1 **GridView**

CST272—ASP.NET

2 **The ASP:GridView Web Control (Page 1)**

- Binds to and displays data from a data source control in tabular view (rows and columns)
- To assign the data source:
  - Click the "smart tag" and select a data source (e.g. a SQLDataSource object) from Choose Data Source: drop-down list
    - This formats the columns to represent the fields of the selected query (the data source)

3 **The ASP:GridView Web Control (Page 2)**

- If the GridView control initially does not reformat to represent the template of the actual data from the SqlDataSource, click Refresh Schema command
  - The source code is updated automatically to include a <Columns> block and <asp:BoundField> tags which represent the data

4 **The ASP:GridView Web Control (Page 3)**

- When a data source is selected for the GridView control and the schema refreshed, the following of its properties are updated automatically:
  - AutoGenerateColumns—how are fields generated?:
    - True—from the data source at run-time
    - False—columns will be defined in <Columns> block
  - DataKeyNames—the DataField property of the BoundColumn(s) that uniquely identify each record (by default the *primary key*)
  - DataSourceID—the ID of its data source controls

6 **Customizing ASP:GridView Using AutoFormat**

- Formats the GridView by allowing user to select from a series of *predefined styles*
- Click the "smart tag" for the GridView and click the AutoFormat... command
- Select one of the predefined styles to *preview* it
- Click the <OK> button to implement the style
  - The appropriate properties are set in the source code

7 **Customizing the ASP:GridView Web Control**

- Levels of customization/formatting:
  - GridView-level—affect *all* data in the GridView
  - Field-level—to format individual fields (columns)
  - Row-level—to format specific groupings of rows (headers, footers, and details), e.g.
    - Larger font size for the header row
    - Alternating background colors of rows

8 **Customizing the ASP:GridView Web Control Using Properties Window (Page 1)**

- Appearance properties set in "Properties Window":

- BackColor—color of entire background
- BackImageUrl—path and filename of background image
- BorderColor—color of border around the control
- BorderStyle—including NotSet, None, Dotted, Dashed, Solid, etc.
- BorderWidth—measured in pixels
- CssClass—name of CSS (cascading style sheet) class in <head> applied for formatting

9 **Customizing the ASP:GridView Web Control Using Properties Window (Page 2)**

- Appearance properties set in "Properties Window":
  - EmplyDataText—text rendered if there are no records
  - Font—typeface and styles (drill-down for subproperties)
  - ForeColor—text color
  - GridLines—None, Vertical, Horizontal or Both
  - ShowFooter—Boolean value indicates whether footer row displays (False by default)
  - ShowHeader—Boolean value indicates whether header row displays (True by default)

10 **Customizing ASP:GridView Row Classes (Page 1)**

- These are drill-down properties set in the "Properties Window" (BackColor, ForeColor, Font, etc.) that affect specific groupings or classes of rows:
  - AlternatingRowStyle—every other row
  - EditRowStyle—temporarily set for a row that is being added, edited or deleted
  - EmptyDataRowStyle—if no records are in the data source and EmptyDataText property is set to a value
  - FooterStyle—style of footer row
  - HeaderStyle—style of header row

11 **Customizing ASP:GridView Row Classes (Page 2)**

- These are drill-down properties set in the "Properties Window" (BackColor, ForeColor, Font, etc.) that affect specific groupings or classes of rows:
  - PagerStyle—set for the "pager" row
  - RowStyle—the style for rows (the records, but not headers or footers)
  - SelectedRowStyle—temporarily set for the row that is selected

12 **Customizing ASP:GridView Fields (Page 1)**

- The "Fields" dialog window lists all selected fields for a GridView control
- Click the "smart tag" for the GridView and click the Edit Columns… command
- Then select the field(s) from the Selected Fields: list

13 **Customizing ASP:GridView Fields (Page 2)**

- Field customizable properties include:
  - HeaderText—text in the field's header row (default it the fieldname)

- DataFormatString—provides formatting for numeric type fields (including dates)
- HeaderStyle—drill-down property to format header (BackColor, ForeColor, Font, etc.) for specific fields
- ItemStyle—drill-down property to format specific fields within rows (BackColor, ForeColor, Font, etc.)

16 **The ASP:GridView Web Paging Property Settings (Page 1)**

- Paging features also are available in GridView
- The properties are:
  - AllowingPaging
    - True—paging is supported and the pager row (the paging controls) are displayed
    - False—paging is not supported (default)
  - PageIndex—specifies index of the page initially displayed (default is 0 for first page but it is possible to start at a subsequent page in the table)

17 **The ASP:GridView Web Paging Property Settings (Page 2)**

- The properties are:
  - PagerSettings—drill-down properties that are used to customize the pager row
    - Mode—options are NextPrevious, Numeric (displays page numbers—the default), NextPreviousFirstLast, and NumericFirstLast
    - The other properties determine the *text* and/or *image* for the Next, Previous, First and Last buttons
  - PageSize—the number of rows (records) to display per page (default is 10)

21 **The ASP:GridView Web Sorting Property Settings**

- To enable *sorting* in GridView, click the "smart tag" for the GridView and click the EnableSorting checkbox “on”
- Click a column header once to sort on that column from lowest to highest (click the column header again to sort in reverse order)
- To *disable* sorting for a specific field:
  - Click the "smart tag" for the GridView and click the EditColumns... command
  - Select the field and *clear* its SortExpression field

24 **The ASP:SqlDataSource Web Control for Inserting, Updating and Deleting**

- Click the smart tag for the SqlDataSource, select “Configure Data Source” and configure as usual
- Once the select statement is specified ...
  - Click the <Advanced> button which displays the “Advanced SQL Generation Options” dialog window
  - Click “on” the Generate INSERT, UPDATE, and DELETE statements checkbox
  - Adds InsertCommand, UpdateCommand, and DeleteCommand properties to <asp:SqlDataSource> control tag

27 **Data Maintenance in SQL (A Review)**

- Keeping data *current*
  - INSERT INTO
    - Adds a new rows (records) to a table
  - UPDATE/SET
    - Modifies contents (value) of one or more fields
  - DELETE [FROM]
    - Removes one or more records from a table

28 **Adding a Row to the Table**

- INSERT INTO *tableName*
  [(*columnNames* …)]
  VALUES(*valueList* …);

```
INSERT INTO Payables
    VALUES ('10004', '650', 1000, 3, .F.);
INSERT INTO Payables
   (PoNumber,  ProductNumber, Quantity, BackOrdered)
    VALUES ('10004', '701', 8, .F.);
```

- *VALUES* must match number of items and data types

29 **Updating Data in a Database**

- UPDATE *tableName*
  SET *columnName1* = *newValue* [,
  columnName2 = *newValue*, …]
  WHERE *columnName* = *criteria*;

```
UPDATE PurchaseOrder
    SET BillDate = '02/28/99'
    WHERE PoNumber = "10001";
UPDATE PurchaseOrder
    SET DiscountDate = '03/31/99';
```

- If there is no WHERE clause, all rows will be updated

30 **Deleting Rows from a Table**

- DELETE [FROM] *tableName*
  WHERE *columnName* = *criteria*;

```
DELETE FROM Payables
    WHERE PoNumber = '10004';
DELETE FROM PurchaseOrder
    WHERE PoNumber = '10004';
```

- Remember to delete all records that refer to PO number "10004" in both *related tables*

31 **SQL Parameters** **(Page 1)**

- Visual Studio uses parameters to represent *variable* data in a "parameterized" SQL statement
- It is an object that can accept different values based upon the logic of the application
  - The values are determined *dynamically* at run-time

32 **SQL Parameters (Page 2)**

- When "Generate INSERT, UPDATE, AND DELETE statements" is selected, there are three parameter blocks inserted for each <asp:SqlDataSource> tag:
  - <InsertParameters>—parameters used within the SQL Insert statement
  - <UpdateParameters>—parameters used within the SQL Update statement
  - <DeleteParameters>—parameters used within the SQL Delete statement

33 **SQL Parameters (Page 3)**

- The properties for the <asp:Parameter> tag include:
  - Name—used within the parameterized SQL statement
  - Type—*data type* which is a Visual C# type consistent with the type in the database
  - Format:

    <asp:Parameter Name="*nameProperty*" Type="*dataType*" />
  - Example:

    <asp:Parameter Name="ProductID" Type="String" />

34 **SQL Parameters (Page 4)**

- Parameter format in a parameterized SQL statement:

  @parameterName
  - Usually the same name as the field it is querying, but not required
- Example of a parameterized SQL statement:

```
<asp:SqlDataSource …
      DeleteCommand="DELETE FROM [Product] WHERE [ProductID] = @ProductID"
        … >
```

  - The expression @ProductID is the parameter

35 **Deleting Data with GridView (Page 1)**

- In GridView records can be *deleted* by adding a Delete command (a button or link) within a CommandField column for each record
- When the button/link is clicked:
  - The page is posted back
  - The data source's SQL DeleteCommand is executed
  - The GridView retrieves that updated table from the data source and redisplays the data
- To configure GridView for deleting, click the smart tag and click "on" the Enable Deleting checkbox

36 **Deleting Data with GridView (Page 2)**

- To format the Delete command, click GridView's smart tag and select the Edit

Columns… command
  – A CommandField column is added with its ShowDeleteButton property set to True)
- The properties include:
  – ButtonType—either Button, Image or Link (default)
  – DeleteImageUrl—an image if ButtonType is Image
  – DeleteText—text if ButtonType is Button or Link
  – Additionally Font and other stylistic properties may be specified

39 **Editing Data with GridView (Page 1)**

- In GridView records can be *edited* by adding an Edit command (button or link) within the CommandField column for each record
- When the button/link is clicked:
  – The page is posted back and all editable fields (not the primary key or other fields set to be read-only) are converted to TextBoxes for updating
  – During editing the Edit button is converted to two buttons, Update and Cancel

40 **Editing Data with GridView (Page 2)**

- When the Update button is clicked:
  – The page is posted back again
  – The data source's SQL UpdateCommand is executed
  – The GridView retrieves that updated table from the data source and redisplays the data
- If the Cancel button is clicked:
  – The page is posted back, but GridView is updated from the unmodified data source
- To configure GridView for editing, click smart tag and click "on" the Enable Editing checkbox

41 **Editing Data with GridView (Page 3)**

- To format the Edit command, click GridView's smart tag and select the Edit Columns… command
  – A CommandField column is added with its ShowEditButton property set to True)
- The properties include:
  – ButtonType—either Button, Image or Link (default) (same property as for the Delete command)
  – UpdateImageUrl—an image if ButtonType is Image
  – UpdateText—text if ButtonType is Button or Link
  – Additionally Font and other stylistic properties may be specified

42 **Editing Data with GridView (Page 4)**

- Since the values written back to the table are contingent upon rules specified in the database, exceptions are generated easily
- Techniques to solve these problems when editing might include:
  – Specify that certain fields (especially the primary key which already is) are *read-only*
  – Specifying *blank strings* in place of nulls

- Create a template field and using *validation controls* (for creating a custom editing interface)

45 **Marking Fields as Read-Only**

- Identity specification and primary key fields are set automatically to *read-only*
- Making other fields read-only is a two-step process:
  - In the "Fields" dialog window set the ReadOnly property of the BoundColumn to True
  - In the source code remove the read-only field from the UpdateCommand property and UpdateParameters group for the SqlDataSource control
- In the browser the field is displayed as text (so that it is not editable) when the Edit command is clicked

46 **Specifying Blank Strings in Place of Nulls**

- It may be preferred to use blank strings (a string with no characters) in place of a null
- In the "Fields" dialog window, set value of the field's ConvertEmptyStringToNull property to False

48 **TemplateFields and Validation Controls (Page 1)**

- A TemplateField is a collection of templates, that is a mix of Web controls and static HTML markup
- Allows developer/programmer to *customize* a data bound control

49 **TemplateFields and Validation Controls (Page 2)**

- To convert a BoundField to a TemplateField:
  - In the "Fields" dialog window, select the column name and click Convert this field into a TemplateField link below the property list (*cannot be undone*)
  - Substitutes ItemTemplate and EditItemTemplate blocks into the Columns block

50 **TemplateFields and Validation Controls (Page 3)**

- When the conversion is completed the following are created automatically:
  - The ItemTemplate contains a Label (display only)

```
<ItemTemplate>
   <asp:Label ID="Label1" runat="server"    Text='<%# Bind("Product") %>'>
   </asp:Label>
</ItemTemplate>
```

51 **TemplateFields and Validation Controls (Page 4)**

- When the conversion is completed the following are created automatically (*con.*):
  - The EditItemTemplate contains a TextBox

```
<EditItemTemplate>
   <asp:TextBox ID="TextBox1" runat="server"
      Text='<%# Bind("Product") %>'>
   </asp:TextBox>
</EditItemTemplate>
```

–

52 **TemplateFields and Validation Controls (Page 5)**

- The Text properties of Label and TextBox are assigned values using *data binding syntax*:
  - Format:
    <%# Bind(*columnName*, [*formatSpecifier*] %>
    - This is a Bind directive that binds the Label or TextBox to a field (column) from the data source
    - Replaces the DataField property
  - Example:
    <%# Bind("Product") %>

53 **TemplateFields and Validation Controls (Page 6)**

- TemplateFields may be edited in "Design" view:
  - Go back to and click the GridView's smart tag again and click the Edit Templates link
  - Select one of five template types from the Display: drop-down list
  - A visual representation of the template appears where any formatting and additional Web elements may be implemented (just as if it were a Web page)
  - Click End Template Editing link when finished

58 **The DataFormatString (Page 1)**

- The DataFormatString property is a property of a <BoundColumn> used to format *numeric* or *date* data in that column
- Format:
  DataFormatString="{*FormatString*}"
- Example:

```
<asp:BoundField DataField="PoTotal"
   DataFormatString="{0:N}"
   HeaderText="PO Total"
   SortExpression="PoTotal" />
```

59 **The DataFormatString (Page 2)**

- Some format string examples:
  - C—currency with dollar sign character, commas and decimal positions (default is 2 decimals):
    - DataFormatString="{0:C}"
      - 1234.567 displays as $1,234.57
  - N—number with commas and decimal positions:
    - DataFormatString="{0:N}"
      - 1234.567 displays as 1,234.57
    - DataFormatString="{0:N1}"
      - 1234.567 displays as 1,234.6

60 **The DataFormatString (Page 3)**

- Some format string examples (*con.*):
  - d—is *short date*:
    - DataFormatString="{0:d}"
      - 4/12/2011 12:00:00 AM displays as 4/12/2011
  - D—it *long date*:
    - DataFormatString="{0:D}"
      - 4/12/2011 12:00:00 AM displays as Tuesday, April 12, 2011

61 **The ItemStyle Property**

- The <ItemStyle> property is a tag element placed inside a <BoundColumn> or other field-level block used to format that column only
- Styles include BackColor, ForeColor, FontNames, HorizontalAlign, VerticalAlign, etc.
- Format:
  <ItemStyle *Property1=Value1* [, *Property2=Value2*, …] />
- Example:
  <ItemStyle HorizontalAlign="Right" />

63 **GridView's Fields (Page 1)**

- BoundField—displays corresponding column of a data source value as text, or as a TextBox for editing or inserting
- ButtonField—renders a Button Web control for user to initiate some action on a record-by-record basis (other than editing, deleting or selecting)
- CheckBoxField—renders corresponding bit (Boolean) type column of a data source

64 **GridView's Fields (Page 2)**

- CommandField—renders an interface for inserting, updating, deleting or selecting records (added by clicking "on" CheckBox through control's smart tag
- HyperLinkField—creates a Hyperlink control whose Text and NavigateUrl properties can be hard-coded or based on column values from data source
- ImageField—renders an Image Web control whose ImageUrl property is based on column value from data source

65 **GridView's Fields (Page 3)**

- TemplateField—allows for a mix of static HTML, Web controls and data binding syntax

66 **The CheckBoxField (Bit Columns)**

- Typically used to store a Boolean value:
  - True/False or Yes/No
- Created automatically for a Boolean field in GridView or DetailsView when the data source is assigned using Configure Data Source: from the smart tag
- Appearance properties can be customized as usual

70 **The ImageField (Page 1)**

- Displays an image whose URL is based upon one of the database values for a record
- Has similar dynamic properties and functionality to those of the HyperLinkField control:
  - DataImageUrlField—specifies what database column value(s) are used in the text or URL
  - DataImageUrlFieldString—surrounds the dynamic value from the database with additional static text

71 **The ImageField (Page 2)**

- To create an ImageField select the Edit Columns … command after clicking smart tag for the GridView or DetailsView control
- Click ImageField from within the Available fields: list
  - Use the up and down arrows to position the new field in the desire column within the Selected fields:
- There is no property for specifying static URL values

72 **Using the DataImageUrlFieldString Property**

- Combine the static URL text with dynamic value(s) represented by position in the DataImageUrlField
- Its position is a zero-based integer value (similar to a parameter) within a set of braces, e.g. {0}
- Format:
  DataImageUrlFieldString="*pathAndFileName_Including*{*integer*}"
  - The *integer* always is {0}
- Example:
  Images/{0}.jpg

75 **The HyperLinkField (Page 1)**

- Displays a field (column) of HyperLink Web controls (one for each record/row)
- Renders a link that takes user to a specified URL
- Can be based upon database values so the link for each row can take user to a different Web page based upon one of the database values for the record

76 **The HyperLinkField (Page 2)**

- To create a HyperLinkField select the Edit Columns … command after clicking the smart tag for the GridView or DetailsView control
- Click the HyperlinkField from within the Available fields: list
  - Use the up and down arrows to position the new field in the desire column within the Selected fields:
- To set the URL to a static value, modify the Text of NavigateURL properties

77 **The HyperLinkField (Page 3)**

- To set the URL to a value based upon dynamic values from the database, modify two properties:
  - DataNavigateUrlFields—specifies which column or columns value(s) from the data

source are used in the DataNavigateUrlFormatString
- DataNavigateUrlFormatString—surrounds the dynamic value from the database with additional static text

Also set a Text property to be displayed in the column

78 **Using the DataNavigateFormatString Property**

- Combines static URL text with dynamic value(s) represented by position in DataNavigateUrlFields
- Position is a zero-based integer value (similar to a parameter) within a set of braces, e.g. {0} … {1} …
- Format:
  DataNavigateUrlFormatString="*staticText_including*{*integers*}"
- Example:
  DataNavigateUrlFormatString="http://www.TaraStore.com/{0}/{1}.htm"

82 **Query Strings (Page 1)**

- A query string is part of a URL that used to send the values from one webpage to other page (like passing arguments)
- Consists of a series of "key and value" pairs
  - The key is the "variable" name
  - The value is a value assigned (=) to the key
- Multiple keys and values may be joined by using the ampersand (&) character and passed together in the same URL, e.g.:
  - *key1=value1*[&*key2=value2* …]

83 **Query Strings (Page 2)**

- In a URL the key and value pairs follow the Web address and a question mark (?) symbol:
  http://*WebAddress*?*key1=value1*[&*key2=value2* …]
  - Each of the *key* and *value* pairs are separated by an ampersand (&) symbol
- Example:
  http://www.MyWebSite.com/VendorDetailsView.aspx?VendorNumber=20001
- Example when the file has the same path (location):
  VendorDetailsView.aspx?VendorNumber=20001

84 **The Response.Redirect Method**

- Causes the browser to redirect the client to (load) a different URL (web address) instead of reposting to itself
- Format:
  Response.Redirect("*WebAddress*");
  - The *WebAddress* is a URL passed as a string
- Examples:
  Response.Redirect("http://www.MyWebSite.com/VendorDetailsView.aspx);
  Response.Redirect("VendorDetailsView.aspx);

85 **Passing a Query String Through the Response.Redirect Method**

- A query string may be included within the string argument of the Response.Redirect method
- Format:

  Response.Redirect("*WebAddress*?*key1*=*value1*[&*key2*=*value2* …"]);
- Examples:

  Response.Redirect("http://www.MyWebSite.
  com/VendorDetailsView.aspx?VendorNumber=20001");

  Response.Redirect("VendorDetailsView.aspx?VendorNumber=20001");

90 **The Request.QueryString Method (Page 1)**

- Retrieves (returns) the *collection* of key and value pairs from an HTTP query string
- Format 1:

  Request.QueryString["*keyName*"]
  - The *KeyName* is a string which is the name (*key*) as assigned when the original query string created by the Response.Redirect method
- Example 1:

  TextBoxVendorNumber.Text = Request.QueryString["VendorNumber"]

91 **The Request.QueryString Method (Page 2)**

- Format 2:

  Request.QueryString[*index*]
  - The *index* is an integer which is the key's position (zero-based) in the original query string created by the Response.Redirect method
- Example 2:

  TextBoxVendorNumber.Text = Request.QueryString[0]

92 **QueryStringParameters in a SqlDataSource Select Command**

- A QueryStringParameter for a SELECT command automatically retrieves values from the query string
- When configuring the SqlDataSource:
  - Click the <WHERE…> button and select the Column (field) for the WHERE clause and the Operator
  - From the Source drop down list select "Query String"
  - Enter the name (*key* from the query string) into the QueryString field textbox
- This process inserts an asp:QueryStringParameter into the SqlDataSource's SelectParameter block and sets its QueryStringField property to the query string *key*

95 **Filtering Records with SQL WHERE Clause in the SqlDataSource (Page 1)**

- One control on the Form to be used in an expression that *filters* the records returned in the WHERE clause of SqlDataSource's SelectCommand
  - This topic was covered in Hour 14
- The contents of the specified control is used as the *variable* information to which the field is compared

  - Stored as an <asp:ControlParameter> tag within the <SelectParameters> block in the source code
- The SELECT command is configured as usual
- The Return only unique rows check box is used to group (summarize) rows on a given field

96 **Query Strings in DataNavigateURLFormatString**

- Example of a query string in the DataNavigateURLFormatString property of a HyperlinkField:
  - Products.aspx?ProductNumber={0}&Product={1}

97 **Filtering Records with SQL WHERE Clause in the SqlDataSource (Page 2)**

- To build the filter click the <WHERE...> button:
  - Column:—select the field upon which the WHERE criteria is based
  - Operator:—select a relational operator (=, <, >, etc.)
  - Source:—the type of object that stores the value to be used in the comparison, for example a Control
  - Control ID:—select the control upon which the WHERE criteria is based
  - Then click the <Add> button to create the WHERE clause for the SQL SELECT statement

98 **Wildcards in a SQL "Where" Filter (Page 1)**

- The LIKE operator is used in SQL to implement wild card processing in a WHERE clause
- The wild card is specified as a percent (%) character which means any single or multiple characters
- Examples:
  - Product LIKE "P%"
    - Meaning the *Product* field starts with the letter "P" followed by any number of characters
  - Product LIKE "%PEN%"
    - Meaning the *Product* field contains the letters "PEN" either at beginning, somewhere within, or at the end

99 **Wildcards in a SQL "Where" Filter (Page 2)**

- To include a LIKE operator within a SqlDataSource:
  - Configure data source as usual including specifying the SQL Select statement
  - Click the <WHERE> button and for the Operator: value drop-down list select LIKE
  - Set a Source: for the LIKE to compare, for example a Control like a TextBox, and set the ControlID: property to name it)

100 **Wildcards in a SQL "Where" Filter (Page 3)**

- Unless a default value is set for the Web control that specifies the WHERE parameter value, no records will be displayed the first time the page is visited
  - Alternately the EmptyDataText property may be set to display a message if no

records are returned

- Separate formatting properties may be set for the EmptyDataText message

- LIKE only can be applied to string (e.g. char, nchar, varchar or nvarchar) or date/time database columns