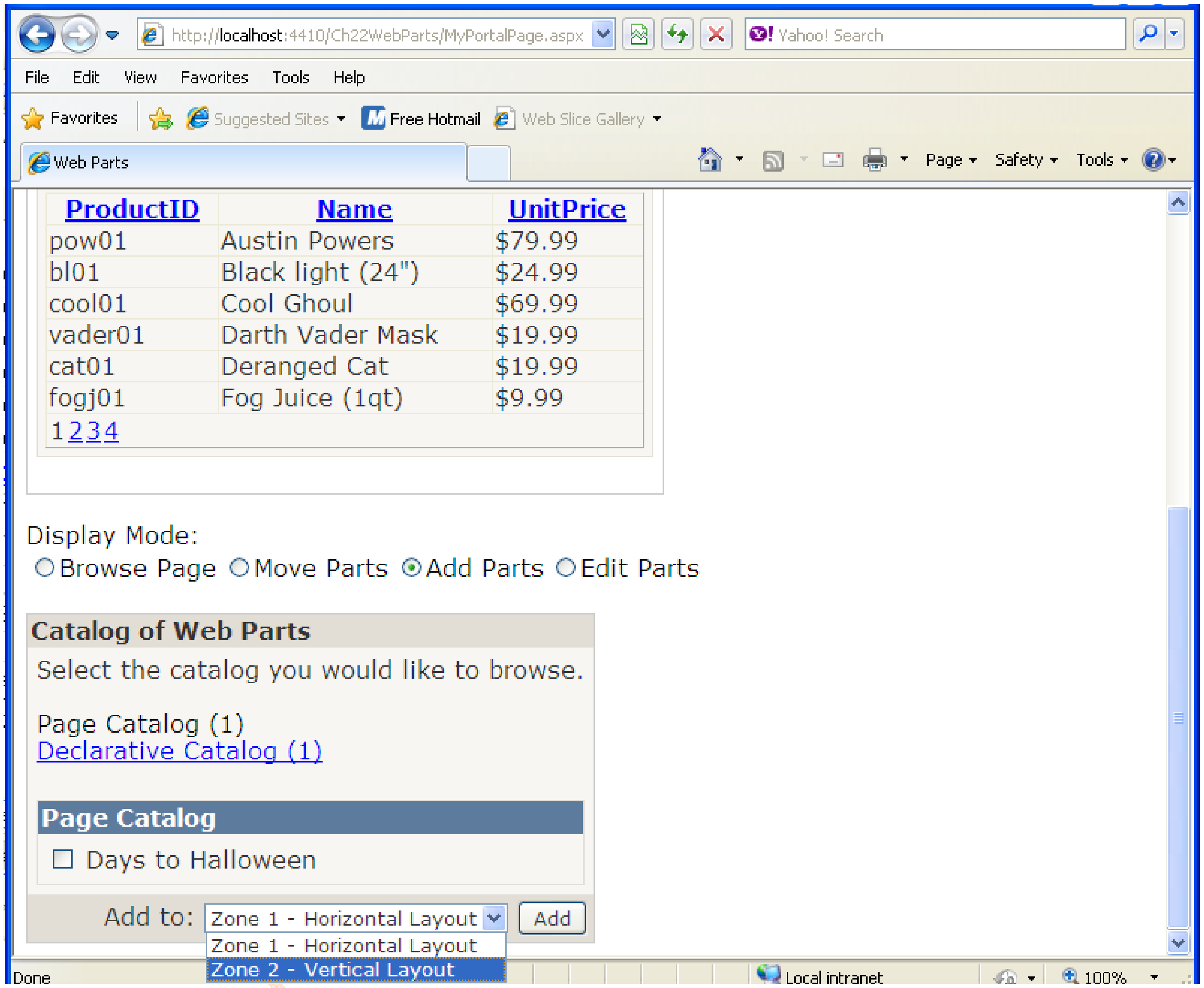
The event handler for the radio button list

```
  protected void rdoDisplayMode_SelectedIndexChanged(object sender, EventArgs e)
  {
    if (rdoDisplayMode.SelectedIndex == 0)
      WebPartManager1.DisplayMode = WebPartManager.BrowseDisplayMode;
    else if (rdoDisplayMode.SelectedIndex == 1)
      WebPartManager1.DisplayMode = WebPartManager.DesignDisplayMode;
    else if (rdoDisplayMode.SelectedIndex == 2)
      WebPartManager1.DisplayMode = WebPartManager.CatalogDisplayMode;
    else if (rdoDisplayMode.SelectedIndex == 3)
      WebPartManager1.DisplayMode = WebPartManager.EditDisplayMode;
  }
```

How to let the user add web parts to a zone

If a user closes a web part when working in browser or design display mode, the part is removed from the page and there's no way for the user to add the web part back to the page. However, you can add functionality to the page that lets the user add web parts that have been closed to the page and add web parts that aren't on the page by default. To do this, you must perform three tasks.

First, you must provide a way for the user to switch to the ***catalog display mode*** that's shown in figure below. To do that, you can extend the *RadioButtonList* control so it includes a third option that lets the user switch into catalog display mode.



ProductID	Name	UnitPrice
pow01	Austin Powers	$79.99
bl01	Black light (24")	$24.99
cool01	Cool Ghoul	$69.99
vader01	Darth Vader Mask	$19.99
cat01	Deranged Cat	$19.99
fogj01	Fog Juice (1qt)	$9.99
1 2 3 4		

Display Mode:
○ Browse Page ○ Move Parts [✓ Add Parts] ○ Edit Parts

Catalog of Web Parts

Select the catalog you would like to browse.

Page Catalog (1)
Declarative Catalog (1)

Page Catalog

☐ Days to Halloween

Add to: Zone 1 - Horizontal Layout [Add]
Zone 1 - Horizontal Layout
Zone 2 - Vertical Layout

Second, you must add a *CatalogZone* control to the page. To do that, you can drag the *CatalogZone* control from the Toolbox onto the page.

Third, you must add one or more of the *CatalogPart* controls within the *CatalogZone* control. In the example, presented above, the *CatalogZone* control contains a *PageCatalogPart* control followed by a *DeclarativeCatalogPart* control. The *DeclarativeCatalogPart* control includes a *WebPartsTemplate* element that declares any web parts that aren't on the default page but should also be available. In this example, the *CustomerList* part is the only part in this catalog, but you can include as many parts as you need.

When the page is running in browse display mode, the catalogs of available web parts that are defined by the *CatalogZone* and *CatalogPart* controls aren't displayed. However, if the user switches to catalog display mode, the catalog is displayed. In this figure, for example, the *CatalogZone* control is displayed in the bottom half of the page, starting with the heading (*Catalog* of *WebParts*) that's provided by the *HeaderText* attribute.

After the heading, the *CatalogZone* control displays *"Select the catalog you would like to browse"* and a menu that lets you switch between the *CatalogPart* controls within the zone. In this figure, the choices are *PageCatalog* and *Declarative Catalog*, and the user has selected *Page Catalog*. This displays a list of all the web parts that were on the page by default, but were closed by the user (this list includes just the *Product List* part). In contrast, if user selects *Declarative Catalog*, the list displays all of the web parts in that catalog. The number in parentheses after each menu item for the catalog zone indicates the number of items in each list (one each).

Under the catalog list, the *CatalogZone* control generates controls for adding a selected web part. Here, the user is using the drop-down list to specify the zone that the selected web part should be added to. Then, when the user clicks the *Add* button, the selected part is added to that zone.

By default, the *CatalogZone* control also displays a *Close* link in its header and a *Close* button next to the *Add* button. This provides the user with a way to close the *CatalogZone* control and exit catalog display mode. However, this also causes the display mode to get out of synch with the *RadioButtonList* control. As a result, the *Visible* attribute of the *HeaderCloseVerb* and *CloseVerb* elements has been used to hide these elements. (When working with web parts, the term ***verb*** is used to refer to the actions that are executed by the buttons, links, and menu items that are available from a web part.)

The *aspx* tags for the catalog

```
<asp:CatalogZone ID="CatalogZone1" runat="server"
   HeaderText="Catalog of Web Parts">
   <ZoneTemplate>
      <asp:PageCatalogPart ID="PageCatalogPart1" runat="server" />
      <asp:DeclarativeCatalogPart ID="DeclarativeCatalogPart1" runat="server">
         <WebPartsTemplate>
            <uc1:CustomerList ID="CustomerList1" runat="server"
               Title="Customer List"/>
         </WebPartsTemplate>
      </asp:DeclarativeCatalogPart>
   </ZoneTemplate>
   <CloseVerb Visible="False" />
   <HeaderCloseVerb Visible="False" />
</asp:CatalogZone>
```

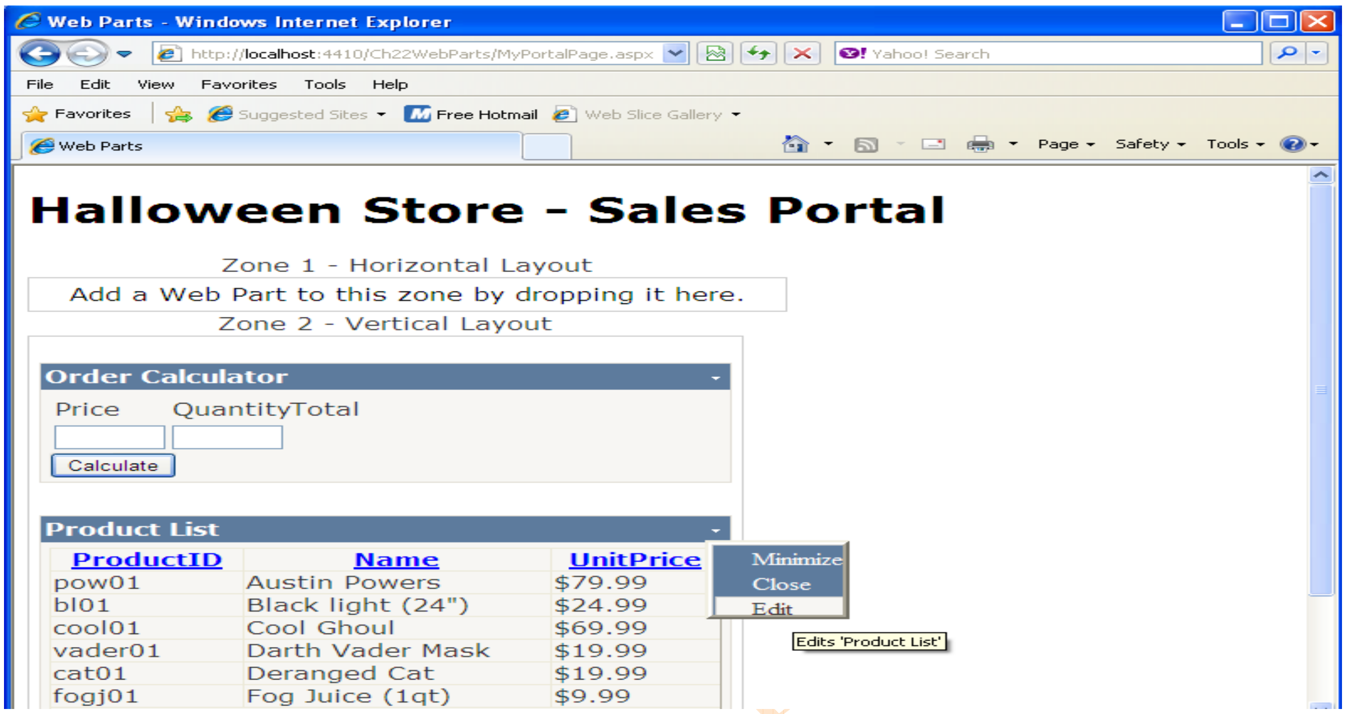
The code that switches to catalog display mode

```
WebPartManager1.DisplayMode = WebPartManager.CatalogDisplayMode;
```

How to let the user edit the properties of a web part

Sometimes you may want to let the user edit the properties that control the appearance and behavior of a web part. To do this, you must perform three tasks that are similar to the three for letting the user add a web part to a page.

First, you must provide a way for the user to switch to the ***edit display mode*** that's shown in the first figure below. To do that, you can extend the *RadioButtonList* control so it includes a forth option that lets the user switch to this mode. Once the switch has been made, an *Edit* command is included in the drop-down control menu for each web part as shown below.


Web Parts - Windows Internet Explorer
http://localhost:4410/Ch22WebParts/MyPortalPage.aspx
Yahoo! Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Parts
Page Safety Tools

Halloween Store - Sales Portal

Zone 1 - Horizontal Layout

Add a Web Part to this zone by dropping it here.

Zone 2 - Vertical Layout

Order Calculator

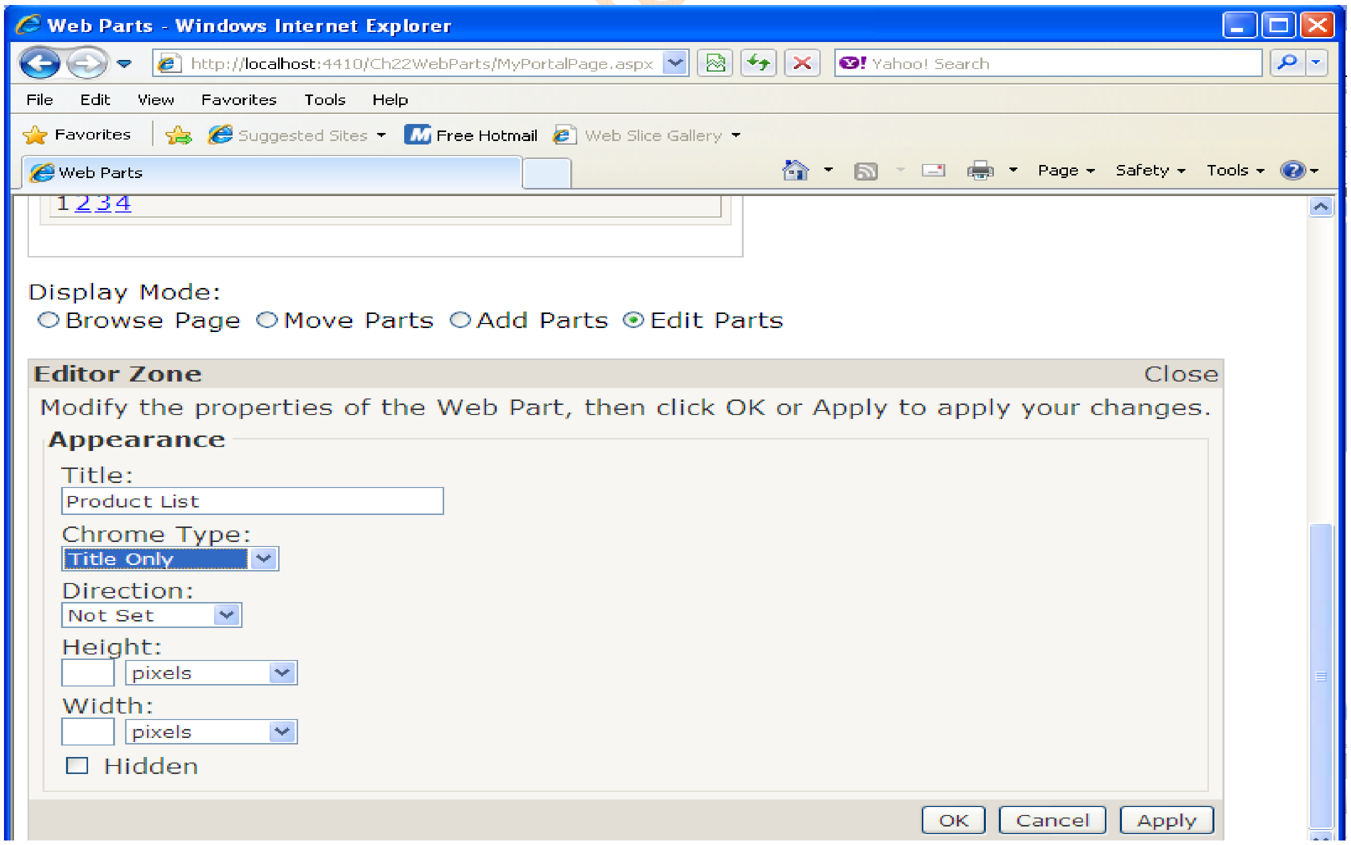
Price QuantityTotal

Calculate

Product List

ProductID	Name	UnitPrice
pow01	Austin Powers	$79.99
bl01	Black light (24")	$24.99
cool01	Cool Ghoul	$69.99
vader01	Darth Vader Mask	$19.99
cat01	Deranged Cat	$19.99
fogj01	Fog Juice (1qt)	$9.99

Then, the user can select this command to display the properties that can be edited, as shown in the second screen.


Web Parts - Windows Internet Explorer
http://localhost:4410/Ch22WebParts/MyPortalPage.aspx
Yahoo! Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Parts
Page Safety Tools

1 2 3 4

Display Mode:

( ) Browse Page ( ) Move Parts ( ) Add Parts [✓ Edit Parts]

Editor Zone Close

Modify the properties of the Web Part, then click OK or Apply to apply your changes.

Appearance

Title: Product List

Chrome Type: Title Only

Direction: Not Set

Height: pixels

Width: pixels

☐ Hidden

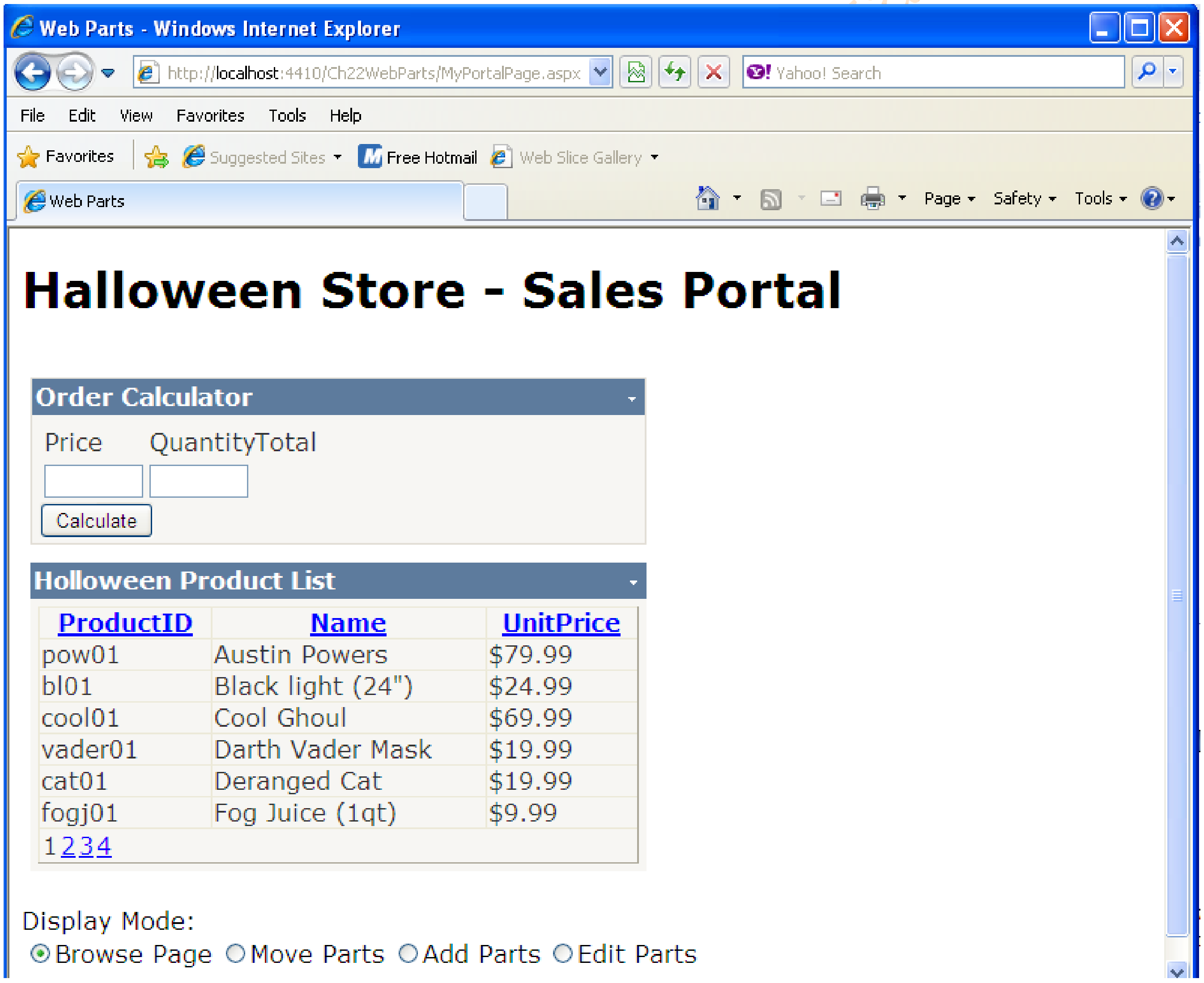
OK Cancel Apply

Second, you must add an *EditZone* control to the page. To do that, you can drag the *EditorZone* control from the Toolbox onto the page. This control is a container control that works similarly to the *WebPartZone* and *CatalogZone* control.

Third, you must add one or more of the *EditPart* controls within the *EditorZone* control. In this figure, for example, the *EditZone* control contains just one *EditorPart* control, the *AppearanceEditorPart* control. But you could add other *EditorPart* controls such as the *BehaviorEditorPart*, *LayoutEditorPart*, or *PropertyGridEditorPart* controls.

By default, the *EditorZone* control displays a *Close* link in its header. This lets the user close the *EditorZone* control, which hides this control and any subordinate *EditorPart* controls, but doesn't exit edit display mode. As a result, the *Edit* command is still available from each of the web parts on the page.

Although the *EditorPart* controls let the user modify the properties of a web part, most of these properties only provide for minor cosmetic changes. In the figure below


Web Parts - Windows Internet Explorer
http://localhost:4410/Ch22WebParts/MyPortalPage.aspx
Yahoo! Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Parts
Page Safety Tools

Halloween Store - Sales Portal

Order Calculator

Price QuantityTotal

Calculate

Holloween Product List

ProductID	Name	UnitPrice
pow01	Austin Powers	$79.99
bl01	Black light (24")	$24.99
cool01	Cool Ghoul	$69.99
vader01	Darth Vader Mask	$19.99
cat01	Deranged Cat	$19.99
fogj01	Fog Juice (1qt)	$9.99

1 2 3 4

Display Mode:
[✓ Browse Page] Move Parts Add Parts Edit Parts

the user has modified the Product List web part by changing the title from *"Product List"* to *"Halloween Product List".*

When you working with web parts, the term ***chrome*** is used to refer to the non-content area of a web part such as the border, title bar, minimize/close buttons, and so on.

The *aspx* tags for the editor zone

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

The code that switches to *EditDisplayMode*

```
WebPartManager1.DisplayMode = WebPartManager.EditDisplayMode;
```

The *EditorPart* controls

AppearanceEditorPart
BehaviorEditorPart
LayoutEditorPart
PropertyGridEditorPart

How to write code that works with web parts

You can develop most portals using just the skills that have already been presented above. However, if you need to customize a portal beyond what has been shown, the portal framework exposes an API that lets you write code that works with web parts. This API also lets you develop custom *WebPart* controls that can take advantage of some of the advanced features of the portal framework.

Classes for working with web parts

Class	Description
WebPartManager	Can be used to work with all of the zones and web parts on the page, to change display modes for the page, and to respond to events that are raised by the zones and web parts on the page.
WebPartZone	Can be used to modify the propertis of a web part zone. Many of these properties control the appearance of the web part within this zone.
CatalogZone	Can be used to modify the properties of a catalog zone. Many of these properties control the appearance of the catalog parts within this zone.
EditorZone	Can be used to modify the properties of an editor zone. Many of these properties control the appearance of the editor parts within this zone.
ConnectionZone	Can be used to work with connection betweencustom WebPart controls.
WebPartVerb	Can be used to work a verb, which is a button, link, or menu item in the title

	bar of a web part.
WebPart	Can be used to work with parts at runtime, or it can be overridden to create a custom *WebPart* that takes advantage of the advanced features of the portal framework.
GenericWebPart	Can be used to work with user controls and custom server controls within a webpart zone. Many of its properties are inherited from the *Panel* and *Part* classes.The ASP.NET 2.0 portal framework automatically uses this class whenever it needs to wrap web parts that are created from ASP.NET controls such as standard controls, user controls, and custom server controls.
PageCatalogPart	Can be used to work with the web parts in the page catalog, including parts that a user has closed.
DeclarativeCatalogPart	Can be used to work with the web parts in the declarative catalog.
ImportCatalogPart	Can be used to import a catalog of the web parts.
BehaviorEditorPart	Can be used to apply behavior changes to an associated web part.
AppearanceEditorPart	Can be used to apply appearance changes to an associated web part.
LayoutEditorPart	Can be used to apply layout changes to web parts.
PropertyGridEditorPart	Can be used to change the property grid of a web part.

In general, the classes described in that table correspond with the controls that are available from the *WebParts* group in Visual Studio's Toolbox. Most of the time, you can use Visual Studio to set the properties for these controls at design time. Whenever necessary, though you can write code that uses these classes to work with these controls at runtime. To do that, you often need to get a more complete description of the class for the control by looking it up in the documentation for the .NET Framework class library.

How to develop and use a custom *WebPart* control

The code that defines the custom *WebPart* control

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

namespace Utech
{
        public class HalloweenCounter : WebPart
        {
                public HalloweenCounter()
                {
                        this.Title = "Days to Halloween";
                }

                protected override void RenderContents(HtmlTextWriter writer)
                {
                        DateTime today = DateTime.Today;

                        DateTime halloween = new DateTime(DateTime.Today.Year, 10, 31);

                        if (today > halloween)
                                halloween = halloween.AddYears(1);

                        TimeSpan span = halloween - today;

                        string content = "There are " + span.Days
                                + " shopping days until Halloween.";

                        writer.Write(content);
                }
        }
}
```

Code that registers a custom *WebPart* control for a page

```
<%@ Register TagPrefix="mma" Namespace="Utech" %>
```

Code that places a custom *WebPart* control on a page

```
<mma:HalloweenCounter ID="HalloweenCounter1" runat="server" />
```

The constructor for the *HalloweenCounter* class sets the *Title* property to *"Days to Halloween".* This is possible because the *Title* property is available from the *Part* class, which is inherited by the *WebPart* class and the *HalloweenCounter* class.

Then, the *HalloweenCounter* class overrides the *RenderContents* method of *WebControl* class. Like all web server controls, the *Panel* class inherits the *WebControl* class. Because the *Part* inherits the *Panel* class, the *RenderContents* method is available from the *HalloweenCounter* class. This method is called by the web page whenever the page needs to display the control. The code within this method calculates the number of shopping days to *Halloween*, creates a message that includes the results of this calculation, and uses the *HtmlTextWriter* object to write that message to the page.

Since the message doesn't include any HTML tags, this *HalloweenCounter* class causes plain text to be displayed as the content of the web part. However, you can include HTML tags in the message if you want to format it.