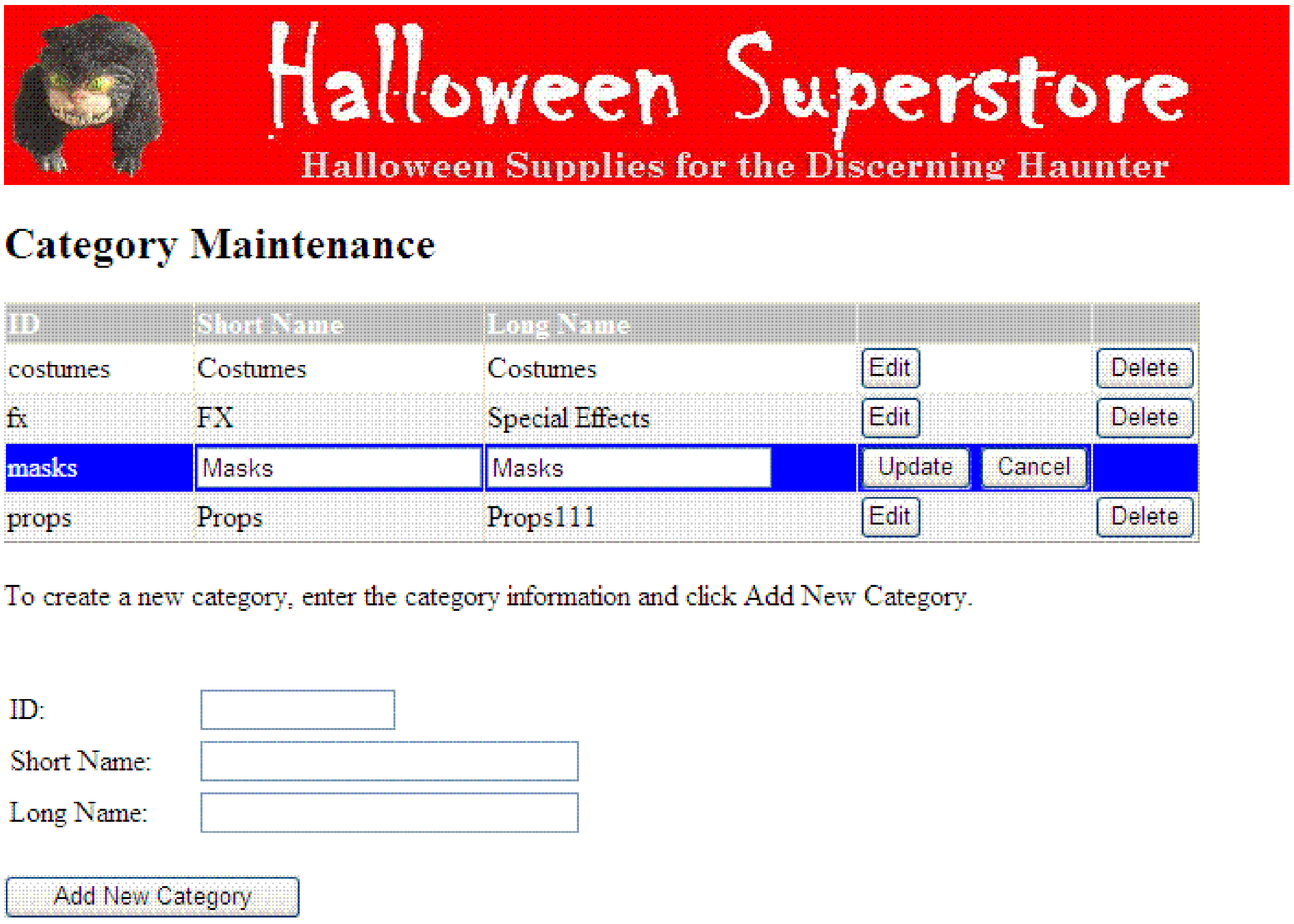
Lecture 10

GridView control

GridView control lets you display the data from a data source in the rows and columns of a table. It includes many advanced features, such as automatic paging and sorting. It lets you update and delete data with minimal *C#* code. And its appearance is fully customizable.

A *GridView* control that provides for updating a table



Category Maintenance

ID	Short Name	Long Name		
costumes	Costumes	Costumes	Edit	Delete
fx	FX	Special Effects	Edit	Delete
masks	Masks	Masks	Update Cancel	
props	Props	Props111	Edit	Delete

To create a new category, enter the category information and click Add New Category.

ID:	
Short Name:	
Long Name:	

The aspx code for the *GridView* control shown above

```
<asp:GridView ID="GridView1" runat="server"
        AutoGenerateColumns="False" DataKeyNames="CategoryID"
        DataSourceID="SqlDataSource1" ForeColor="Black"
        OnRowDeleted="GridView1_RowDeleted" OnRowUpdated="GridView1_RowUpdated">
    <Columns>
        <asp:BoundField DataField="CategoryID" HeaderText="ID" ReadOnly="true">
            <HeaderStyle HorizontalAlign="Left" />
            <ItemStyle Width="100px" />
        </asp:BoundField>
        <asp:BoundField DataField="ShortName" HeaderText="Short Name">
            <HeaderStyle HorizontalAlign="Left" />
            <ItemStyle Width="150px" />
        </asp:BoundField>
        <asp:BoundField DataField="LongName" HeaderText="Long Name">
            <HeaderStyle HorizontalAlign="Left" />
            <ItemStyle Width="200px" />
        </asp:BoundField>
        <asp:CommandField ButtonType="Button" ShowEditButton="True"
            CausesValidation="False" />
        <asp:CommandField ButtonType="Button" ShowDeleteButton="True"
            CausesValidation="False" />
    </Columns>
    <HeaderStyle BackColor="Silver" Font-Bold="True" ForeColor="White" />
    <RowStyle BackColor="White" ForeColor="Black" />
    <AlternatingRowStyle BackColor="WhiteSmoke" ForeColor="Black" />
    <EditRowStyle BackColor="Blue" ForeColor="White" />
</asp:GridView>
```

Basic attributes of the *GridView* control

Attribute	Description
ID	The *ID* of the control
Runat	Must specify *"server"*
DataSourceID	The *ID* of the data source to bind to
DataKeyNames	The names of the primary key fields separated by columns
AutoGenerateColumns	Specifies whether the control's columns should be automatically generated
SelectedIndex	Specifies the row to be initially selected

Description

- The *DataView* control display data from a data source in a row and column format. The data is rendered as an HTML table.
- To create a *GridView* control, drag the *GridView* icon from the *Data* group of the Toolbox.
- To bind a *GridView* control to a data source, use the smart tag menu's *Choose Data Source* command.

Common used fields properties when you use a Field dialog box

Property	Description
DataField	For bound field, the name of the column in the underlying data source that the field should be bound to
DataFormatString	A format string used to format the data. For example, use *{0:c}* to format a decimal value as currency.
HtmlEncode	Determines if values are HTML-encoded before they're displayed in a bound field. Set this property to *False* if you use the *DetailFormatString* property.
ItemStyle.Width	The width of the field
ReadOnly	True if the field is used for display only
NullDisplayText	The text that's displayed if the data field is *null*
ConvertEmptyStringToNull	If *True* (the default), empty strings are treated as nulls when data is updated in the database. Set this property to *False* if the underlying database field doesn't allow nulls.
HeaderText	The text that's displayed in the header row for the field
ShowHeader	True if the header should be displayed for this field

Description

- By default, the *GridView* control displays one column for each column in the data source.
- To define the fields that you want to display in the *GridView* control, display the Fields dialog box by selecting the *Edit Columns* command in the control's smart tag menu.
- You can also add columns by choosing the *Add New Column* command from the smart tag menu.

Elements used to create and format fields

Column field elements

Element	Description
Columns	The columns that are displayed by a *GridView* control
asp:BoundField	A field bound to a data source column
asp:ButtonField	A field that displays a button
asp:CheckBoxField	A field that displays a check box
asp:CommandField	A field that contains *Select, Edit, Delete, Update*, or *Cancel* buttons.
asp:HyperLinkField	A field that displays a hyperlink

asp:ImageField	A field that displays an image
asp:TemplateField	Lets you create a column with custom content

Style elements

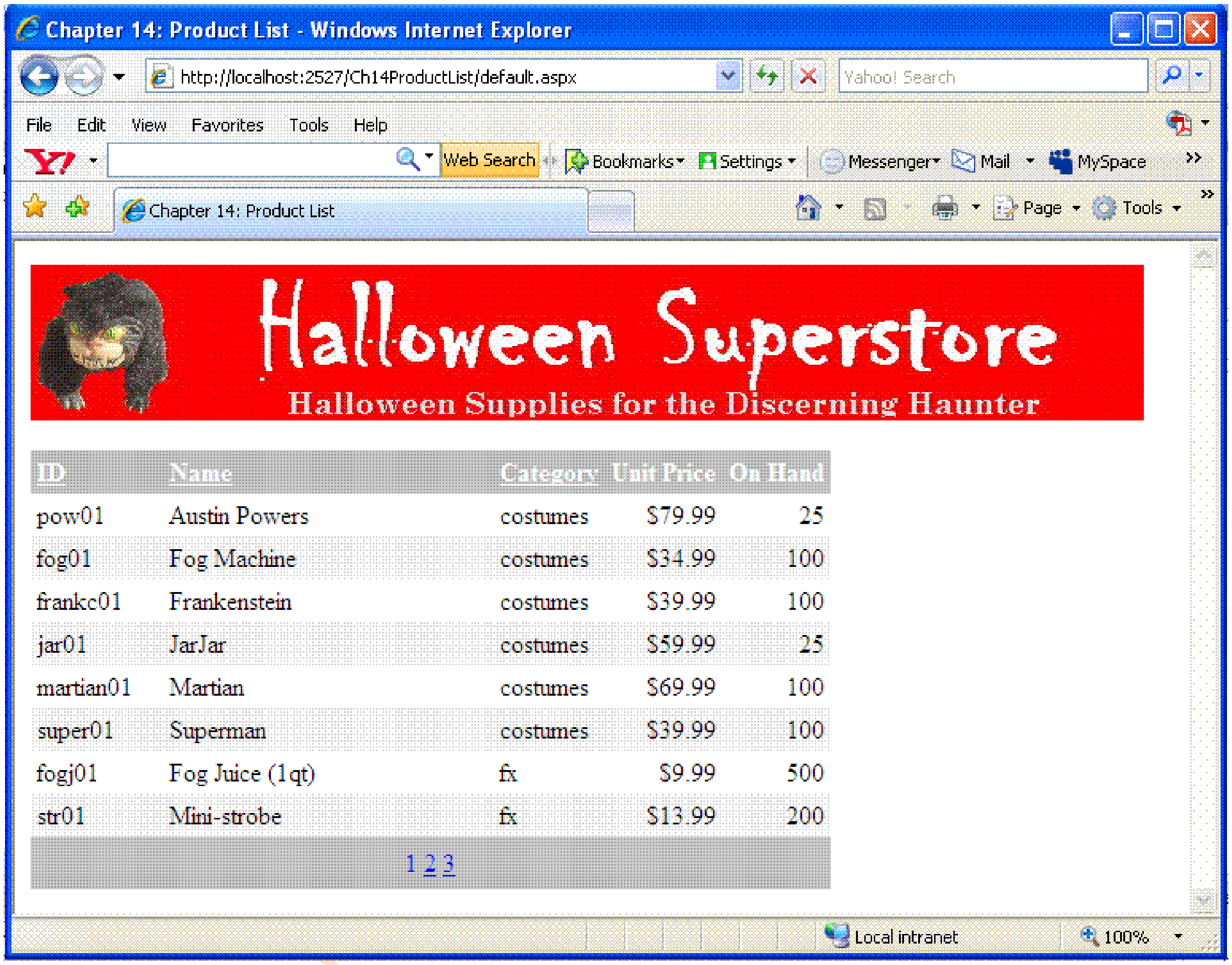
Element	Description
RowStyle	The style used for data rows
AlternatingRowStyle	The style used for alternating data row
SelectedRowStyle	The style used when the row is selected
EditRowStyle	The style used when the row is being edited
EmptyDataRowStyle	The style used when the data source is empty
ItemStyle	The style used for an individual field
HeaderStyle	The style used to format the header row
FooterStyle	The style used to format the footer row
PagerStyle	The style used to format the *GridView's* page row

The aspx code for a control that uses field and style elements

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
                DataKeyNames="CategoryID" DataSourceID="SqlDataSource1">
        <Columns>
                <asp:BoundField DataField="CategoryID" HeaderText="ID" ReadOnly="True"
                        ItemStyle Width="100px" />
                </asp:BoundField>
                .
                .
        </Columns>
        <HeaderStyle BackColor="LightGray" ForeColor="White" Font-Bold="True" />
        <RowStyle BackColor="White" ForeColor="Black" />
        <SelectedRowStyle BackColor="Gray" ForeColor="White" Font-Bold="True" />
        <FooterStyle BackColor="LightGray" ForeColor="Blue" />
        <PagerStyle BackColor="LightGray" ForeColor="Blue" HorizontalAlign="Center" />
</asp:GridView>
```

Enable sorting

A GridView control with sorting and paging enabled


Chapter 14: Product List - Windows Internet Explorer
http://localhost:2527/Ch14ProductList/default.aspx
Halloween Superstore
Halloween Supplies for the Discerning Haunter

ID	Name	Category	Unit Price	On Hand
pow01	Austin Powers	costumes	$79.99	25
fog01	Fog Machine	costumes	$34.99	100
frankc01	Frankenstein	costumes	$39.99	100
jar01	JarJar	costumes	$59.99	25
martian01	Martian	costumes	$69.99	100
super01	Superman	costumes	$39.99	100
fogj01	Fog Juice (1qt)	fx	$9.99	500
str01	Mini-strobe	fx	$13.99	200

1 2 3

The aspx code for the *GridView* control shown above

```
<asp:GridView ID="GridView1" runat="server"
        AllowSorting="True" AllowPaging="True" PageSize="8"
        DataKeyNames="ProductID" DataSourceID="SqlDataSource1"
        AutoGenerateColumns="False"  CellPadding="4" GridLines="None" ForeColor="Black">
    <Columns>
        <asp:BoundField DataField="ProductID" HeaderText="ID"
                ReadOnly="True" SortExpression="ProductID">
            <HeaderStyle HorizontalAlign="Left" />
            <ItemStyle Width="75px" />
        </asp:BoundField>
        <asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name">
            <HeaderStyle HorizontalAlign="Left" />
            <ItemStyle Width="200px" />
        </asp:BoundField>
        <asp:BoundField DataField="CategoryID" HeaderText="Category"
                SortExpression="CategoryID, Name" />
        <asp:BoundField DataField="UnitPrice" DataFormatString="{0:c}"
                HeaderText="Unit Price" HtmlEncode="False">
            <ItemStyle HorizontalAlign="Right" />
        </asp:BoundField>
        <asp:BoundField DataField="OnHand" HeaderText="On Hand">
            <ItemStyle HorizontalAlign="Right" />
        </asp:BoundField>
    </Columns>
    <HeaderStyle BackColor="Silver" Font-Bold="True"  ForeColor="White" />
    <RowStyle BackColor="White" ForeColor="Black" />
    <AlternatingRowStyle BackColor="WhiteSmoke" ForeColor="Black" />
    <FooterStyle BackColor="Silver" Font-Bold="True" ForeColor="White" />
    <PagerStyle BackColor="Silver" ForeColor="Blue"   HorizontalAlign="Center" />
    <PagerSettings Mode="NumericFirstLast" />
</asp:GridView>
```

Description

- To enable sorting, set the *AllowSorting* attribute to *True*. Then, add a *SortExpression* attribute to each column you want to allow sorting for.
- For sorting to work, the *DataSourceMode* attribute of the data source must be set to *DataSet* mode (the default)
- To enable paging, set the *AllowPaging* attribute to *True*. Then, add a *PagerStyle* element to define the appearance of the pager controls. You can also add a *PagerSettings* element to customize the way paging works (described below).
- For paging to work, the *DataSourceMode* attribute of the data source must be set to *DataSet* mode (the default)

Attributes of the GridView control that affect paging

Attribute	Description
AllowPaging	Set to *True* to enable paging
PageSize	Specifies the number of rows to display on each page. The default is 10.

Example

A PagerSettings element

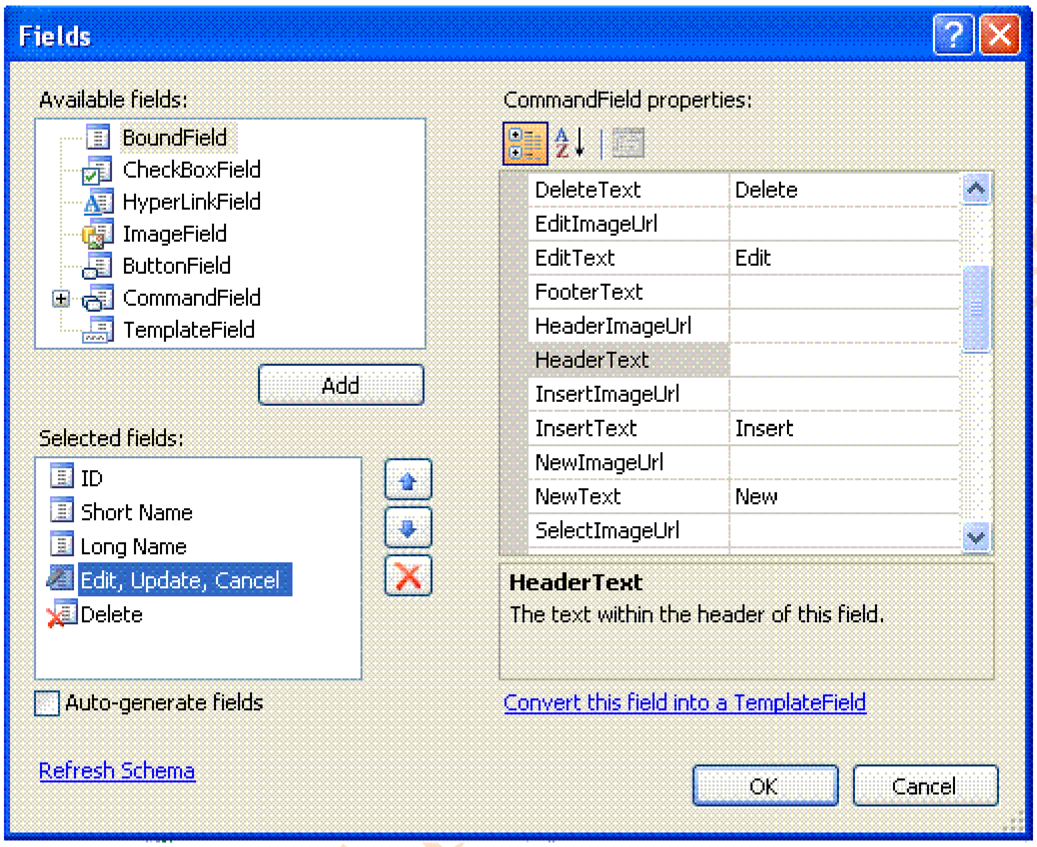
```
<PagerSettings Mode="NextPreviousFirstLast"
                    NextPageText="Next" PreviousPageText="Prev"
                    FirstPageText="First" LastPageText="Last" />
```

Update *GridView* data

One of the impressive features of the *GridView* control is its ability to update data in the underlying data source with a little additional code. Before you can set that up, though, you must configure the data source with *Update, Delete*, and *Insert* statements.

Working with command fields

The Fields dialog box for working with a command field



DeleteText	Delete
EditImageUrl	
EditText	Edit
FooterText	
HeaderImageUrl	
HeaderText	
InsertImageUrl	
InsertText	Insert
NewImageUrl	
NewText	New
SelectImageUrl	

HeaderText
The text within the header of this field.

Typical code to define command fields

```
<asp:CommandField ButtonType="Button" ShowEditButton="True"  CausesValidation="False" />
<asp:CommandField ButtonType="Button" ShowDeleteButton="True"  CausesValidation="False" />
```

Attributes of the *CommandField* element

Attributes	Description
ButtonType	Specifies the type of button displayed in the command field. Valid options are *Button, Link*, or *Image*.
CausesValidation	Specifies whether validation should be performed if the user clicks the button.
ValidationGroup	Specifies the name of the group to be validated if *CausesValidation* is *True*

Attributes that show buttons and set the text or images they display

Button	Show	Text	Image
Cancel	*ShowCancelButton*	*CancelText*	*CancelImage*
Delete	*ShowDeleteButton*	*DeleteText*	*DeleteImage*
Edit	*ShowEditButton*	*EditText*	*EditImage*
Select	*ShowSelectButton*	*SelectText*	*SelectImage*
Update	*n/a*	*UpdateText*	*UpdateImage*

Events raised by the *GridView* control

Although the *GridView* control provides many features automatically, you still must write some code to handle such things as a data validation, database exceptions, and concurrency errors.

The most common reason to handle the ***before-action events*** is to provide data validation. For example, when the user clicks the *Update* button, you can handle the *RowUpdating* event to make sure the user has entered correct data. If not, you can set the ***e*** argument's *Cancel* property to *True* to cancel the update.

In contrast, the ***after-action events*** give you opportunity to make sure the database operation completed successfully. In most applications, you should test for two conditions. First, you should check for any database exceptions by checking the *Exception* property of the ***e*** argument. If this property refers to a valid object, an exception has occurred and you can notify the user with an appropriate error message.

Events raised by the GridView control

Event	Raised when …
RowCancelingEdit	The Cancel button of a row in edit mode is clicked
RowDataBound	Data binding completes for a row
RowDeleted	A row has been deleted
RowDeliting	A row is about to be deleted
RowEditing	A row is about to be edited
RowUpdated	A row has been updated
RowUpdating	A row is about to be updated
SelectedIndexChanged	A row has been selected
SelectedIndexChangingcx	A row is about to be selected

An event handler for the RowUpdated event

```
protected void GridView1_RowUpdated(object sender, GridViewUpdatedEventArgs e)
{
  if (e.Exception != null)
  {
    lblError.Text = "An exception occurred. " +
      "Please correct any invalid data and try again.<br /><br />" +
      "Message: " + e.Exception.Message;
    e.ExceptionHandled = true;
    e.KeepInEditMode = true;
  }
  else if (e.AffectedRows == 0)
    lblError.Text = "No rows were updated. " +
      "Another user may have updated that category.<br />" +
      "Please try again.";
}
```

Description

- The *GridView* control raises various events that can be handled when data is updated
- The *RowUpdating* and *RowDeleting* events are often used for data validation. You can cancel the update or delete operation by setting the ***e*** argument's *Cancel* property to *True*.
- You can handle the *RowUpdate* and *RowDeleted* events to ensure that the row was successfully updated or deleted.
- To determine if a SQL exception has occurred, check the *Exception* property of the e argument. If an exception has occurred, the most likely cause is a *null* value for a column that doesn't accept nulls. To suppress the exception, you can set the *ExceptionHandled* property to *True*. And to keep the control in edit mode, you can set the *KeepEditMode* property to *True*.
- To determine how many rows were updated or deleted, check the *AffectRows* property of the e argument. If this property is *zero* and an exception has not been thrown, the most likely cause is a concurrency error.

Inserting a row in a GridView control

To provide for insertions, you must first create a set of input controls such as text boxes in which the user can enter data for the row to be inserted. Next you must provide a button that user can click to start the insertion. Then, in the event handler for this button, you can set the insert parameter values to the values entered by the user and call the data source's Insert method to add the new row.

Method and properties of the SqlDataSource class for inserting rows

Method	**Description**
Insert	Executes the *Insert* command defined for the data source
Property	**Description**
InsertCommand	The *Insert* command to be executed
InsertParameters("name")	The parameter with the specified name

Property of the Parameter class for inserting rows

Property	Description
DefaultValue	The default value of a parameter. This value is used if no other value is assigned to the parameter.

Code that uses a SqlDataSource control to insert a row

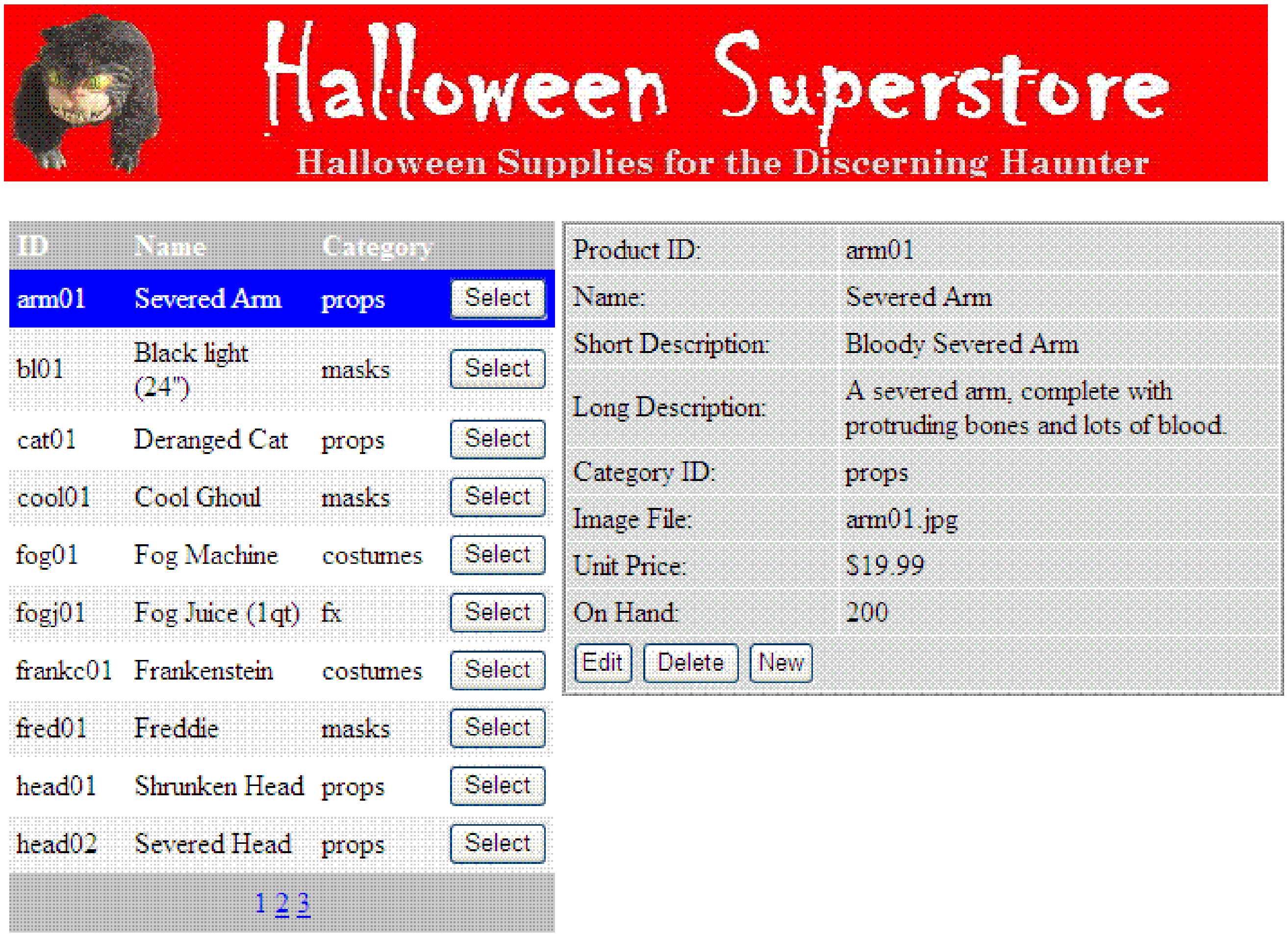
```
protected void btnAdd_Click(object sender, EventArgs e)
{
   SqlDataSource1.InsertParameters["CategoryID"].DefaultValue  = txtID.Text;
   SqlDataSource1.InsertParameters["ShortName"].DefaultValue  = txtShortName.Text;
   SqlDataSource1.InsertParameters["LongName"].DefaultValue  = txtLongName.Text;

   try
   {
      SqlDataSource1.Insert();
      txtID.Text = "";
      txtShortName.Text = "";
      txtLongName.Text = "";
   }
   catch (Exception ex)
   {
      lblError.Text = "An exception occurred. " +
         "The category was not added. Please try again.<br /><br />" +
         "Message: " + ex.Message;
   }
}
```

***DetailsView* control**

The *DetailsView* control is designed to display the data for a single item of a data source. To use this control effectively, you must provide some way for the user to select which data item to display. The most common way to do that is to use the *DetailsView* control in combination with another control such as a *GridView* control or a drop-down list.

A *DetailsView* control that displays data for a selected product



ID	Name	Category	
arm01	Severed Arm	props	Select
bl01	Black light (24")	masks	Select
cat01	Deranged Cat	props	Select
cool01	Cool Ghoul	masks	Select
fog01	Fog Machine	costumes	Select
fogj01	Fog Juice (1qt)	fx	Select
frankc01	Frankenstein	costumes	Select
fred01	Freddie	masks	Select
head01	Shrunken Head	props	Select
head02	Severed Head	props	Select
1 2 3			

Product ID:	arm01
Name:	Severed Arm
Short Description:	Bloody Severed Arm
Long Description:	A severed arm, complete with protruding bones and lots of blood.
Category ID:	props
Image File:	arm01.jpg
Unit Price:	$19.99
On Hand:	200
Edit Delete New	

The aspx code for the *DetailsView* control shown above

```
<asp:DetailsView ID="DetailsView1" runat="server" DataSourceID="SqlDataSource2"
        DataKeyNames="ProductID"  Height="50px" Width="400px" AutoGenerateRows="False"
         BackColor="White" BorderColor="White" BorderStyle="Ridge"
         BorderWidth="2px" CellPadding="3" CellSpacing="1"
         GridLines="None">
    <RowStyle BackColor="#DEDFDE" ForeColor="Black" />
    <Fields>
        <asp:BoundField DataField="ProductID" HeaderText="Product ID:"
                ReadOnly="True" SortExpression="ProductID">
            <HeaderStyle HorizontalAlign="Left" Width="150px" />
            <ItemStyle Width="250px" />
        </asp:BoundField>
        <asp:BoundField DataField="Name" HeaderText="Name:">
            <HeaderStyle HorizontalAlign="Left" Width="150px" />
            <ItemStyle Width="250px" />
        </asp:BoundField>
        <asp:BoundField DataField="ShortDescription" HeaderText="Short Description:">
            <HeaderStyle HorizontalAlign="Left" Width="150px" />
            <ItemStyle Width="250px" />
        </asp:BoundField>
        <asp:BoundField DataField="LongDescription" HeaderText="Long Description:">
            <HeaderStyle HorizontalAlign="Left" Width="150px" />
            <ItemStyle Width="250px" />
        </asp:BoundField>
        <asp:BoundField DataField="CategoryID" HeaderText="Category ID:">
            <HeaderStyle HorizontalAlign="Left" Width="150px" />
            <ItemStyle Width="250px" />
        </asp:BoundField>
        <asp:BoundField DataField="ImageFile" HeaderText="Image File:"
                SortExpression="ImageFile">
            <HeaderStyle HorizontalAlign="Left" Width="150px" />
            <ItemStyle Width="250px" />
        </asp:BoundField>
        <asp:BoundField DataField="UnitPrice" HeaderText="Unit Price:" DataFormatString="{0:c}"
                HtmlEncode="False">
            <HeaderStyle HorizontalAlign="Left" Width="150px" />
            <ItemStyle Width="250px" />
        </asp:BoundField>
        <asp:BoundField DataField="OnHand" HeaderText="On Hand:">
            <HeaderStyle HorizontalAlign="Left" Width="150px" />
            <ItemStyle Width="250px" />
        </asp:BoundField>
        <asp:CommandField ButtonType="Button"  ShowDeleteButton="True"
             ShowEditButton="True"  ShowInsertButton="True" />
    </Fields>
    <HeaderStyle BackColor="Silver" Font-Bold="True" ForeColor="Black" />
    <EditRowStyle BackColor="Blue" Font-Bold="True"  ForeColor="White" />
</asp:DetailsView>
```

Three modes of the *DetailsView* control

Mode	Description
ReadOnly	Used to display an item from the data source
Edit	Used to edit an item in the data source
Insert	Used to insert a new item into a data source

Attributes and child elements for the *DetailsView* control

DetailsView control attributes

Attribute	Description
ID	The *ID* of this control
Runat	Must specify *"server"*
DataSourceID	The *ID* of the data source to bind the *DetailsView* control to
DataKeyNames	A list of field names that the primary key for the data source
AutoGenerateRows	If *True*, a row is automatically generated for each column in the data source. If *False*, you must define the rows in the *Fields* element.
DefaultMode	Sets the initial mode of the *DetailsView* control. Valid options are *Edit, Insert*, or *ReadOnly*.
AllowPaging	Set to True to allow paging

DeatailsView child element

Element	Description
Fields	The fields that are displayed by a *DetailsView* controls
RowStyle	The style used for data rows in *ReadOnly* mode
AlternatingRowStyle	The style used for alternate rows
EditRowStyle	The style used for data rows in *Edit* mode
InsertRowStyle	The style used for data rows in *Insert* mode
CommandRowStyle	The style used for command rows
EmptyDataRowStyle	The style used for data rows when the data source is empty
EmptyDataTemplate	The template used when the data source is empty
HeaderStyle	The style used for the header row
HeaderTemplate	The template used for the header row
FooterStyle	The style used for the footer row
FooterTemplate	The template used for the footer row
PagerSettings	The settings used to control the page row
PagerStyle	The style used for the pager row
PagerTemplate	The template used for the pager row

Fields child elements

Element	Description
BoundField	A field bound to a data source column
ButtonField	A field that displays a button
CheckBoxField	A field that displays a check box
CommandField	A field that contains command buttons
HyperlinkField	A field that displays a hyperlink
ImageField	A field that displays an image
TemplateField	A field with custom content

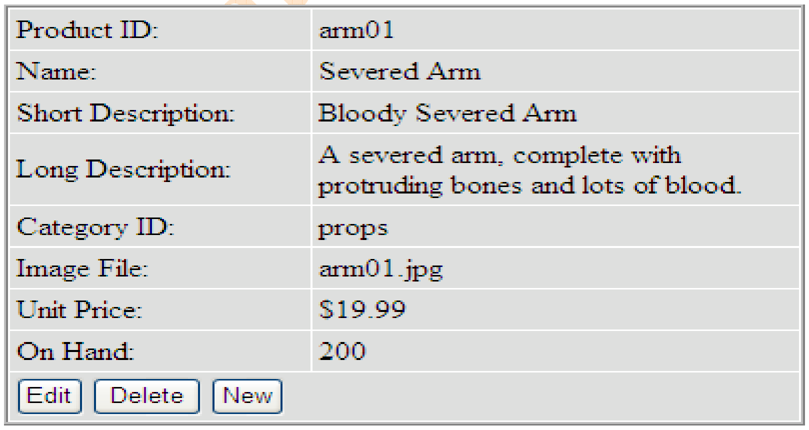
Enable paging

The aspx code for the DetailsView control

```
<asp:DetailsView ID="DetailsView1" runat="server" AllowPaging="true"
  DataSourceID="SqlDataSource2"  DataKeyNames="ProductID"
  Height="50px" Width="400px" AutoGenerateRows="False"
  BackColor="White" BorderColor="White" BorderStyle="Ridge"
  BorderWidth="2px" CellPadding="3" CellSpacing="1"
  GridLines="None">
  <RowStyle BackColor="#DEDFDE" ForeColor="Black" />
  <PagerSettings Mode="NextPreviousFirstLast" />
  <Fields>
    <asp:BoundField DataField="ProductID" HeaderText="Product ID:"
      ReadOnly="True" SortExpression="ProductID">
      <HeaderStyle HorizontalAlign="Left" Width="150px" />
      <ItemStyle Width="250px" />
    </asp:BoundField>
    <asp:BoundField DataField="Name" HeaderText="Name:">
      <HeaderStyle HorizontalAlign="Left" Width="150px" />
      <ItemStyle Width="250px" />
    </asp:BoundField>
    <asp:BoundField DataField="ShortDescription"
     HeaderText="Short Description:">
      <HeaderStyle HorizontalAlign="Left" Width="150px" />
      <ItemStyle Width="250px" />
    </asp:BoundField>
    <asp:BoundField DataField="LongDescription"
      HeaderText="Long Description:">
      <HeaderStyle HorizontalAlign="Left" Width="150px" />
      <ItemStyle Width="250px" />
    </asp:BoundField>
    <asp:BoundField DataField="CategoryID"
      HeaderText="Category ID:">
      <HeaderStyle HorizontalAlign="Left" Width="150px" />
      <ItemStyle Width="250px" />
    </asp:BoundField>
    <asp:BoundField DataField="ImageFile"
      HeaderText="Image File:" SortExpression="ImageFile">
      <HeaderStyle HorizontalAlign="Left" Width="150px" />
      <ItemStyle Width="250px" />
    </asp:BoundField>
    <asp:BoundField DataField="UnitPrice"
      HeaderText="Unit Price:" DataFormatString="{0:c}"
      HtmlEncode="False">
      <HeaderStyle HorizontalAlign="Left" Width="150px" />
      <ItemStyle Width="250px" />
    </asp:BoundField>
    <asp:BoundField DataField="OnHand" HeaderText="On Hand:">
      <HeaderStyle HorizontalAlign="Left" Width="150px" />
      <ItemStyle Width="250px" />
    </asp:BoundField>
    <asp:CommandField ButtonType="Button"
      ShowDeleteButton="True"
      ShowEditButton="True"
      ShowInsertButton="True" />
  </Fields>
  <HeaderStyle BackColor="Silver" Font-Bold="True"
    ForeColor="Black" />
  <EditRowStyle BackColor="Blue" Font-Bold="True"
    ForeColor="White" />
</asp:DetailsView>
```

Updating of the DetailsView data

A DetailsView control with automatically generated command buttons



Product ID:	arm01
Name:	Severed Arm
Short Description:	Bloody Severed Arm
Long Description:	A severed arm, complete with protruding bones and lots of blood.
Category ID:	props
Image File:	arm01.jpg
Unit Price:	$19.99
On Hand:	200

Command buttons

Button	Description
Edit	Places the *DetailsView* control in *Edit* mode
Delete	Deletes the current item leaves and the *DetailsView* control in *ReadOnly* mode
New	Places the *DetailsView* control in *Insert* mode
Update	Displays only in *Edit* mode. Updates the data source, and then returns to *ReadOnly* mode.
Insert	Displays only in *Insert* mode. Inserts the data, and then returns to *ReadOnly* mode.
Cancel	Displays in *Edit* or *Insert* mode. Cancels the operation and returns to *ReadOnly* mode.

Attributes that generate command buttons

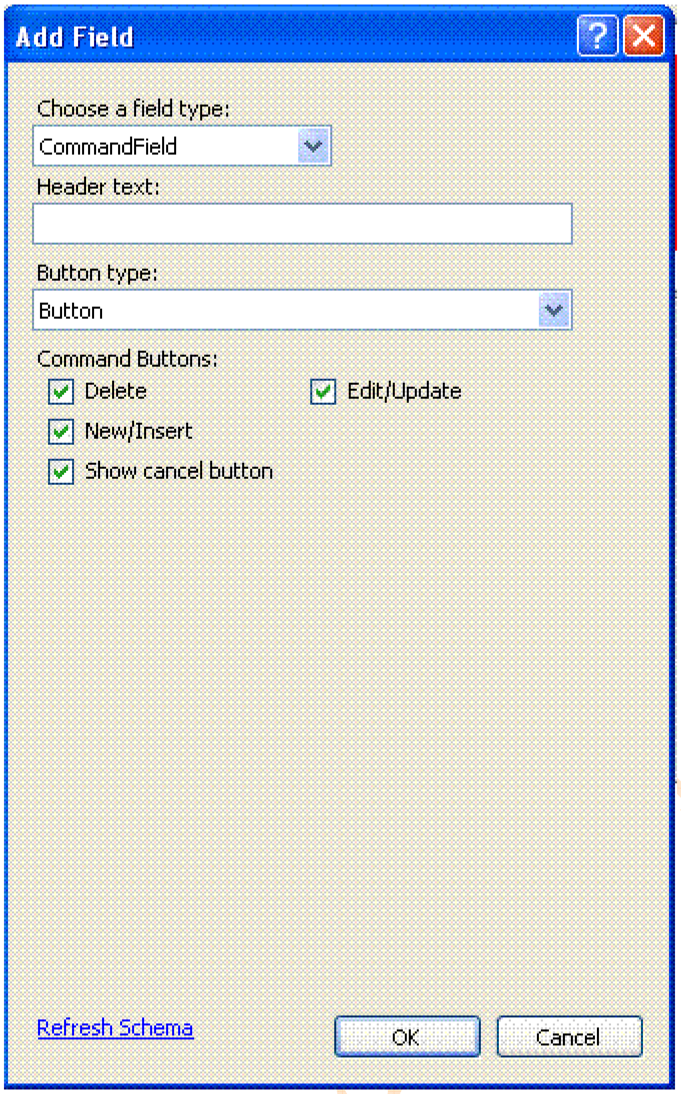
Attribute	Description
AutoGenerateDeleteButton	Generates a *Delete* button
AutoGenerateEditButton	Generates an *Edit* button
AutoGenerateInsertButton	Generates a *New* button

A DetailsView element that automatically generates command buttons

```
<asp:DetailsView ID="DetailsView1" runat="server"
    DataSourceID=”SqlDataSource2” DataKeyNames=”ProductID”
    AutoGenerateRows=”False”
    AutoGenerateDeleteButton=”True”
    AutoGenerateEditButton=”True”
    AutoGenerateInsertButton=”True”>
```

Adding command buttons

The Add Field dialog box for adding a command field



Code generated by the above dialog box

```
<asp:CommandField ButtonType="Button"
   ShowDeleteButton="True"
   ShowEditButton="True"
   ShowInsertButton="True" />
```

Description

- You can add command to a *DetailsView* control to let the user update, insert, and delete data.
- The command buttons for a *DetailsView* control are similar to the command buttons for a *GridView* control. However, the *DetailsView* control doesn't provide a *Select* button, and it does provide *New* and *Insert* buttons.
- To display the *Add Field* dialog box, choose *Add New Field* from the smart tag menu of the *DetailsView* control.

Handling *DetailsView* events

Events raised by the *DetailsView* control

Event	Description
ItemCommand	Raised when a button is clicked
ItemCreated	Raised when an item is created
DataBound	Raised when data binding competes for an item
ItemDeleted	Raised when an item has been deleted
ItemDeliting	Raised when an item is about to be deleted
ItemInserted	Raised when an item has been inserted
ItemInserting	Raised when is about to be deleted
ItemUpdated	Raised when an item has been updated
ItemUpdating	Raised when item is about to be deleted
PageIndexChanged	Raised when the index of the displayed item has changed
PageIndexChanging	Raised when the index of the displayed item is about to change

The event handler for the ItemUpdated event

```
protected void FormView1_ItemUpdated(object sender, FormViewUpdatedEventArgs e)
{
   if (e.Exception != null)
   {
      lblError.Text = "An exception has occurred. " +
         "Please check all entries and try again.<br /><br />" +
         "Message: " + e.Exception.Message;
      e.ExceptionHandled = true;
      e.KeepInEditMode = true;
   }
   else if (e.AffectedRows == 0)
   {
      lblError.Text = "Another user may have updated that product. "
         + "Please try again.";
   }
   else
   {
      GridView1.DataBind();
   }
}
```

Description

- Like the *GridView* control, the *DetailsView* control raises events that can be handled when data is updated. At the minimum, you should use these events to test for database exceptions and concurrency errors.
- To determine if a SQL exception has occurred, test the *Exception* property of the ***e*** argument. If an exception has occurred, you can set the *ExceptionHandled* property to *True* to suppress the exception. You can also set the *KeepInEditMode* property to true to keep the *DetailsView* control in *Edit* mode. And you can also set the *KeepInInsertMode* property to true to keep the control in *Insert* mode.
- If the *AffectRows* property of the e argument is zero and an exception has not been thrown, a concurrency error has probably occurred.

Fixing the Optimistic concurrency bug

Optimistic concurrency works by using a Where clause that compares each column in the database row with the values saved when the row was originally retrieved. If that row can't be found, it means that another user has updated the row and changed one of the columns. Then, the row isn't updated or deleted.

Unfortunately, there's a bug in the way ASP.NET 2.0 generates the Where clauses for columns that allows nulls. This bug, along with a workaround for it, below.

A generated Delete statement that handles concurrency errors

```
DELETE FROM [Products]
        WHERE [ProductID] = @original_ProductID
                AND [Name] = @original_Name
                AND [ShortDescription] = @original_ShortDescription
                AND [LongDescription] = @original_LongDescription
                AND [CategoryID] = @original_CategoryID
                AND [ImageFile] = @original_ImageFile
                AND [UnitPrice] = @original_UnitPrice
                AND [OnHand] = @original_OnHand
```

How to modify the Delete statement for a column that allows nulls

```
DELETE FROM [Products]
        WHERE [ProductID] = @original_ProductID
                AND [Name] = @original_Name
                AND [ShortDescription] = @original_ShortDescription
                AND [LongDescription] = @original_LongDescription
                AND [CategoryID] = @original_CategoryID
                AND ( [ImageFile] = @original_ImageFile
                 OR ImageFile IS NULL AND @original_ImageFile IS NULL )
                AND [UnitPrice] = @original_UnitPrice
                AND [OnHand] = @original_OnHand
```

In short, the problem is that when a database column allows nulls, the comparisons generated for the *WHERE* clause don't work. That's because SQL defines the result of an equal comparison between a null and null as *False*. (Since null represents an unknown value, no value-even another null-can be considered equal to a *null*).

The Halloween database illustrates this problem because it allows nulls for the *ImageFile* column in the *Product* table. But look at the *WHERE* clause that's generated for the first *DELETE* statement here:

[ImageFile] = @original_ImageFile

In this case, if the original value of the *ImageFile* column is *null*, this comparison will never test true, so the row will never be deleted.

The workaround to this bug is to modify the generated *DELETE* and *UPDATE* statements for any database table that allows nulls in any of its columns. For the *ImageFile* column, you can modify the *DetailsView* statement so it looks like this:

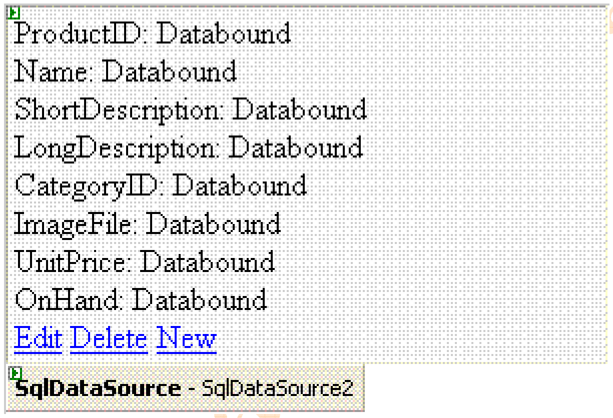
([ImageFile] = @original_ImageFile OR ImageFile IS NULL AND @original_ImageFile IS NULL)

Then, the comparison will test true if both the *ImageFile* column and the *@original_ImageFile* parameter are null.

***FormView* control**

Like the *DetailsView* control, the *FormView* control is designed to display data for a single item from a data source. But this control uses a different approach to displaying its data.

A *FormView* control after a data source has been assigned


ProductID: Databound
Name: Databound
ShortDescription: Databound
LongDescription: Databound
CategoryID: Databound
ImageFile: Databound
UnitPrice: Databound
OnHand: Databound
Edit Delete New
SqlDataSource - SqlDataSource2

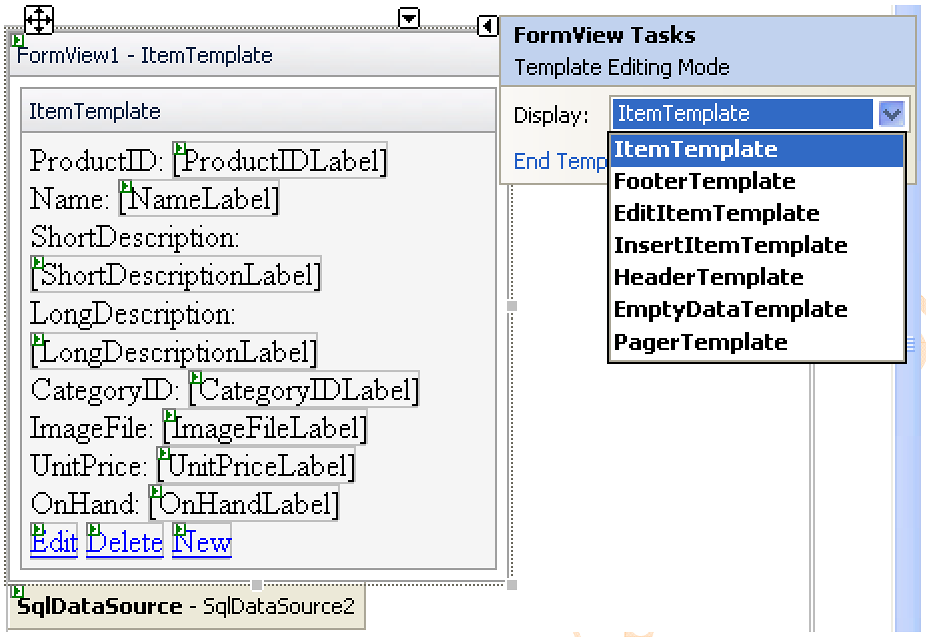
How the *FormView* control fifers from the *DetailsView* control

- The *DetailsView* control is easier to work with, but the *FormView* control provides more formatting and *Layout* options
- The *DetailsView* control renders each data source item as a table row, but the *FormView* control uses a template to render each item.
- The *DetailsView* control uses *BoundField* elements to define bound data fields, but the *FormView* control uses data binding expressions in its templates to display bound data.

Description

- A *FormView* control is similar to a *DetailsView* control, but uses templates that give you more control over how its data is displayed.
- To create a *FormView* control, you drag it from the *Data* group of the Toolbox onto the page, and you assign a data source to the control. Then, you edit the control's templates so the data is displayed the way you want.

The FormView control in template-editing mode


FormView1 - ItemTemplate

ItemTemplate

ProductID: [ProductIDLabel]
Name: [NameLabel]
ShortDescription:
[ShortDescriptionLabel]
LongDescription:
[LongDescriptionLabel]
CategoryID: [CategoryIDLabel]
ImageFile: [ImageFileLabel]
UnitPrice: [UnitPriceLabel]
OnHand: [OnHandLabel]
Edit Delete New

SqlDataSource - SqlDataSource2

FormView Tasks
Template Editing Mode
Display: ItemTemplate
End Temp

The Item template generated for a FormView control

```
<asp:FormView ID="FormView1" runat="server" BackColor="White" BorderColor="#999999"
 BorderStyle="None" BorderWidth="1px" CellPadding="3" DataKeyNames="ProductID"
 DataSourceID="SqlDataSource2" GridLines="Vertical" Width="300px"
 OnItemDeleted="FormView1_ItemDeleted" OnItemDeleting="FormView1_ItemDeleting"
 OnItemInserted="FormView1_ItemInserted" OnItemUpdated="FormView1_ItemUpdated">
 <FooterStyle BackColor="Silver" ForeColor="White" />
 <RowStyle BackColor="WhiteSmoke" ForeColor="Black" />
 <EditItemTemplate>
   ProductID:
   <asp:Label ID="ProductIDLabel1" runat="server" Text='<%# Eval("ProductID")
        %>'></asp:Label><br />
   Name:
   <asp:TextBox ID="NameTextBox" runat="server" Text='<%# Bind("Name") %>'>
   </asp:TextBox><br />
   ShortDescription:
   <asp:TextBox ID="ShortDescriptionTextBox" runat="server" Text='<%# Bind("ShortDescription")
        %>'>
```

```
    </asp:TextBox><br />
    LongDescription:
    <asp:TextBox ID="LongDescriptionTextBox" runat="server" Text='<%# Bind("LongDescription")
        %>'>
    </asp:TextBox><br />
    CategoryID:
    <asp:TextBox ID="CategoryIDTextBox" runat="server" Text='<%# Bind("CategoryID") %>'>
    </asp:TextBox><br />
    ImageFile:
    <asp:TextBox ID="ImageFileTextBox" runat="server" Text='<%# Bind("ImageFile") %>'>
    </asp:TextBox><br />
    UnitPrice:
    <asp:TextBox ID="UnitPriceTextBox" runat="server" Text='<%# Bind("UnitPrice") %>'>
    </asp:TextBox><br />
    OnHand:
    <asp:TextBox ID="OnHandTextBox" runat="server" Text='<%# Bind("OnHand") %>'>
    </asp:TextBox><br />
    <asp:LinkButton ID="UpdateButton" runat="server" CausesValidation="True"
CommandName="Update"
      Text="Update">
    </asp:LinkButton>
    <asp:LinkButton ID="UpdateCancelButton" runat="server" CausesValidation="False"
CommandName="Cancel"
      Text="Cancel">
    </asp:LinkButton>
  </EditItemTemplate>
  <InsertItemTemplate>
    ProductID:
    <asp:TextBox ID="ProductIDTextBox" runat="server" Text='<%# Bind("ProductID") %>'>
    </asp:TextBox><br />
    Name:
    <asp:TextBox ID="NameTextBox" runat="server" Text='<%# Bind("Name") %>'>
    </asp:TextBox><br />
    ShortDescription:
    <asp:TextBox ID="ShortDescriptionTextBox" runat="server" Text='<%# Bind("ShortDescription")
        %>'>
    </asp:TextBox><br />
    LongDescription:
    <asp:TextBox ID="LongDescriptionTextBox" runat="server" Text='<%# Bind("LongDescription")
        %>'>
    </asp:TextBox><br />
    CategoryID:
    <asp:TextBox ID="CategoryIDTextBox" runat="server" Text='<%# Bind("CategoryID") %>'>
    </asp:TextBox><br />
    ImageFile:
    <asp:TextBox ID="ImageFileTextBox" runat="server" Text='<%# Bind("ImageFile") %>'>
    </asp:TextBox><br />
    UnitPrice:
    <asp:TextBox ID="UnitPriceTextBox" runat="server" Text='<%# Bind("UnitPrice") %>'>
    </asp:TextBox><br />
    OnHand:
    <asp:TextBox ID="OnHandTextBox" runat="server" Text='<%# Bind("OnHand") %>'>
    </asp:TextBox><br />
    <asp:LinkButton ID="InsertButton" runat="server" CausesValidation="True"
CommandName="Insert"
      Text="Insert">
    </asp:LinkButton>
    <asp:LinkButton ID="InsertCancelButton" runat="server" CausesValidation="False"
CommandName="Cancel"
      Text="Cancel">
    </asp:LinkButton>
  </InsertItemTemplate>
  <PagerStyle BackColor="Silver" ForeColor="White" HorizontalAlign="Center" />
  <HeaderStyle BackColor="Silver" Font-Bold="True" ForeColor="White" />
  <EditRowStyle BackColor="Silver" Font-Bold="True" ForeColor="White" />
  <ItemTemplate>
    ProductID:
    <asp:Label ID="ProductIDLabel" runat="server" Text='<%# Eval("ProductID")
       %>'></asp:Label><br />
    Name:
    <asp:Label ID="NameLabel" runat="server" Text='<%# Bind("Name") %>'></asp:Label><br />
    ShortDescription:
    <asp:Label ID="ShortDescriptionLabel" runat="server" Text='<%# Bind("ShortDescription")
       %>'>
    </asp:Label><br />
    LongDescription:
    <asp:Label ID="LongDescriptionLabel" runat="server" Text='<%# Bind("LongDescription") %>'>
    </asp:Label><br />
    CategoryID:
    <asp:Label ID="CategoryIDLabel" runat="server" Text='<%# Bind("CategoryID") %>'>
    </asp:Label><br />
    ImageFile:
    <asp:Label ID="ImageFileLabel" runat="server" Text='<%# Bind("ImageFile")
       %>'></asp:Label><br />
    UnitPrice:
    <asp:Label ID="UnitPriceLabel" runat="server" Text='<%# Bind("UnitPrice", "{0:c}")
       %>'></asp:Label><br />
    OnHand:
    <asp:Label ID="OnHandLabel" runat="server" Text='<%# Bind("OnHand") %>'></asp:Label><br
/>
    <asp:LinkButton ID="EditButton" runat="server" CausesValidation="False"
CommandName="Edit"
```

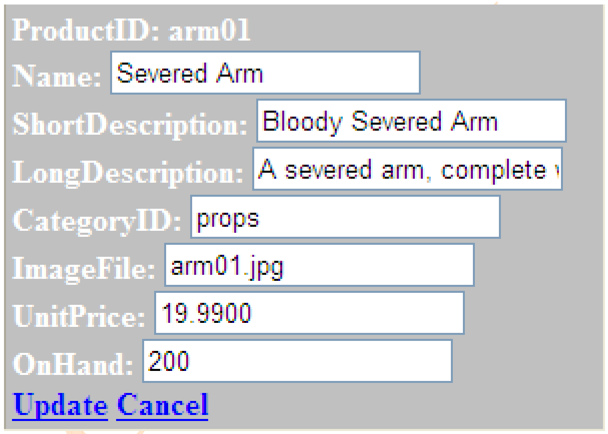
```
    Text="Edit">
  </asp:LinkButton>
  <asp:LinkButton ID="DeleteButton" runat="server" CausesValidation="False"
    CommandName="Delete"  Text="Delete">
  </asp:LinkButton>
  <asp:LinkButton ID="NewButton" runat="server" CausesValidation="False"
    CommandName="New"  Text="New">
  </asp:LinkButton>
 </ItemTemplate>
</asp:FormView>
```

Description

- When you bind a *FormView* control to a data source, templates are created with heading text, bound labels, and text boxes for each columns in the data source
- The Item template is rendered whenever the *FormView* control is bound in *ReadyOnly* mode
- The generated templates use the new *Eval* and *Bind* methods to create binding expressions for each of the columns in the data source.
- If the data source includes *Update, Delete*, and *Insert* statements, the generated Item template will include *Edit, Delete*, and *New* buttons.
- The Web Forms Designer also generates an *EditItem* template and an *InsertItem* template, even if the data source doesn't include an *Update* or *Insert* command.
- You can add a table to a generated template to control the layout of the data that's rendered for that template.

Working with the EditItem and InsertItem templates

A generated EditItem template as displayed in a browser window



ProductID: arm01
Name: Severed Arm
ShortDescription: Bloody Severed Arm
LongDescription: A severed arm, complete
CategoryID: props
ImageFile: arm01.jpg
UnitPrice: 19.9900
OnHand: 200
Update Cancel

The aspx code for the edit item template shown above

```
<EditItemTemplate>
 ProductID:
 <asp:Label ID="ProductIDLabel1" runat="server" Text='<%# Eval("ProductID") %>'>
 </asp:Label><br />
 Name:
 <asp:TextBox ID="NameTextBox" runat="server" Text='<%# Bind("Name") %>'>
 </asp:TextBox><br />
 ShortDescription:
 <asp:TextBox ID="ShortDescriptionTextBox" runat="server"
       Text='<%# Bind("ShortDescription") %>'>
 </asp:TextBox><br />
 LongDescription:
 <asp:TextBox ID="LongDescriptionTextBox" runat="server"
       Text='<%# Bind("LongDescription") %>'>
 </asp:TextBox><br />
 CategoryID:
 <asp:TextBox ID="CategoryIDTextBox" runat="server" Text='<%# Bind("CategoryID") %>'>
 </asp:TextBox><br />
 ImageFile:
 <asp:TextBox ID="ImageFileTextBox" runat="server" Text='<%# Bind("ImageFile") %>'>
 </asp:TextBox><br />
 UnitPrice:
 <asp:TextBox ID="UnitPriceTextBox" runat="server" Text='<%# Bind("UnitPrice") %>'>
 </asp:TextBox><br />
 OnHand:
 <asp:TextBox ID="OnHandTextBox" runat="server" Text='<%# Bind("OnHand") %>'>
 </asp:TextBox><br />
 <asp:LinkButton ID="UpdateButton" runat="server" CausesValidation="True"
       CommandName="Update"  Text="Update">
 </asp:LinkButton>
 <asp:LinkButton ID="UpdateCancelButton" runat="server" CausesValidation="False"
       CommandName="Cancel"  Text="Cancel">
 </asp:LinkButton>
</EditItemTemplate>
```

Description

- The *EditItem* template determines how the *FormView* control is rendered in *Edit* mode. It includes a text box for each bound column in the data source. The *Text* attribute for each text box uses a binding expression that binds the text box to its data source column.
- The *EditItem* template also includes *Update* and *Cancel* buttons.
- The *InsertItem* template is similar to the *EditItem* template. It determines how the *FormView* control is rendered in *Insert* mode.