Unit 10

12.0 User Controls & Graphics (Text Book Chapter 11)

12.0 User Controls & Graphics

12..1 Introduction

In this chapter, you'll consider two ways to extend your web pages another notch. First, you'll tackle user controls, which give you an efficient way to reuse a block of user interface markup—and the code that goes with it. User controls are a key tool for building modular web applications. They can also help you create consistent website designs and reuse your hard work.

Next, you'll explore custom drawing with GDI+. You'll see how you can paint exactly the image you need on request. You'll also learn the best way to incorporate these images into your web pages.

12.2 User Controls

A well-built web application divides its work into discrete, independent blocks. The more modular your web application is, the easier it is to maintain your code, troubleshoot problems, and reuse key bits of functionality.

Although it's easy enough to reuse code (you simply need to pull it out of your pages and put it into separate classes), it's not as straightforward to reuse web page markup. You can cut and paste blocks of HTML and ASP.NET control tags, but this causes endless headaches if you want to change your markup later. Instead, you need a way to wrap up web page markup in a reusable package, just as you can wrap up ordinary C# code. The trick is to create a user control. User controls look pretty much the same as ASP.NET web forms. Like web forms, they are composed of a markup portion with HTML and control tags (the .ascx file) and can optionally use a code-behind file with event-handling logic. They can also include the same range of HTML content and ASP.NET controls, and they experience the same events as the Page object (such as Load and PreRender). The only differences between user controls and web pages are as follows:

12.0 User Controls & Graphics

- User controls use the file extension .ascx instead of .aspx, and their code-behind files inherit from the System.Web.UI.UserControl class. In fact, the UserControl class and the Page class both inherit from the same base classes, which is why they share so many of the same methods and events, as shown in the inheritance diagram in Figure 12.1-1.
- The .ascx file for a user control begins with a **<%@ Control %>** directive instead of a **<%@ Page %>** directive.
- User controls can't be requested directly by a web browser. Instead, they must be embedded inside other web pages

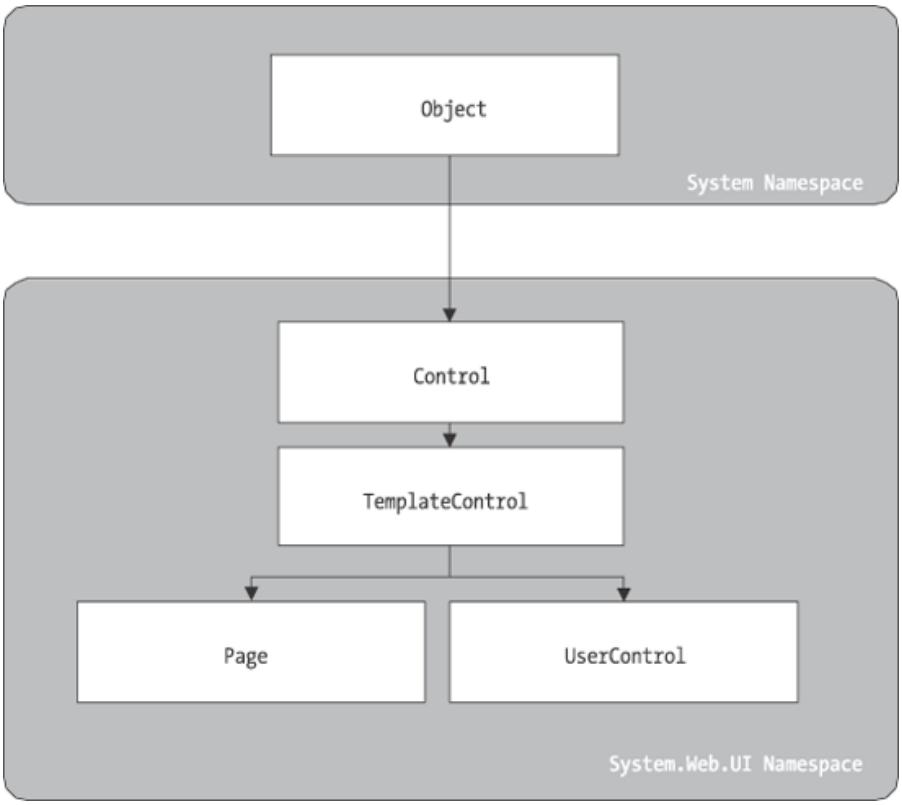

Figure 12.1-1 - User Controls Inheritance Diagram

12..3 Creating a Simple User Control

You can create a user control in Visual Studio in much the same way you add a web page. Just select Website Add New Item, and choose Web User Control from the list.

The following user control contains a single Label control:

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

<asp:Label id="lblFooter" runat="server" />
```

Example 12.3-1a User Control Presentation

```
public partial class Footer : System.Web.UI.UserControl
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         lblFooter.Text = "This page was served at ";
         lblFooter.Text += DateTime.Now.ToString();
      }
   }
```

Example 12.3-1b User Control Code Behind

Note that the Control directive uses the same attributes used in the Page directive for a web page, including Language, AutoEventWireup, and Inherits. The code-behind class for this sample user control is similarly straightforward. It uses the UserControl.Load event to add some text to the label:

To test this user control, you need to insert it into a web page. This is a two-step process. First, you need to add a Register directive to the page that will contain the user control. You place the Register directive immediately after the **Page directive**. The Register directive identifies the control you want to use and associates it with a unique control prefix, as shown here:

<%@ Register TagPrefix="apress" TagName="Footer" Src="Footer.ascx" %>

The Register directive specifies a tag prefix and name. Tag prefixes group sets of related controls (for example, all ASP.NET web controls use the tag prefix asp). Tag prefixes are usually lowercase—technically, they are case-insensitive—and should be unique for your company or organization. The Src directive identifies the location of the user control template file, not the code-behind file.

Second, you can now add the user control whenever you want (and as many times as you want) in the page by inserting its control tag. Consider this page example:

```
<%@ Page Language="C#" AutoEventWireup="true"
   CodeFile="FooterHost.aspx.cs" Inherits="FooterHost"%>
<%@ Register TagPrefix="apress" TagName="Footer" Src="Footer.ascx" %>

<!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 runat="server">
   <title>Footer Host</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
     <h1>A Page With a Footer</h1><hr />
     Static Page Text<br /><br />
     <apress:Footer id="Footer1" runat="server" />
   </div>
   </form>
</body>
</html>
```

Example 12.3-1c Using the User Control

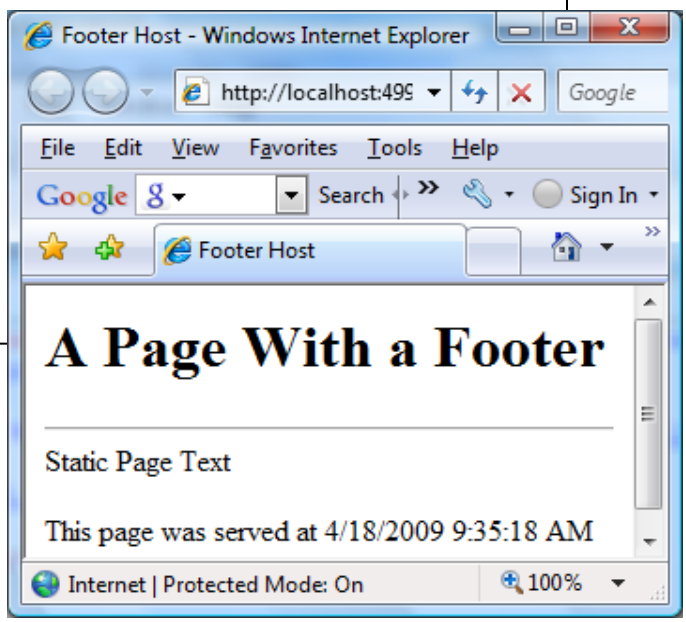

Figure 12.3-1 User Control In Action

➢12.0 User Controls & Graphics

This example (shown in Figure 12.3-1) demonstrates a simple way that you can create a header or footer and reuse it in all the pages in your website just by adding a user control. In the case of your simple footer, you won't save much code. However, this approach will become much more useful for a complex control with extensive formatting or several contained controls.

Of course, this only scratches the surface of what you can do with a user control. In the following sections, you'll learn how to enhance a control with properties, methods, and events—transforming it from a simple "include file" into a full-fledged object.

NOTE:
The Page class provides a special LoadControl() method that allows you to create a user control dynamically at runtime from an .ascx file. The user control is returned to you as a control object, which you can then add to the Controls collection of a container control on the web page (such as Place Holder or Panel) to display it on the page. This technique isn't a good substitute for declaratively using a user control, because it's more complex. However, it does have some interesting applications if you want to generate a user interface dynamically.

In Visual Studio, you have a useful shortcut for adding a user control to a page without typing the Register directive by hand. Start by opening the web page you want to use. Then, find the .ascx file for the user control in the Solution Explorer. Drag it from the Solution Explorer and drop it onto the visual design area of your web form (not the source view area). Visual Studio will automatically add the Register directive for the user control, as well as an instance of the user control tag.

12.4 Independent User Controls

Conceptually, two types of user controls exist: independent and integrated. Independent user controls don't interact with the rest of the code on your form. The Footer user control is one such example. Another example might be a LinkMenu control that contains a list of buttons offering links to other pages. This LinkMenu user control can handle the events for all the buttons and then run the appropriate Response.Redirect() code to move to another web page. Or it can just be an ordinary HyperLink control that doesn't have any associated server-side code. Every page in the website can then include the same LinkMenu user control, enabling painless website navigation with no need to worry about frames.

```
<%@ Control Language="C#" AutoEventWireup="true"
    CodeBehind="LinkMenu.ascx.cs" Inherits="TestingUserControls.LinkMenu" %>
<div>
 Products:<br />
 <asp:HyperLink id="lnkBooks" runat="server"
  NavigateUrl="TestUserControl.aspx?product=Books">Books
 </asp:HyperLink><br />
 <asp:HyperLink id="lnkToys" runat="server"
  NavigateUrl="TestUserControl.aspx?product=Toys">Toys
 </asp:HyperLink><br />
 <asp:HyperLink id="lnkSports" runat="server"
  NavigateUrl="TestUserControl.aspx?product=Sports">Sports
 </asp:HyperLink><br />
 <asp:HyperLink id="lnkFurniture" runat="server"
  NavigateUrl="TestUserControl.aspx?product=Furniture">Furniture
 </asp:HyperLink>
</div>
```

Example 12.4-1a An independent User Control

NOTE: You can use the more feature-rich navigation controls to provide Website navigation. Creating your own custom controls gives you a simple, more flexible, but less powerful approach to providing navigation. You might use custom controls rather than a whole site map for straightforward navigation between a few pages.

The following sample defines a simple control that presents an attractively formatted list of links. Note that the style attribute of the tag (which defines fonts and formatting) has been omitted for clarity.

The links don't actually trigger any server-side code—instead, they render themselves as ordinary HTML anchor tags with a hard-coded URL.

To test this menu, you can use the following TestingUserControls.aspx web page. It includes two controls: the Menu control and a Label control that displays the product query string parameter. Both are positioned using a table.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="TestUserControl.aspx.cs"
    Inherits="TestingUserControls._Default" %>
<%@ Register TagPrefix="uhclKP" TagName="LinkMenu" Src="LinkMenu.ascx" %>
<!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>Menu Host</title>
</head>
<body>
  <form id="form1" runat="server">
<div>
    <table>
      <tr>
        <td><uhclKP:LinkMenu id="Menu1" runat="server" /></td>
        <td><asp:Label id="lblSelection" runat="server" /></td>
      </tr>
    </table>
  </div>
  </form>
</body>
</html>
```

Example 12.4-1b Using independent User Control

When the TestingUserControls.aspx page loads, it adds the appropriate information to the lblSelection control:

```
protected void Page_Load(Object sender, EventArgs e)
{
  if (Request.Params["product"] != null)
  {
    lblSelection.Text = "You chose: ";
    lblSelection.Text += Request.Params["product"];
  }
}
```

Example 12.4-1c Using Code behind for the independent User Control

➢12.0 User Controls & Graphics

Following figure 12.4-1 shows the end result. Whenever you click a button, the page is posted back, and the text is updated.

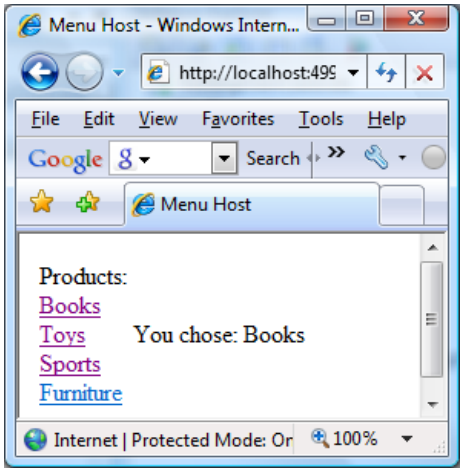

Figure 12.4-1a

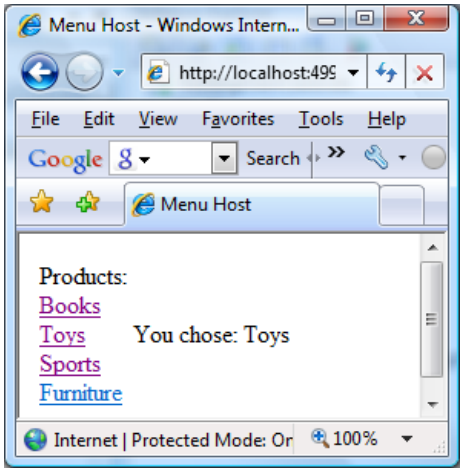

Figure 12.4-1b

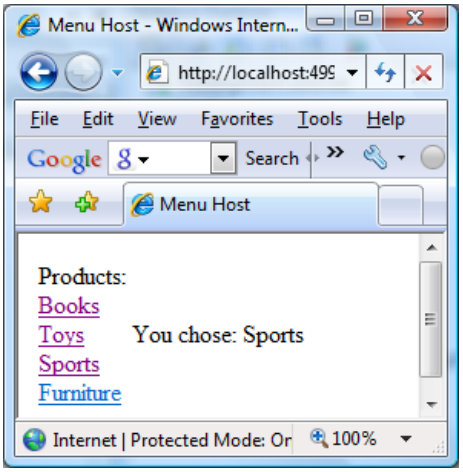

Figure 12.4-1c

You could use the **LinkMenu** control to repeat the same menu on several pages. This is particularly handy in a situation where you can't use master pages to standardize layout (possibly because the pages are too different).

12.5 Integrated User Controls

Integrated user controls interact in one way or another with the web page that hosts them. When you're designing these controls, the class-based design tips you learned in Chapter 4 really become useful.

A typical example is a user control that allows some level of configuration through properties. For instance, you can create a footer that supports two different display formats: long date and short time. To add a further level of refinement, the Footer user control allows the web page to specify the appropriate display format using an enumeration.

The first step is to create an enumeration in the custom Footer class. Remember, an enumeration is simply a type of constant that is internally stored as an integer but is set in code by using one of the allowed names you specify.

Variables that use the FooterFormat enumeration can take the value FooterFormat.LongDate or FooterFormat.ShortTime:

```
public enum FooterFormat
{
  LongDate,
  ShortTime
}
```

The next step is to add a property to the Footer class that allows the web page to retrieve or set the current format applied to the footer. The actual format is stored in a private variable called _format, which is set to the long date format by default when the control is first created. (You can Accomplish the same effect, in a slightly sloppier way, by using a public member variable named Format instead of a full property procedure.) If you're hazy on how property procedures work, feel free to review the explanation in Chapter 3. Finally, the **UserControl.Load** event handler needs to take account of the current footer state and format the output accordingly. The following is the full Footer class code:

```
private FooterFormat format = FooterFormat.LongDate;

public FooterFormat Format
{
   get { return format; }
   set { format = value; }
}
```

```
public partial class Footer : System.Web.UI.UserControl {
  public enum FooterFormat { LongDate, ShortTime }
  private FooterFormat format = FooterFormat.LongDate;

  public FooterFormat Format {
        get { return format; }
        set { format = value; }
  }

  protected void Page_Load(object sender, EventArgs e){
    lblFooter.Text = "This page was served at ";

    if (format == FooterFormat.LongDate)
    {
     lblFooter.Text += DateTime.Now.ToLongDateString();
    }
    else if (format == FooterFormat.ShortTime)
    {
     lblFooter.Text += DateTime.Now.ToShortTimeString();
    }
  }
}
```

Example 12.4-1 Using an Independent User Control

To test this footer, you need to create a page that modifies the Format property of the Footer user control. Figure 12-4.1 shows an example page, which automatically sets the Format property for the user control to match a radio button selection whenever the page is posted back.

➢12.0 User Controls & Graphics

To test this footer, you need to create a page that modifies the Format property of the Footer user control. Figure 13-4.2 shows an example page, which automatically sets the Format property for the user control to match a radio button selection whenever the page is posted back.

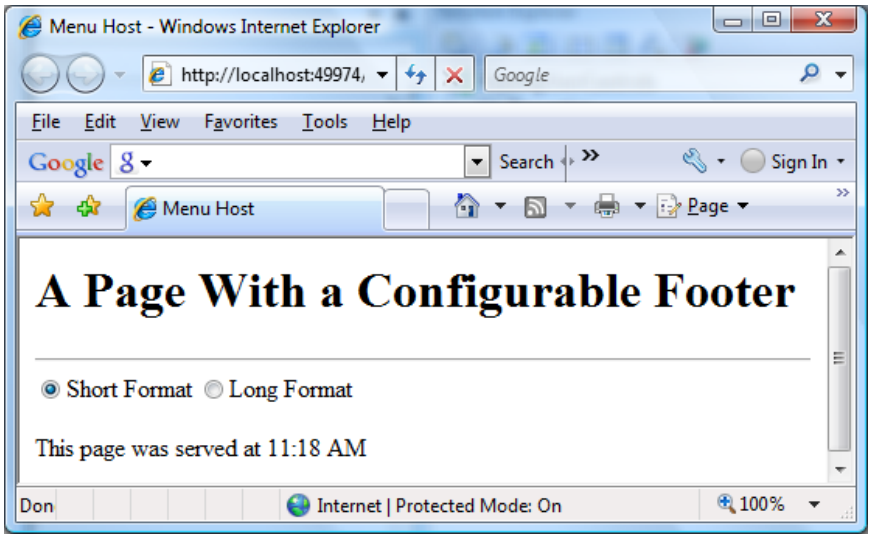

Figure 12.5-1a Format Property Set

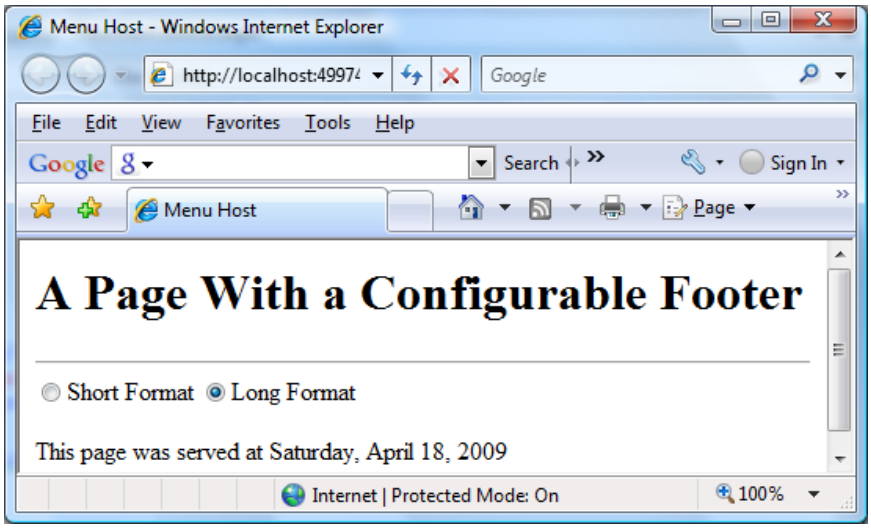

Figure 12.5-1b Format Property Set

12.6 User Controls Events

Another way that communication can occur between a user control and a web page is through events. With methods and properties, the user control reacts to a change made by the web page code. With events, the story is reversed: the user control notifies the web page about an action, and the web page code responds.

Creating a web control that uses events is fairly easy. In the following example, you'll see a version of the LinkMenu control that uses events. Instead of navigating directly to the appropriate page when the user clicks a button, the control raises an event, which the web page can choose to handle.

The first step to create this control is to define the events. Remember, to define an event, you must first choose an event signature. The .NET standard for events specifies that every event should use two parameters. The first one provides a reference to the control that sent the event, while the second incorporates any additional information. This additional information is wrapped into a custom EventArgs object, which inherits from the System.EventArgs class. (If your event doesn't require any additional information, you can just use the predefined EventArgs class, which doesn't contain any additional data. Many events in ASP.NET, such as Page.Load or Button.Click, follow this pattern.) You can refer to Chapter 4 for a quick overview of how to use events in .NET. The LinkMenu2 control uses a single event, which indicates when a link is clicked:

```
public partial class LinkMenu2 : System.Web.UI.UserControl
{
  public event EventHandler LinkClicked;
}
```

Example 12.6-1 Defining an Event

This code defines an event named LinkClicked. The LinkClicked event has the signature specified by the System.EventHandler delegate, which includes two parameters—the event sender and an ordinary EventArgs object. That means that any event handler you create to handle the LinkClicked event must look like this:

```
protected void LinkMenu_LinkClicked(object sender, EventArgs e){ ... }
```

Example 12.6-2 Defining an Event

This takes care of defining the event, but what about raising it? This part is easy. To fire the event, the LinkMenu2 control simply calls the event by name and passes in the two parameters, like this:

```
// Raise the LinkClicked event, passing a reference to
// the current object (the sender) and an empty EventArgs object.
LinkClicked(this, EventArgs.Empty);
```

The LinkMenu2 control actually needs a few more changes. The original version used the HyperLink control. This won't do, because the HyperLink control doesn't fire an event when the link is clicked. Instead, you'll need to use the LinkButton. The LinkButton fires the Click event, which the LinkMenu2 control can intercept, and then raises the LinkClicked event to the web page.

The following is the full user control code:

```
public partial class LinkMenu2 : System.Web.UI.UserControl
{
    public event EventHandler LinkClicked;
    protected void lnk_Click(object sender, EventArgs e)
    {
        // One of the LinkButton controls has been clicked.
        // Raise an event to the page.
        if (LinkClicked != null) { LinkClicked(this, EventArgs.Empty); }
    }
}
```

Example 12.6-3 Using an Event

Notice that before raising the **LinkClicked** event, the LinkMenu2 control needs to test the **LickedClick** event for a null reference. A null reference exists if no event handlers are attached to the event. In this case, you shouldn't try to raise the event, because it would only cause an error.

You can create a page that uses the LinkMenu2 control and add an event handler. Unfortunately, you won't be able to connect these event handlers using the Visual Studio Properties window, because the Properties

window won't show the custom events that the user control provides. Instead, you'll need to modify the **LinkMenu2** tag directly, as shown here:

```
<apress:LinkMenu2 id="Menu1" runat="server"
  OnLinkClicked="LinkClicked" />
```

and here's the event handler that responds in the web page:

Figure 12.6-1 Using LinkMenu2 User Control

Conceptually, this approach should give your web page more power to customize how the user control works. Unfortunately, that's not the case at the moment, because a key piece of information is missing. When the **LinkClicked** event occurs, the web page has no way of knowing what link was clicked, which prevents it from taking any kind of reasonable action. The only way to solve this problem is to create a more intelligent event that can transmit some information through event arguments. You'll see how in the next section.

12.7 Passing Information with Events

In the current LinkMenu2 example no custom information is passed along with the event. In many cases, however, you want to convey additional information that relates to the event. To do so, you need to create a custom class that derives from EventArgs.

The **LinkClickedEventArgs** class that follows allows the LinkMenu2 user control to pass the URL that the user selected through a Url property. It also provides a Cancel property. If set to true, the user control will stop its processing immediately. But if Cancel remains false (the default), the user control will send the user to the new page. This way, the user control still handles the task of redirecting the user, but it allows the web page to plug into this process and change it or stop it (for example, if there's unfinished work left on the current page).

To use this **EventArgs** class, you need to create a new delegate that represents the **LinkClicked** event signature. Here's what it looks like shown In Example 12.7-1

```
public class LinkClickedEventArgs : EventArgs
{
  private string url;
  public string Url
  {
     get { return url; }
     set { url = value; }
  }

  private bool cancel = false;
  public bool Cancel {
      get { return cancel; }
      set { cancel = value; }
   }

   public LinkClickedEventArgs(string url) { Url = url;  }
}
```

Example 12.7-1a Defining an Events with custom class.

```
public delegate void LinkClickedEventHandler(object sender,  LinkClickedEventArgs e);
```

Example 12.7-1b Defining the Event Handler

➢12.0 User Controls & Graphics

Both the **LinkClickedEventArgs** class and the **LinkClickedEventHandler** delegate should be placed in the App_Code directory. That way, these classes will be compiled automatically and made available to all web pages. Now you can modify the **LinkClicked** event to use the **LinkClickedEventHandler** delegate:

public event **LinkClickedEventHandler** LinkClicked;

Next, your user control code for raising the event needs to submit the required information when calling the event. But how does the user control determine what link was clicked? The trick is to switch from the **LinkButton.Click** event to the **LinkButton.Command** event. The Command event automatically gets the **CommandArgument** that's defined in the tag. So if you define your **LinkButton** controls like this:

```
<asp:LinkButton ID="lnkBooks" runat="server"
  CommandArgument="Menu2Host.aspx?product=Books" OnCommand="lnk_Command">Books
</asp:LinkButton><br />
<asp:LinkButton ID="lnkToys" runat="server"
  CommandArgument="Menu2Host.aspx?product=Toys" OnCommand="lnk_Command">Toys
</asp:LinkButton><br />
<asp:LinkButton ID="lnkSports" runat="server"
  CommandArgument="Menu2Host.aspx?product=Sports" OnCommand="lnk_Command">Sports
</asp:LinkButton><br />
<asp:LinkButton ID="lnkFurniture" runat="server"
  CommandArgument="Menu2Host.aspx?product=Furniture" OnCommand="lnk_Command">
Furniture</asp:LinkButton>
```

Example 12.7-1c Defining the presentation with LinkButton

you can pass the link along to the web page like this:
Code View: Scroll / Show All

```
LinkClickedEventArgs args = new
  LinkClickedEventArgs((string)e.CommandArgument);
LinkClicked(this, args);
```

Here's the complete user control code(Figure 12.7-1d). It implements one more feature. After the event has been raised and handled by the web page, the LinkMenu2 checks the Cancel property. If it's false, it goes ahead and performs the redirect using Reponse.Redirect().

```
public partial class LinkMenu2 : System.Web.UI.UserControl
{
  public event LinkClickedEventHandler LinkClicked;
  protected void lnk_Command(object sender, CommandEventArgs e)
  {
   // One of the LinkButton controls has been clicked.
   // Raise an event to the page.
   if (LinkClicked != null)
   {
     // Pass along the link information.
     LinkClickedEventArgs args =
              new LinkClickedEventArgs((string)e.CommandArgument);
     LinkClicked(this, args);
     // Perform the redirect.
     if (!args.Cancel){
     // Notice we use the Url from the LinkClickedEventArgs
     // object, not the original link. That means the web page
     // can change the link if desired before the redirect.
      Response.Redirect(args.Url);
    }
   }
  }
}
```

Example 12.7-1d Complete User Control Code

Finally, you need to update the code in the web page (where the user control is placed) so that its event handler uses the new signature. In the following code, the **LinkClicked** event handler checks the URL and allows it in all cases except one:

```
protected void LinkClicked(object sender, LinkClickedEventArgs e)
{
   if (e.Url == "Menu2Host.aspx?product=Furniture")
   {
      lblClick.Text = "This link is not allowed.";
      e.Cancel = true;
   }
   else
   {
      // Allow the redirect, and don't make any changes to the URL.
   }
}
```

Example 12.7-1e Updating the Web Page

If you click the Furniture link, you'll see the message shown in Figure 12.7-1.

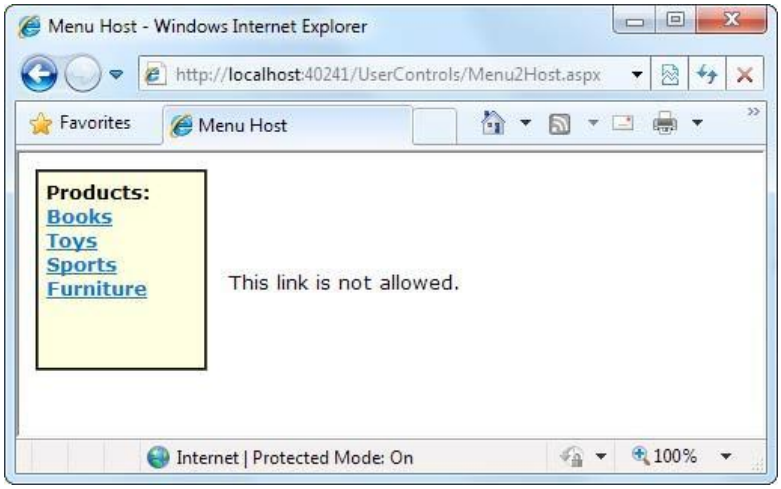

Figure12.7-1 Handling a User Control

The rest of the chapter is outside the scope for this course. But if you plan to develop professional web applications, you might as well read and apply the content.