

Container Controls and Naming Containers

Unlike some of the other container controls covered in this book, the `Panel` and `MultiView` controls are not *naming containers*. This is an important concept in ASP.NET. Container controls generate a special ID-based namespace for their child controls; this control namespace is called a naming container. This naming container guarantees that the ID of each of its children is unique within the page. When child controls are created at runtime (for instance, with the `Wizard` control in this chapter or the data controls covered in Chapter 10) the naming container of the parent is combined with the child control's `Id` to create the control's `UniqueId` property.

The fact that the `MultiView` is not a naming container means that the `Id` of every control contained within all the `View` controls *must* be unique. That is, you cannot have a control named `btnNext` in `view1` and `view2`. This also means that controls in each view are always available for programmatic manipulation. However, the flipside to this functionality is that the view state for all controls in a `MultiView`, regardless of whether they are visible or not, is posted back and forth for every request that uses that `MultiView`.

Wizard Control

Web applications often require some type of wizard-style interface that leads the user through a series of step-by-step forms. A wizard is thus used to retrieve information from the user via a series of discrete steps; each step in the wizard asks the user to enter some subset of information. Some examples of Web-based wizards are the typical user registration procedure for a site or the checkout procedure in a Web store. For instance, the Amazon.com checkout procedure (see Figure 4.7) is not called a wizard but has the features of one: discrete steps separate from the rest of the application/site, an indication of the current step, and a way to move to the next step.

In regular Windows applications, wizards tend to be *modal* in that the user cannot do any other processing while the wizard is active: The user can only move forward, backward, finish, or possibly jump to some other step in the wizard. As well, Windows-based wizards are *pipelines* in that a wizard is a chain of processes in which there is only one entrance and where the output of each step in the process is the input to the next step.

Implementing wizards in Web applications poses several problems. A Windows application can strictly control how a wizard is launched, and can easily ensure users start and exit the wizard in the appropriate way. This is much harder to implement in a Web environment. For instance, a user could bookmark an intermediate step in the wizard and try to return to it at some point in the future; a Web-based wizard must thus be able to maintain the pipeline nature of the wizard and ensure that the user only ever starts the wizard at the first step.

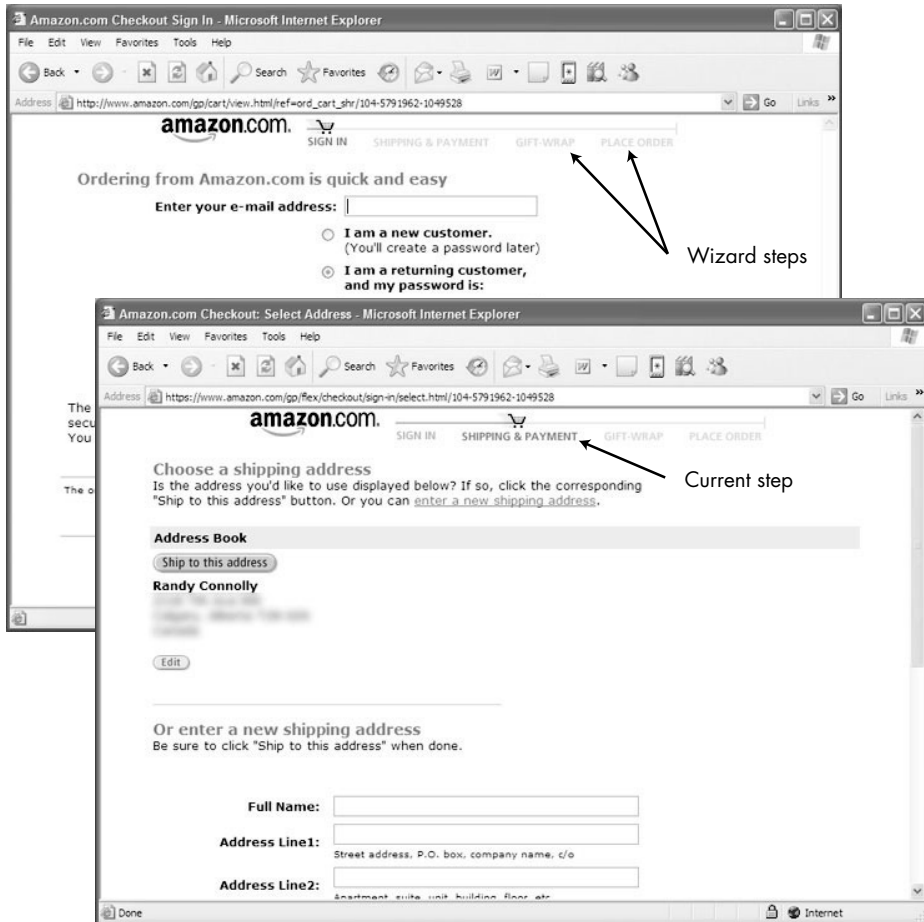


Figure 4.7 Amazon checkout process

Maintaining state information in a multipage Web wizard can also be tricky. Because the intent of a wizard is to gather information from the user in a series of discrete steps, a Web wizard needs to be able to pass information gathered in previous steps onto the next step. This requires repetitive coding for the retrieval and verification of the previous step's data. As well, there is typically a great deal of repetitive code for the handling of navigation within the wizard, as shown in Figure 4.8.

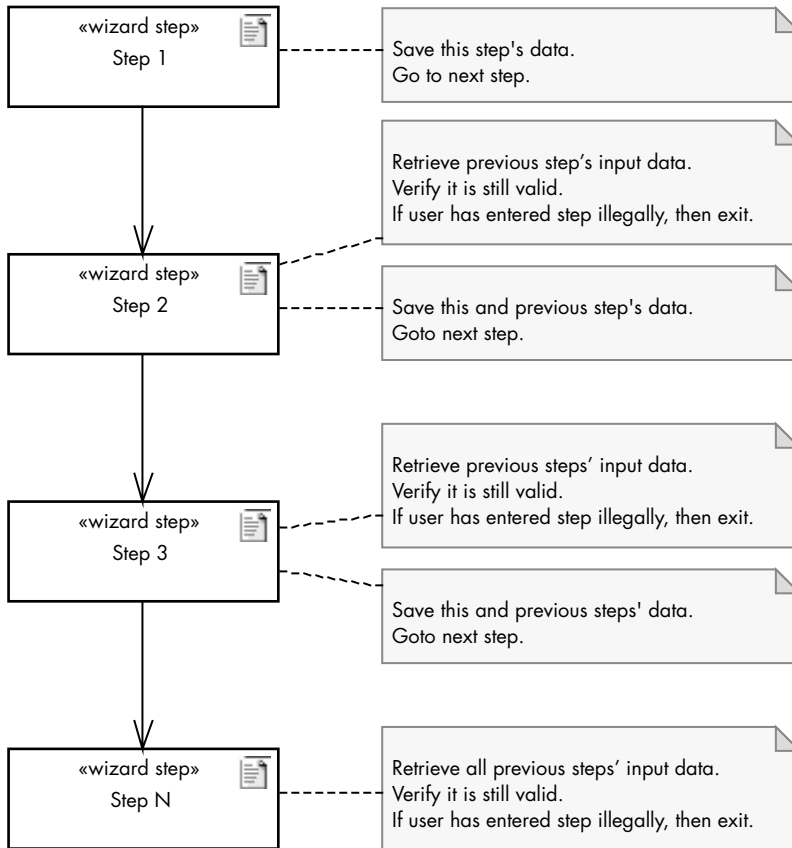


Figure 4.8 Typical Web wizard processing

In ASP.NET 1.1, wizards were implemented using either multiple pages with plenty of duplicate, boiler-plate code (as in Figure 4.8), or within a single page with numerous `Panel` controls, or by dynamically adding user controls to a single page based on the wizard step. The multiple panels within a single page approach eliminates the duplicated code problem of numerous pages; unfortunately, it comes at the cost of plenty of awkward code to toggle the visibility of the panels and buttons and to track the wizard state. I was involved with an ASP.NET 1.1 wizard project with nine discrete steps and more than 20 possible panels all contained within a single ASP.NET page; the conditional logic just for handling the panels and button controls was enough to make even the most hardened ASP spaghetti-code veteran wince in pain.

The new `Wizard` control in ASP.NET 2.0 makes the process of creating a Web wizard significantly easier. This control provides a simpler mechanism for building

steps, adding new steps, or reordering the steps. The wizard's navigation can be either linear or nonlinear and can be implemented declaratively without coding. The `Wizard` control eliminates the need to manage the persistence of your data across pages because the control itself maintains state while the user completes the various steps. As well, the `Wizard` control is fully supported by the Visual Studio designer, thus significantly easing the process of creating and modifying your wizard's steps.

Table 4.6 lists some of the notable properties of the `Wizard` control. There are quite a few additional properties not listed in this table; you can view the MSDN documentation for a complete listing.

Table 4.6 Unique Properties of the Wizard Control

Property	Description
<code>ActiveStep</code>	Retrieves the <code>WizardStepBase</code> object that is currently displayed.
<code>ActiveStepIndex</code>	The index of the <code>WizardStepBase</code> object currently displayed. This property can be used to programmatically set the step to be displayed at runtime.
<code>CancelButtonImageUrl</code>	Specifies the URL of the image displayed for the Cancel button.
<code>CancelButtonText</code>	Specifies the text caption for the Cancel button.
<code>CancelButtonType</code>	Specifies the type of Cancel button. Possible values are defined by the <code>ButtonType</code> enumeration (<code>Button</code> , <code>Image</code> , <code>Link</code>).
<code>CancelDestinationPageUrl</code>	Specifies the URL to redirect to when the user clicks the Cancel button.
<code>DisplayCancelButton</code>	Specifies whether the wizard should display the Cancel button (default is <code>false</code>).
<code>DisplaySideBar</code>	Specifies whether the sidebar area of the wizard should be displayed. The default is <code>true</code> .
<code>FinishCompleteButtonImageUrl</code>	Specifies the URL of the image displayed for the Complete button for the Finish step.
<code>FinishCompleteButtonText</code>	Specifies the text caption for the Complete button of the Finish step.

Table 4.6 Unique Properties of the Wizard Control (*continued*)

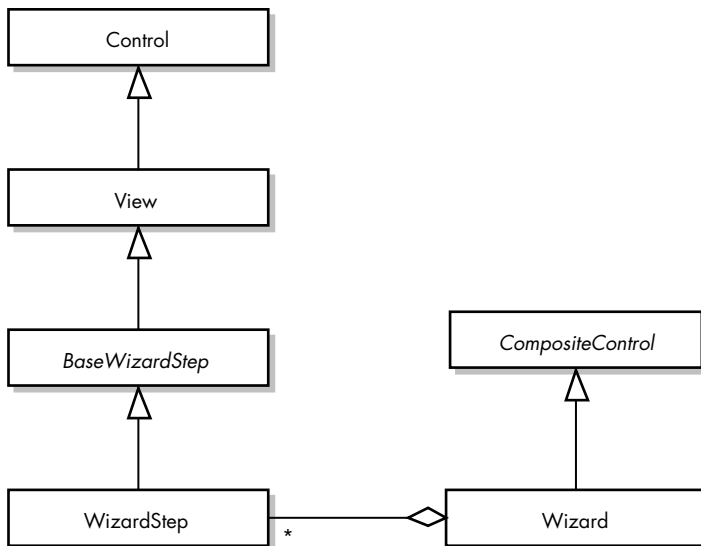
Property	Description
<code>FinishCompleteButtonType</code>	Specifies the type of the Complete button of the Finish step. Possible values are defined by the <code>ButtonType</code> enumeration (<code>Button</code> , <code>Image</code> , <code>Link</code>).
<code>FinishDestinationPageUrl</code>	Specifies the URL to redirect to when the user clicks the Finish button.
<code>FinishPreviousButtonImageUrl</code>	Specifies the URL of the image displayed for the Previous button of the Finish step.
<code>FinishPreviousButtonText</code>	Specifies the text caption for the Previous button of the Finish step.
<code>FinishPreviousButtonType</code>	Specifies the type of the Previous button of the Finish step. Possible values are defined by the <code>ButtonType</code> enumeration (<code>Button</code> , <code>Image</code> , <code>Link</code>).
<code>HeaderStyle</code>	Retrieves the collection of style properties for the Header area of the wizard.
<code>HeaderText</code>	The text caption to display in the header area of the wizard.
<code>SkipLinkText</code>	Specifies the alternate text for a hidden image that allows screen readers to skip the content in the sidebar area. Used for accessibility.
<code>StartNextButtonImageUrl</code>	Specifies the URL of the image displayed for the Next button on the Start step.
<code>StartNextButtonText</code>	Specifies the text caption for the Next button of the Start step.
<code>StartNextButtonType</code>	Specifies the type of the Next button of the Start step. Possible values are defined by the <code>ButtonType</code> enumeration (<code>Button</code> , <code>Image</code> , <code>Link</code>).
<code>StepNextButtonImageUrl</code>	Specifies the URL of the image displayed for the Next button.
<code>StepNextButtonText</code>	Specifies the text caption for the Next button.

Table 4.6 Unique Properties of the Wizard Control (*continued*)

Property	Description
<code>StepNextButtonType</code>	Specifies the type of the Next button. Possible values are defined by the <code>ButtonType</code> enumeration (<code>Button</code> , <code>Image</code> , <code>Link</code>).
<code>StepPreviousButtonImageUrl</code>	Specifies the URL of the image displayed for the Previous button.
<code>StepPreviousButtonText</code>	Specifies the text caption for the Previous button.
<code>StepPreviousButtonType</code>	Specifies the type of the Previous button. Possible values are defined by the <code>ButtonType</code> enumeration (<code>Button</code> , <code>Image</code> , <code>Link</code>).

Using the Wizard Control

The wizard control is composed of a number of separate `WizardStep` controls. Each `WizardStep` control represents a single step in the wizard process. The `WizardStep` control inherits from an abstract class called `BaseWizardStep`, which in turn inherits from the `View` control covered in the previous section (see Figure 4.9).

**Figure 4.9** Object model for Wizard control

Like the `View` control, the `WizardStep` control is a container for HTML markup and other ASP.NET controls. The parent `Wizard` control manages which `WizardStep` to display and helps maintain the data collected in each step. The following example shows the markup for a simple two-step wizard with some sample content in each of the two steps.

```
<asp:Wizard ID="myWizard" Runat="server"
  HeaderText="Sample Wizard">

  <WizardSteps>

    <asp:WizardStep ID="WizardStep1" runat="server"
      Title="Step 1">
      <b>Step One</b><br />
      <asp:Label ID="label1" runat="server">Email</asp:Label>
      <asp:TextBox ID=" txtEmail " runat="server" /><br />
      <asp:Label ID="label2" runat="server">
        Password
      </asp:Label>
      <asp:TextBox ID="txtPassword" runat="server" />
    </asp:WizardStep>

    <asp:WizardStep ID="WizardStep2" runat="server"
      Title="Step 2">
      <b>Step Two</b><br />
      <asp:Label ID="label3" runat="server">
        Shipping
      </asp:Label>
      <asp:DropDownList ID="drpShipping" runat="server">
        <asp:ListItem>Air Mail</asp:ListItem>
        <asp:ListItem>Fed Ex</asp:ListItem>
      </asp:DropDownList>
    </asp:WizardStep>

  </WizardSteps>

</asp:Wizard>
```

Figure 4.10 illustrates how this example wizard appears in the browser.

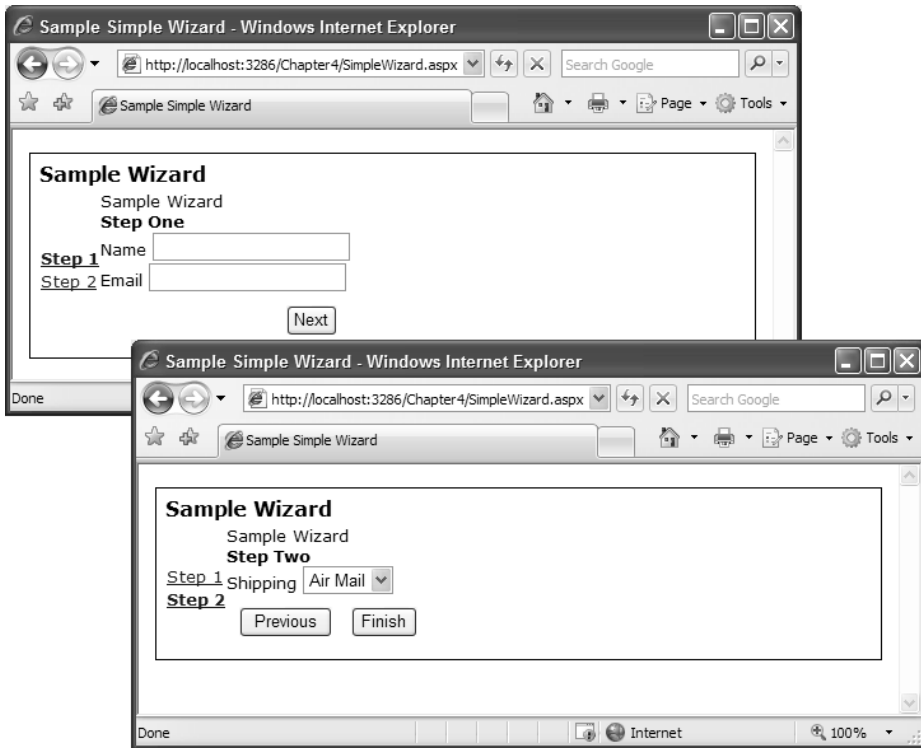


Figure 4.10 Simple wizard in the browser

Table 4.7 lists the notable properties of the `wizardStep` control.

Table 4.7 Notable Properties of the WizardStep Control

Property	Description
<code>AllowReturn</code>	Indicates whether the user is allowed to return to this step from another subsequent step.
<code>Name</code>	The name of the step.
<code>StepType</code>	The type of navigation interface to display for the step. Possible values are described by the <code>wizardStepType</code> enumeration (Auto, Complete, Finish, Start, Step).
<code>Title</code>	The title of the step.

Understanding the Layout of the Wizard Control

Like the `Calendar` control covered in the previous chapter, the `Wizard` control can be fully customized in terms of its appearance and behavior via properties, style elements, and template elements. **Style elements** are used to specify the font, border, color, size, or CSS class of the wizard part referenced by the style element. **Template elements** are used to define a custom layout for a given element. Table 4.8 lists the available style and templates for the `Wizard` control.

Table 4.8 Style and Template Elements for the Wizard Control

Name	Description
<code>FinishNavigationTemplate</code>	The layout template for the navigation area for the Finish step.
<code>HeaderStyle</code>	The style properties for the Header area of the wizard.
<code>HeaderTemplate</code>	The template used to define the custom content that is displayed in the header area of the wizard.
<code>NavigationButtonStyle</code>	The style properties for the buttons used in the navigation area of the wizard.
<code>NavigationStyle</code>	The style properties for the navigation area of the wizard.
<code>SideBarButtonStyle</code>	The style properties for the sidebar area buttons.
<code>SideBarStyle</code>	The style properties for the sidebar area of the wizard.
<code>SideBarTemplate</code>	The template used to define the custom content that is displayed in the sidebar area of the wizard.
<code>StartNavigationTemplate</code>	The template used to define the custom content that is displayed in the navigation area of the wizard for the Start step.
<code>StepNavigationTemplate</code>	The template used to define the custom content that is displayed in the navigation area of the wizard.
<code>StepStyle</code>	The style properties for the wizard step area.

To help clarify the relationship between the different style and template elements and the actual `Wizard` control layout areas, Figure 4.11 illustrates these layout areas.

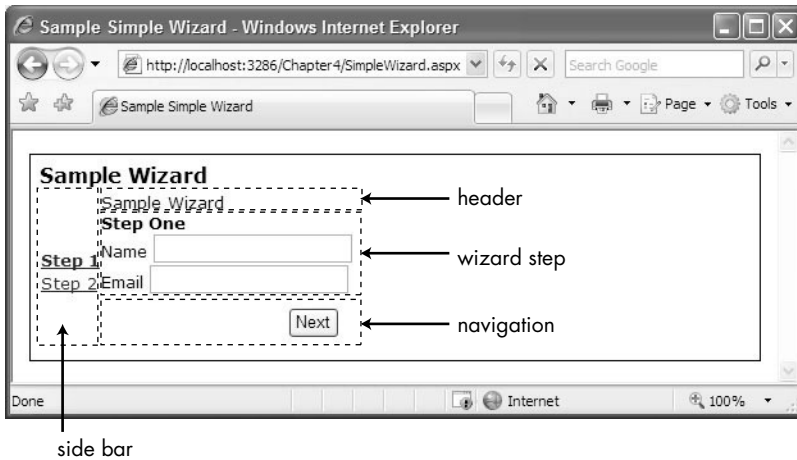


Figure 4.11 Wizard layout areas

The header area in Figure 4.11 is set via the `HeaderText` property of the `Wizard` control. The sidebar area contains a series of links for navigating to each wizard step. This area is optional and can be hidden via the `DisplaySideBar` property of the wizard control. The sidebar can be styled via `SideBarTemplate`, `SideBarStyle`, and `SideBarButtonStyle` (covered shortly). The navigation area contains navigation buttons (the user interface for these buttons can also be an image or a hyperlink) for moving to the next or to the previous step. You have complete control over the appearance of both the next and previous step buttons. You can also add navigation elements for finishing the wizard or canceling the wizard (covered shortly). The wizard control also allows you to customize the appearance of the navigation area for both the first and the final `WizardStep`.

Finally, the wizard step area in Figure 4.11 contains the active `WizardStep` control. A `WizardStep` control can be one of five different types (set via the `StepType` property): `Auto`, `Start`, `Step`, `Finish`, and `Complete`. The default `StepType` is `Auto`, which means the navigation interface for the step is determined by the order in which the step is declared. In a wizard with three steps, the first step is a `Start` step, the second step is a `Step` step, and the third is a `Finish` step. `Start` and `Finish` steps display a different set of navigation buttons than the `Step` step: `Start` does not display a previous button, whereas `Finish` does not display a next button. The `Complete` step is a special type of optional step. It is always displayed last, it collects no data, and contains no navigation. It is useful if you want your wizard to display some type of explicit indication that the wizard is completed.

Customizing the Wizard

The Wizard control provides a large number of avenues for customization. You can customize each of the areas shown in Figure 4.11. The following sections discuss and illustrate how to style and customize the header, sidebar, wizard step, and navigation areas.

CORE NOTE

Although the next sections describe how to use the various style and template elements of the Wizard control, you may find it easier to use the Visual Studio designer to customize these elements, as shown in Figure 4.12.

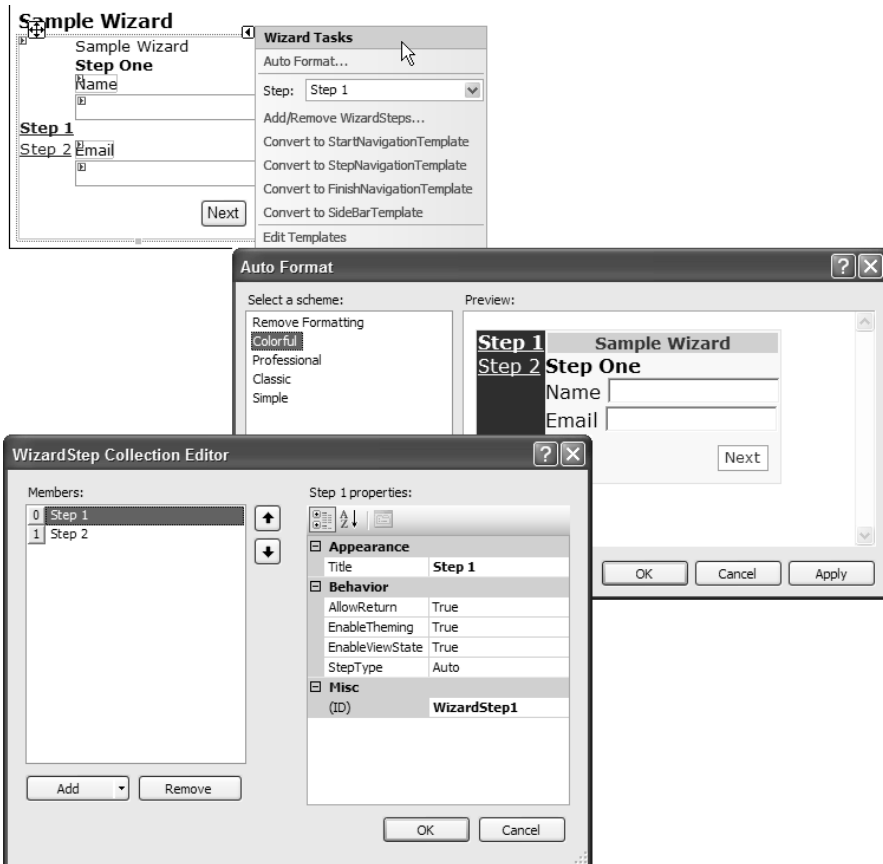


Figure 4.12 Customizing the Wizard with the Visual Studio designer

Styling the Header Area

The wizard header can be customized via the templates `HeaderStyle` and the `HeaderTemplate`. The `HeaderStyle` template allows you to specify the font, color, border, and CSS class to be used for displaying the text in the `HeaderText` property. Like all template properties, you can set their values within its template tag, or within the parent control via hyphen notation. For instance:

```
<asp:Wizard ID="myWizard" runat="server" HeaderText="Checkout">
  <HeaderStyle BackColor="#CC9966" Font-Bold="true"
    Font-Size="Large"/>
</asp:Wizard>
```

or

```
<asp:Wizard ID="myWizard" runat="server" HeaderText="Checkout"
  HeaderStyle-BackColor="#CC9966" HeaderStyle-Font-Bold="true"
  HeaderStyle-Font-Size="Large">
</asp:Wizard>
```

As mentioned in the previous chapter, you may want to only specify a CSS class within a template rather than specifying appearance properties. Chapter 6 discusses the pros and cons of using CSS versus using appearance properties.

Customizing the Header Area

The `HeaderTemplate` lets you fully customize not only the formatting but also the content of the header. For instance, the following example (the result can be seen in Figure 4.13) shows how to use information from the `Wizard` control and the current `WizardStep` control to display a more useful wizard header. Notice that it uses inline expressions (covered back on page 40 of Chapter 1) to display the current value of various `Wizard` properties in the page.

```
<asp:Wizard ID="myWizard" runat="server"
  HeaderText="Checkout" ... >

  <HeaderTemplate>
    <div style="margin: 5px 5px 5px 5px">
      <i><%= myWizard.HeaderText %>
        Step <%= myWizard.ActiveStepIndex+1 %> of 2</i><br />
      <b><%= myWizard.ActiveStep.Title%></b>
    </div>
  </HeaderTemplate>

  <WizardSteps>
    <asp:WizardStep ID="WizardStep1" runat="server"
      Title="Login">
    ...
```

```

    <asp:WizardStep ID="WizardStep2" runat="server"
        Title="Address">
        ...
    </WizardSteps>
</asp:Wizard>

```

You could also use the `HeaderTemplate` to display an image. The following example demonstrates how this might work. The filename for the image to be displayed in the header is constructed based on the current step index. Notice that it also supplies an `alt` attribute based on the title. The result in the browser can be seen in Figure 4.13, which shows both the text and image header approaches.

```

<HeaderTemplate>
    <div style="margin: 5px 5px 5px 5px">
        <img src=
            'images/title_checkout_step
                <%= myWizard.ActiveStepIndex+1 %>.gif'
                alt='Checkout <%= myWizard.ActiveStep.Title%>' />
        </div>
</HeaderTemplate>

```

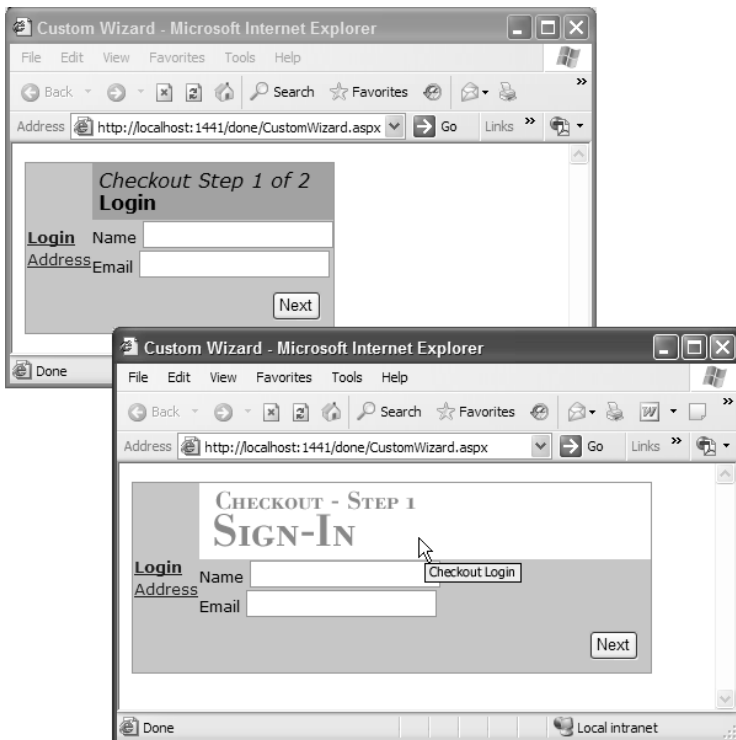


Figure 4.13 Using a header template

CORE NOTE

For usability reasons, you should ensure that the total height of the wizard remains constant for each step (thus emulating the behavior of a Windows wizard). Nothing is more annoying than having the navigation button's position jump up and down on the screen as you move through the steps of the wizard!



Styling the Sidebar Area

As mentioned earlier, the sidebar area contains a series of links for navigating to each wizard step. This area is optional and can be hidden by setting the `DisplaySideBar` property of the `Wizard` control to `false`. If you want to ensure that the user only moves linearly through the wizard, it might be best to hide the sidebar area or to disable the links within it.

The sidebar can be styled via the `SideBarStyle` and `SideBarButtonStyle`. The `SideBarStyle` template allows you to specify the font, color, border, and CSS class to be used for displaying the content in the sidebar. The `SideBarButtonStyle` template allows you to specify the font, color, border, and CSS class to be used for displaying the links within the sidebar. The following example illustrates a sample `SideBarStyle` template.

```
<SideBarStyle CssClass="sidebar" />
```

Customizing the Sidebar Area

Just as with the header, you can customize (albeit only partially) the sidebar by using the `SideBarTemplate`. With the `SideBarTemplate`, you *must* use a `DataList` control (covered in detail in Chapter 10, page 587) to contain the links. You can replace the links with any other control that implements the `IButtonControl` interface, such as `ImageButton`, `LinkButton`, or `Button`. For instance, the following sample illustrates how to display buttons rather than links in the sidebar.

```
<SideBarTemplate>
  <asp:DataList ID="SideBarList" runat="server">

    <ItemTemplate>
      <asp:Button ID="SideBarButton" runat="server"
        BackColor="#CCCC99" />
    </ItemTemplate>

    <SelectedItemTemplate>
      <asp:Button ID="SideBarButton" runat="server"
        BackColor="#CCCCCC" />
    </SelectedItemTemplate>
  </asp:DataList>
</SideBarTemplate>
```

```
</asp:DataList>
</SideBarTemplate>
```

Notice how this example uses the `SelectedItemTemplate` of the `DataList` to format the button that corresponds to the currently active step in the wizard differently than the other step buttons.

CORE NOTE

The `DataList` used in the `SideBarTemplate` must follow a few rules. It must have its `ID` property set to `SideBarList`. Within the `ItemTemplate`, `SelectedItemTemplate`, or `AlternatingItemTemplate`, it must contain a control that implements the `IButtonControl`. Also, this `IButtonControl` control must have its `ID` property set to `SideBarButton`.

You can also display images in the sidebar by using the `ImageButton` control. However, to get a unique image for each wizard step, you need to add some programming logic. As well, you need some way to associate the image filenames with the wizard steps. Perhaps the easiest way to do this is to include the wizard step title in the image filename. For instance, let's define a wizard step as follows:

```
<asp:WizardStep ID="WizardStep1" runat="server" Title="Login">
```

In this case, you need to have an image with the title `Login` in the filename, such as `Login.gif`, `step_Login.gif`, or `checkoutStepLoginSelected.gif`. If you have your images thus named, you only need to define a method that constructs the image filename. The following example shows two such methods. The first returns the image filename for the image that to be displayed in the `ItemTemplate`; the second returns the filename for the image to be displayed in the `SelectedItemTemplate`.

```
public object GetStepImage(string title)
{
    return "images/sidebar_" + title + ".gif";
}
public object GetSelectedStepImage(string title)
{
    return "images/sidebar_" + title + "_selected.gif";
}
```

These methods are called when you set the `ImageUrl` property of the `ImageButton` control in the `SideBarTemplate`. For instance, the following illustrates the use of these methods within something called data-binding expressions, which are covered in detail in Chapter 8.

```

<SideBarTemplate>
  <asp:DataList ID="SideBarList" runat="server">
    <ItemTemplate>
      <asp:ImageButton ID="SideBarButton" runat="server"
        ImageUrl='<%=# GetStepImage( (string)Eval("Title") )%>' />
    </ItemTemplate>
    <SelectedItemTemplate>
      <asp:ImageButton ID="SideBarButton" runat="server"
        ImageUrl='<%=# GetSelectedStepImage(
          (string)Eval("Title") )%>' />
    </SelectedItemTemplate>
  </asp:DataList>
</SideBarTemplate>

```

The `DataList` in the sidebar is bound to a collection of `WizardStepBase` objects. Thus, this example passes the contents of the `Title` property (which is defined as an object and hence must be cast to a `string`) of the current `WizardStepBase` object in the collection. The result in the browser might look like that shown in Figure 4.14.

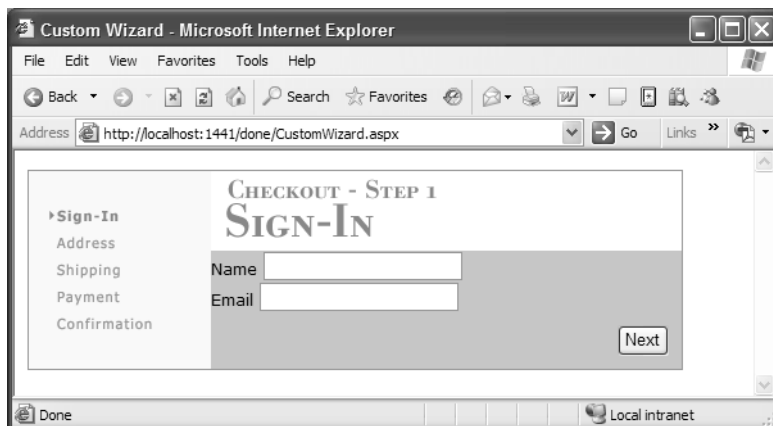


Figure 4.14 Using images in the sidebar

Styling the Wizard Step Area

The content of each wizard step is defined within each step's `WizardStep` element. However, you can specify a consistent style for each step in the wizard by using the `StepStyle` template. Like the other style templates, it allows you to specify the font, color, border, and CSS class to be used for the content of each step. For instance, if you want to provide some padding (i.e., space between the contents of the wizard step and its boundaries) to each wizard step, you could define a CSS style such as

```
<style type="text/css">
  .wizardStepContent { padding: 5px; ... }
</style>
```

You could then use that CSS class via the `StepStyle` template.

```
<StepStyle CssClass="wizardStepContent" ... />
```

Customizing the Wizard Step Area

Earlier, you saw how the wizard contains a collection of `WizardStep` elements. If you want to have full control over the look and behavior of a wizard step (including the ability to have a completely different navigation system), you could use a `TemplatedWizardStep` instead of a `WizardStep`, as in

```
<asp:Wizard ID="myWizard" Runat="server"
  HeaderText="Sample Wizard">
  <WizardSteps>

    <asp:WizardStep ID="ws1" runat="server" Title="Step 1">
    </asp:WizardStep>
    ...
    <asp:TemplatedWizardStep ID="tws1" runat="server"
      Title="Step 2">
      <ContentTemplate>
        ...
      </ContentTemplate>
      <CustomNavigationTemplate>
        ...
      </CustomNavigationTemplate>
    </asp:TemplatedWizardStep>

  </WizardSteps>
</asp:Wizard>
```

Styling the Navigation Area

Like the other areas of the wizard, the navigation area of the wizard can be fully styled. There are a number of different navigation style tags. With these tags, you can

- Style the entire navigation area (`NavigationStyle`)
- Style all possible buttons (`NavigationButtonStyle`)
- Style the Cancel button for the wizard (`CancelButtonStyle`)
- Style the Next button for the starting wizard step (`StartNextButtonStyle`)
- Style the Complete button for the finished wizard step (`FinishCompleteButtonStyle`)
- Style the Previous button for the finished wizard step (`FinishPreviousButtonStyle`)

- Style the Next button for the regular wizard steps (`StepNextButtonStyle`)
- Style the Previous button for the regular wizard steps (`StepPrevious-ButtonStyle`)

Of course, for most situations, you would probably want to have the same style for all the buttons in the wizard regardless of whether they are Next or Previous buttons. Thus, for most situations, you could just use the `NavigationStyle` and `NavigationButtonStyle` to consistently style your buttons, as in the following:

```
<NavigationStyle BackColor="white" VerticalAlign="Bottom" />
<NavigationButtonStyle BackColor="#FFFFCC" ForeColor="#666633" />
```

Customizing the Navigation Area

You can also fully customize how the navigation area appears with template tags. There is a navigation template for the Start, Finish, and regular Step step types (`StartNavigationTemplate`, `FinishNavigationTemplate`, and `StepNavigationTemplate`). With these templates, you can completely control what is displayed within the navigation area. For instance, if you decide that you do not want the standard Next and Previous buttons, but would prefer to use `LinkButton` or `ImageButton` controls, you can do so via these templates.

There are some requirements that must be followed with these navigation templates. The `StepNavigationTemplate` *must* contain two `IButtonControl` controls (i.e., `LinkButton`, `ImageButton`, or `Button`). One of these `IButtonControl` controls *must* have its `CommandName` property set to `MoveNext` and the other *must* have its `CommandName` property set to `MovePrevious` to have the wizard navigation work. The `StartNavigationTemplate` *must* have a single `IButtonControl` control with a `CommandName` of `MoveNext`. The `FinishNavigationTemplate` *must* have two `IButtonControl` controls with their `CommandName` properties set to `MovePrevious` and `MoveComplete`.

The following example illustrates the use of the three custom navigation templates. It illustrates how to use images rather than buttons in the navigation area for all three steps. The result can be seen in Figure 4.15.

```
<NavigationStyle BackColor="white" VerticalAlign="Bottom" />
<NavigationButtonStyle BackColor="#FFFFCC" ForeColor="#666633" />

<StartNavigationTemplate>
  <asp:Panel ID="panStart" runat="server"
    CssClass="wizardNavContent">
    <asp:ImageButton runat="server" ID="imgStart"
      ImageUrl="~/images/button_checkout_start.gif"
      CommandName="MoveNext" AlternateText="Sign-In" />
  </asp:Panel>
</StartNavigationTemplate>
```

```

<StepNavigationTemplate>
  <asp:Panel ID="panStep" runat="server"
    CssClass="wizardNavContent">
    <asp:ImageButton runat="server" ID="imgPrev"
      ImageUrl="~/images/button_checkout_previous.gif"
      CommandName="MovePrevious" AlternateText="Previous" />
    <asp:ImageButton runat="server" ID="imgNext"
      ImageUrl="~/images/button_checkout_next.gif"
      CommandName="MoveNext" AlternateText="Next" />
  </asp:Panel>
</StepNavigationTemplate>

<FinishNavigationTemplate>
  <asp:Panel ID="panFinish" runat="server"
    CssClass="wizardNavContent">
    <asp:ImageButton runat="server" ID="imgPrevFin"
      ImageUrl="~/images/button_checkout_previous.gif"
      CommandName="MovePrevious" AlternateText="Previous" />
    <asp:ImageButton runat="server" ID="imgFinish"
      ImageUrl="~/images/button_checkout_finish.gif"
      CommandName="MoveComplete"
      AlternateText="Make Payment" />
  </asp:Panel>
</FinishNavigationTemplate>

```

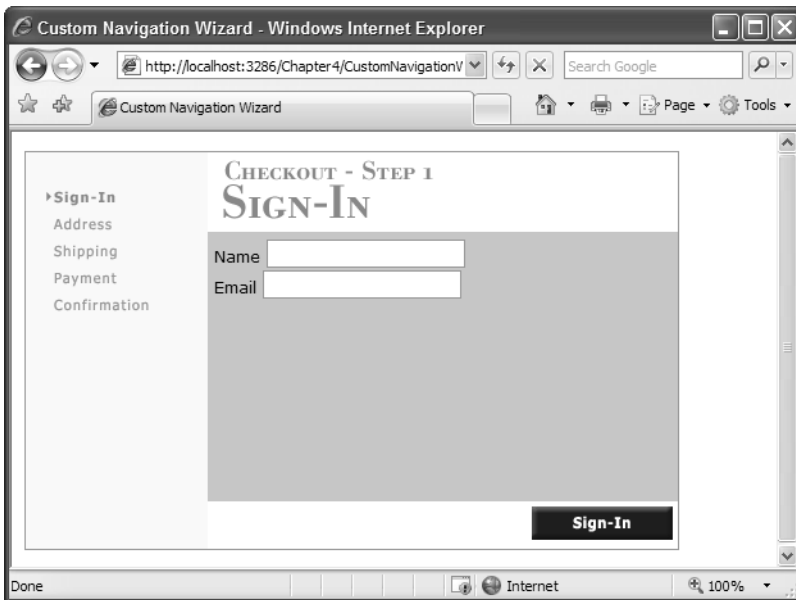


Figure 4.15 Customizing the navigation area

Wizard Event Handling

One of the great things about the `wizard` control is that it dramatically reduces the amount of coding necessary to implement a step-by-step series. You no longer have to worry, for instance, about coding the navigation logic or the maintenance of the state between steps. Yet, even with the `wizard` control, some coding is required. After the `Finish` step, you need to code the processing and (probably) the persistence of the data gathered in the wizard steps. Alternately, you may want to persist the data gathered at each step, and thus need to write code that executes before the wizard moves to the next step. Another reason you may need to write code for the wizard is to override the default linear progression through the wizard steps.

The `Wizard` control supports several unique events (see Table 4.9). Five of the events are associated with click events on the navigation controls. The other event (`ActiveStepChanged`) is triggered when the active view (i.e., the current step) changes. Note that the click events for the navigation buttons are handled *before* the `ActiveStepChanged` handler. This allows you to potentially prevent the view from changing (and thus prevent the `ActiveStepChanged` handler from being called). Thus, you can perform some type of server-side validation on the data gathered in a given wizard step and cancel the step change if the data is not valid.

Table 4.9 Events of the Wizard Control

Event	Description
<code>ActiveStepChanged</code>	Raised on a postback when the user switches to a new step.
<code>CancelButtonClick</code>	Raised on a postback when the user clicks the Cancel button.
<code>FinishButtonClick</code>	Raised on a postback when the user clicks the Finish button.
<code>NextButtonClick</code>	Raised on a postback when the user clicks the Next button.
<code>PreviousButtonClick</code>	Raised on a postback when the user clicks the Previous button.
<code>SideBarButtonClick</code>	Raised on a postback when the user clicks one of the sidebar buttons.

NextButtonClick Event Handler

Because the same navigation click handler is used regardless of the step, the handler typically needs some type of conditional logic using the `ActiveStepIndex` of the wizard. For instance, the following `NextButtonClick` event handler illustrates how you might perform different validation for the different wizard steps.

```
protected void myWizard_NextButtonClick(object sender,
    WizardNavigationEventArgs e)
{
    if (myWizard.ActiveStep == WizardStep1)
    {
        string email = txtEmail.Text;
        string passwd = txtPassword.Text;

        // Check database to see if this user exists
        bool okay = UserBusObject.CheckIfOkay(email, passwd);
        if ( ! okay )
        {
            myLabel.Text += "User does not exist<br/>";
            // Cancel the move to the next wizard step
            e.Cancel = true;
        }
    }
    else if (myWizard.ActiveStep == WizardStep2)
    {
        // Validation for step 2 goes here
    }
    // etc
}
```

This example uses some type of business object, which presumably does some type of database lookup on the email and password values entered by the user. If the email and password do not exist, the move to the next wizard step is cancelled (by setting the `Cancel` property of the passed-in `WizardNavigationEventArgs` object). Notice as well how the `ActiveStep` property of the wizard is compared to the various wizard step objects. An alternative way of doing this comparison is on the step index rather than the step objects themselves, as in

```
myWizard.ActiveStepIndex ==
    myWizard.WizardSteps.IndexOf(WizardStep1)
```

ActiveStepChanged Event Handler

The `ActiveStepChanged` handler is typically used to modify the step order. For instance, in the following example, the `ActiveStepChanged` handler checks if the add address wizard step (which is step 2) is unnecessary; if it is unnecessary, it skips the step and moves to the third step in the wizard.

```
protected void myWizard_ActiveStepChanged(object sender,
    EventArgs e)
{
    // If we are on the first step, then ...
    if (myWizard.ActiveStep == WizardStep1)
    {
```

```
// ... check the database to see if we need to
// enter an address
if ( UserBusObject.NeedAddress(
    txtEmail.Text,txtPassword.Text) )
{
    myWizard.MoveTo(WizardStep2);
}
else
{
    myWizard.MoveTo(WizardStep3);
}
}
}
```

FinishButtonClick Event Handler

The handler for the Finish button is where you perform the final processing on the data gathered in the wizard. As well, if your wizard has a Complete step, you can then populate its content in this handler. The following example illustrates a sample FinishButtonClick event handler.

```
protected void myWizard_FinishButtonClick(object sender,
    WizardNavigationEventArgs e)
{
    // Gather the data from the various steps
    string email = txtEmail.Text;
    ...
    // Gather the data from the various steps
    ...
    // Now fill the content of the confirmation wizard step
    myLabel.Text += "FinishButtonClick called<br/>";
    Label labConfirm =
        (Label)myWizard.FindControl("labConfirmation");
    labConfirm.Text = "Order has been processed for " +
        txtEmail.Text;
}
```

Notice that in this example the `FindControl` method of the `Wizard` control is used to reference a control within the confirmation wizard step. This is necessary because the confirmation page has not yet been displayed; as a result, you cannot simply directly reference the control like you did with controls on the current or previously visited steps.