# An Alternate Approach to a GridView

Not every screen needs to be a grid with rows and columns. However, I see more grid screens than almost any other interface on webpages. With just a little creative thought, you can present data to the user that will make it easier for them to view that data.

I have long had a problem with using grids to display data to the user. I have an even bigger problem with editing on a grid. A grid is easy to implement for a developer, and this is normally why developers use them. However, a grid is not always the best method for conveying data to a user. Of course there are always exceptions, but these should be “the exception” and not the rule.

Some reasons why a grid is not suitable for user consumption:

* A grid presents too much data on the screen so the user’s eye has too much to try to concentrate on.
* A user will get much more tired at the end of the day using a grid as opposed to other display types.
* The user cannot distinguish between the data in each column since each column is very uniform and nothing stands out.
* The header for the data can scroll out of sight and thus the user can no longer distinguish between the different columns of data.

The list above includes just some of the reasons why a grid might not be the appropriate choice for displaying a list of data to the user. Just take a look at **Figure 1** and then take a look at **Figure 2** and see which one you think is easier to read.

## A Normal GridView

I would like you to look at both **Figure 1** and **Figure 2**. **Figure 1** shows a default GridView control using normal column definitions. However, I want you to learn how to use the same GridView control to create a screen that looks like **Figure 2.**

To create both versions of this customer list shown in **Figure 1** and **Figure 2** you will start by dragging a SqlDataSource object from your toolbox onto a new ASP.NET Web Form page. For this sample I am going to use the AdventureWorksLT database that you can download from Microsoft’s website for your SQL Server. Go into Design view on the Web Form page and drag a SqlDataSource onto your page. Use the Smart Tag that appears after adding the SqlDataSource to configure the data source.

Using the ObjectDataSource

In this article I used a SqlDataSource object because it was the easiest method to get some data into the GridView without having to cover some data access technology like Entity Framework, LINQ to SQL or some other data access method. However, everything that I presented here using the SqlDataSource can also be accomplished using an ObjectDataSource instead.

I prefer an ObjectDataSource because it allows you to connect to a data class that wraps up the SQL calls instead of having SQL directly in your front-end UI code. All you need in your data classes is some method to return a DataSet or collection object that can be sorted and filtered. If you have this type of data object then all of the code and HTML presented in this article should work with little or no changes.

In the wizard that appears you will make a connection to the AdventureWorksLT database. Next, you will select the Customer table as the source of the data. When prompted for the list of columns, select CustomerId, Title, FirstName, LastName, CompanyName, EmailAddress and Phone. Click on the “Order By” button and choose the CompanyName field. Click the Finish button and switch to the Source view of the ASP.NET webpage. When you are finished configuring the SqlDataSource, the ASP.NET code will look like the following listing:

<asp:SqlDataSource

ID="custData"

runat="server"

ConnectionString="<%$

ConnectionStrings:AdvWorksConnectionString %>"

SelectCommand="SELECT [CustomerId], [Title],

[FirstName],

[LastName], [CompanyName],

[EmailAddress], [Phone]

FROM [Customer]

ORDER BY [CompanyName]">

</asp:SqlDataSource>

Unfortunately, there is a bug with the SqlDataSource object. It does not understand that the Customer table is located in the SalesLT schema. You will need to manually fix the above SelectCommand and add [SalesLT] with a period before the [Customer] text. Your fixed-up FROM statement should look like the following:

FROM [SalesLT].[Customer]

Switch back to Design view of your webpage and drag an ASP.NET GridView control from the toolbox onto the page. Use the Smart Tag to set the data source to the SqlDataSource object you just finished creating. Once you set the data source, click off the smart tag and press F5 to run your webpage. You should now see a very boring looking grid of all of the customer data. If you want you can click on the smart tag back in Design view and click Auto Formats to give the grid a more appealing look.

Sample Code

You can download the sample code for this article by visiting my website at <http://www.pdsa.com/downloads>. Select “PDSA Articles” and then select “Code Magazine - An Alternate Approach to a GridView” from the drop down list.

If you look at the GridView in source mode you will just see a lot of formatting and no column definitions. The reason is because there is an AutoGenerateColumns property which, by default, is set to true (see **Listing 1**). This causes the grid to get the column names from the SELECT statement in the SqlDataSource object. You can add grid lines in between the rows and columns by changing the GridLines property from “None” to “Both” in the Properties window. All of this will produce a grid that should look similar to **Figure 1**.

Figure 1: The normal GridView control displays rows and columns with some basic colors and formats.

To make this grid look a little better you might want to explicitly define each column instead of having the GridView just use the column names for the header text. Click on the Smart Tag on the GridView control and choose Edit Columns. In the dialog box that comes up, add a BoundField for each of the columns in your SELECT statement. Set the HeaderText property to a more readable name such as “First Name” instead of “FirstName”. Set the DataField and SortExpression properties to the name of the column. After adding each column from your table, the source for your GridView will now contain the following columns definitions.

<Columns>

<asp:BoundField HeaderText="Title"

DataField="Title" SortExpression="Title" />

<asp:BoundField HeaderText="First Name"

DataField="FirstName"

SortExpression="FirstName" />

<asp:BoundField HeaderText="Last Name"

DataField="LastName"

SortExpression="LastName" />

<asp:BoundField HeaderText="Company Name"

DataField="CompanyName"

SortExpression="CompanyName" />

<asp:BoundField HeaderText="Email Address"

DataField="EmailAddress"

SortExpression="EmailAddress" />

<asp:BoundField HeaderText="Phone"

DataField="Phone" SortExpression="Phone" />

</Columns>

## Using a TemplateField

The BoundField object that you just created is great for displaying a single field within one column of the Grid. However, our goal here is to eliminate columns and display the data in a more appealing format. For this you can use a TemplateField. In a TemplateField you can define any HTML you want. With just a little creativity you can create a much better layout than that shown in **Figure 1**. You will now see how to create the GridView to look like **Figure 2**.

A TemplateField is a better choice than just a normal BoundField in a GridView control. You can define any HTML you want within a TemplateField.

Figure 2: Using a template will make your GridView easier to read for the user.

In order to create the second version of the GridView you will first need to click on the GridView control and in the Properties window set the AutoGenerateColumns property and the ShowHeader property to false. Now you need to add a TemplateField and eliminate all of your other BoundField columns. I know, I just had you create them, but I wanted to make sure you understood where columns came from. When working with TemplateField columns, you will mostly work right in the HTML/Source view of your grid.

The first step is to remove all BoundField columns from within the <Columns> tag in your GridView. Go ahead and remove all BoundField columns now. Then, within the <Columns> tag, add an <asp:TemplateField> and within that add an <ItemTemplate> tag. Your result should like the following:

<Columns>

<asp:TemplateField>

<ItemTemplate>

</ItemTemplate>

</asp:TemplateField>

</Columns>

Now you can create the HTML within the <ItemTemplate> tag to make the GridView look like **Figure 2**. The HTML code to create this version of the GridView is shown below. Again, all of the formatting code has been removed to keep the source code simple.

As you can see, writing this HTML code will take a little more thought and a little more typing, but your users will thank you. The reasons the second version of the grid is better than the first are:

1. The most important piece of data, the Company Name, is in a larger font than the other data that is less important.
2. You keep the labels close to the data as opposed to in headers on the grid that will eventually scroll out of sight.
3. Having the data go top left to bottom right in terms of importance is how most people in western cultures tend to process data.

## Add Paging

The great thing about using this alternate view in the GridView control is you can still use all the normal sorting, paging and filtering that you would with a normal tabular GridView. Let’s now hook up Paging to our GridView. Go into Design view in Visual Studio and click on your GridView control. In the Properties window, set the AllowPaging property to true and the PageSize property to 2. After setting these two properties, run your webpage and you will see a page that looks similar to **Figure 3**.

If you style a GridView using TemplateField controls you still get the advantages of Paging, Sorting and Filtering. You just get a better look and feel.

Figure 3: Paging is accomplished with some property settings. No code required.

## Add Filtering

Another feature that is often requested by users is to be able to filter the data by some field or set of fields. Add a TextBox control above the GridView to allow the user to input a partial company name. You will also add two Button controls. One will perform the search function and one will reset the text box to a blank string. Add the HTML code shown in **Listing 2** to create the screen shown in **Figure 4**.

To hook up filtering you simply add a FilterExpression and a FilterParameters element to the SqlDataSource. Once again, there is no UI for this, so you must edit your HTML code in Source view on your webpage. Make your SqlDataSource look like the following code. I have removed the pieces you don’t change for brevity.

<asp:SqlDataSource ID="custData"

...

FilterExpression="CompanyName LIKE '{0}%'">

<FilterParameters>

<asp:ControlParameter Name="CompanyName"

ControlID="txtCompany"

PropertyName="Text" />

</FilterParameters>

</asp:SqlDataSource>

Now you can run your webpage, enter a value like ‘F’ or ‘S’ in the text box, and you will see that the GridView is automatically filtered. You have bound the text box control named “txtCompany” to the FilterExpression within the SqlDataSource. The SqlDataSource object will rebuild the SQL statement on the fly to include the WHERE clause using the FilterExpression and the value from the txtCompany text box.

Figure 4: Just like paging, no code behind is required for filtering.

The last thing to do is to bring up your webpage in Design view and double-click on the Reset button in order to create the Click event procedure for this button. You can now add one line of code to reset the company name text box to an empty string. This will not only reset the text box but will also cause the SqlDataSource to reset the filter and display all rows of data in your GridView control.

protected void btnReset\_Click(object sender,

EventArgs e)

{

txtCompany.Text = string.Empty;

}

## Add Sorting

Another common feature of the GridView control is the ability to sort the data. Normally this is accomplished by setting the AllowSorting property to true, and then the user just needs to click on the HeaderText and the GridView will sort itself on that column. Since you have removed the headers from the GridView, you need to come up with a new way to perform the sorting.

As shown in **Figure 5**, I added a DropDownList control to the webpage with a list of columns that the user can sort by. Add the following code within the <fieldset> element you added earlier.

<div class="clear">

<div class="label">

<asp:Label runat="server" Text="Sort By" />

</div>

<div class="control">

<asp:DropDownList ID="sortFields"

runat="server" AutoPostBack="True">

<asp:ListItem Value="CompanyName"

Text="Company Name" />

<asp:ListItem Value="LastName"

Text="Last Name" />

<asp:ListItem Value="EmailAddress"

Text="Email Address" />

</asp:DropDownList>

</div>

</div>

You are adding class names to the <div> tags, but you have not added any styles yet. I will show you the styles used for these <div> tags later in this article.

Figure 5: To sort the grid you will need to add a DropDownList of columns, and a line of code in the SelectedIndexChanged event of that DropDownList control.

To perform the sorting on the GridView control the user will select one of the values from the DropDownList control. You set the AutoPostBack property to true on the DropDownList so when you change the value, a post back will occur. Now in Design view, double-click on the DropDownList to generate the SelectedIndexChanged event procedure. In this event you will use the SelectedValue property from the DropDownList. This property contains a column name. You pass this value to the Sort method on the GridView control. You can also set Ascending or Descending. If you want to, add two Radio Button controls and allow the user to select Ascending or Descending.

protected void sortFields\_SelectedIndexChanged(

object sender, EventArgs e)

{

grdCust.Sort(

sortFields.SelectedValue.ToString(),

SortDirection.Ascending);

}

Now run the page and select a column from the DropDownList control and watch the data sort within your GridView control.

## Styling the Page

To make your page a little nicer looking, you might want to add a style sheet with some styling. In the final version shown in **Figure 5** I have an <h2> tag with “Customer List” at the top of the page. I also styled the buttons a little so the solution does not have the default button look. Another enhancement was to remove the hard-coded “style” tag in the <div> tag around the CompanyName HTML in the TemplateField and replace it with a class called “customerNameTitle”. I also added some styles for the <fieldset> elements. You can download the complete project with all the styles and final version of the grid to see everything added. **Listing 3** shows all of the styles.

## Summary

In this article I presented you with a method to present data to the user that is different from a traditional grid view. While you may not like the exact user interface presented here, hopefully this article will spur you to question your user interfaces a little more and come up with some alternate methods of displaying data. We have found that our users find user interfaces that are not so busy and show important data in a larger font size easier to use.

Paul D. Sheriff

[PSheriff@pdsa.com](mailto:PSheriff@pdsa.com)   
(714) 734-9792

Paul D. Sheriff is the President of PDSA, Inc. (www.pdsa.com), a Microsoft Partner in Southern California. Paul acts as the Microsoft Regional Director for Southern California assisting the local Microsoft offices with events and evangelism. Paul has authored literally hundreds of books, webcasts, videos and articles on .NET, WPF, Silverlight, HTML 5, Windows Phone and SQL Server. You can reach Paul via email at PSheriff@pdsa.com. Follow Paul at @pdsainc and facebook.com/PDSApaulsheriff.

**Listing 1:** HTML code to create a better GridView layout

<asp:GridView AutoGenerateColumns="false" ShowHeader="false">

<Columns>

<asp:TemplateField>

<ItemTemplate>

<div>

<div style="font-size: x-large">

<%# DataBinder.Eval(Container.DataItem,

"CompanyName") %>

</div>

<div>

Contact: <%# DataBinder.Eval(Container.DataItem,

"FirstName")%>&nbsp;

<%# DataBinder.Eval(Container.DataItem,

"LastName")%>&nbsp;&nbsp;

(<%# DataBinder.Eval(Container.DataItem,

"EmailAddress")%>)

</div>

<div>

Phone: <%# DataBinder.Eval(Container.DataItem,

"Phone")%>

</div>

<br />

<div>

<asp:Button ID="btnEdit" Text="Edit"

CommandArgument='<%#

DataBinder.Eval(Container.DataItem,

"CustomerID")%>'

CommandName="Edit" runat="server" />

<asp:Button ID="btnDelete" Text="Delete"

CommandArgument='<%#

DataBinder.Eval(Container.DataItem,

"CustomerID")%>'

CommandName="Delete" runat="server" />

</div>

</div>

</ItemTemplate>

</asp:TemplateField>

</Columns>

</asp:GridView>

**Listing 2:** Add a fieldset with controls to handle filtering of your GridView

<fieldset>

<legend>Customer Search Filter</legend>

<div class="clear">

<div class="label">

<asp:Label runat="server"

Text="Search by Company Name" />

</div>

<div class="control">

<asp:TextBox ID="txtCompany" runat="server" />

</div>

</div>

<div class="clear">

<div class="label">

&nbsp;

</div>

<div class="control">

<asp:Button ID="btnSearch" runat="server"

Text="Search" />

<asp:Button ID="btnReset" runat="server"

Text="Reset" />

</div>

</div>

</fieldset>

**Listing 3:** Adding styles can make your application look a little nicer

h2

{

margin: 0.5em 0.5em 0.5em 0.5em;

}

input.submit

{

color: #050;

background-color: #fed;

font: bold 84% 'trebuchet ms' ,helvetica,sans-serif;

border-left: 0.08em solid #696;

border-right: 0.08em solid #363;

border-top: 0.08em solid #696;

border-bottom: 0.08em solid #363;

margin: 0.2em 0.2em 0.2em 0.2em;

}

fieldset

{

margin: 0.5em 0.5em 0.5em 0.5em;

padding: 0.5em 0.5em 0.5em 0.5em;

}

fieldset .clear

{

clear: both;

margin-top: 0.25em;

margin-bottom: 0.25em;

}

fieldset .label

{

position: relative;

float: left;

width: 14em;

}

fieldset .control

{

position: relative;

float: none;

}

.customerBlock

{

border: 0.08em solid gray;

margin: 0.5em 0.5em 0.5em 0.5em;

padding: 0.5em 0.5em 0.5em 0.5em;

}

.customerNameTitle

{

font-size: x-large;

}

.customerList

{

margin: 0.5em 0.5em 0.5em 0.5em;

}