GridView C#

Introduction:

**GridView** is a powerful and versatile data presentation control in C# that allows developers to display data in a tabular format with sorting, paging, and editing capabilities. Data from a database or other information sources can be shown using this method frequently in online applications. In this article, we will explore GridView in C# in detail, including its features, properties, and how to use it to display data.

What is GridView in C#?

Data can be displayed in a tabular manner with sorting, paging, and editing options using the C# data presentation control **GridView**. The namespace ***Web.UI.WebControls*** is frequently used in web applications. **GridView** can be bound to various data sources like databases, XML, and other data sources.

Features of GridView in C#:

**GridView** in C# has several features that make it a popular choice for displaying data in web applications. Some of the key features of GridView are:

* **Sorting:**

**GridView** allows the sorting of data in ascending or descending order. By pressing and holding down the shift key while clicking on the column headers, you can sort across several columns.

* **Paging:**

**GridView** supports the paging of data, which allows you to display a subset of data at a time. This helps to increase the application's performance when dealing with large amounts of data.

* **Editing:**

**GridView** provides editing capabilities, which allow users to edit the data directly in control. This feature is useful when the user needs to make changes to the data displayed in the control.

* **Selection:**

**GridView** allows the selection of data rows, which can be utilized to perform various tasks on the selected rows, like deletion, updating, or exporting.

* **Formatting:**

**GridView** provides formatting options for data displayed in the control. You can manipulate the font, size, color, and other formatting options for the data.

Properties of GridView in C#:

**GridView** in C# has several properties that can be used to customize its appearance and behavior. Some of the key properties of GridView are:

* **AutoGenerateColumns:**

This property allows you to automatically generate columns based on the data source.

* **AllowSorting:**

This property allows you to enable or disable data sorting in the **GridView**.

* **AllowPaging:**

This property allows you to enable or disable the paging of data in the **GridView**.

* **ShowHeader:**

This property allows you to show or hide the header of the **GridView**.

* **ShowFooter:**

This property allows you to show or hide the footer of the **GridView**.

* **EditIndex:**

This property allows you to set the index of the row being edited in the **GridView**.

Using GridView in C#:

Using **GridView** in C# is easy and straightforward. The first thing to do is to give your web form a GridView control. We can implement it by dragging and dropping the control from the toolbox onto your form.

Next, you need to bind the **GridView** control to your data source. Setting the data source as the **GridView** control's DataSource attribute will do this. If you wish to automatically create columns based on the data source, you can also set the AutoGenerateColumns attribute to true.

Once you have bound the **GridView** control to your data source, you can customize its appearance and behavior by setting various properties like AllowSorting, AllowPaging, and EditIndex.

Moreover, **GridView** has a number of events that you can utilize to modify its behavior. Some of the key events of **GridView** are:

* **RowDataBound:**

This event is fired when a row is bound to the **GridView**. This event allows you to change how the row looks.

* **RowEditing:**

This event is triggered whenever a row in the **GridView** is edited. This event can be used to validate data or execute other actions on the row.

* **RowDeleting:**

This event is triggered whenever a row in the **GridView** is deleted. You can use this event to confirm the deletion or perform any other operation before deleting the row.

* **PageIndexChanged:**

This event is fired when the user changes the page in the **GridView**. You can use this event to rebind the **GridView** with the new data.

Let's take a simple example of using **GridView** in C#. In this example, we will bind the **GridView** control to a dataset and display the data in a tabular format with sorting and paging capabilities.

1. First, add a **GridView** control to your web form.

2. Next, create a dataset and add some data to it.

**C# Code:**

1. DataSet ds = **new** DataSet();
2. DataTable dt = **new** DataTable();
3. dt.Columns.Add(**new** DataColumn("ID", **typeof**(**int**)));
4. dt.Columns.Add(**new** DataColumn("Name", **typeof**(**string**)));
5. dt.Columns.Add(**new** DataColumn("Age", **typeof**(**int**)));
6. DataRow dr = dt.NewRow();
7. dr["ID"] = 1;
8. dr["Name"] = "John";
9. dr["Age"] = 25;
10. dt.Rows.Add(dr);
11. dr = dt.NewRow();
12. dr["ID"] = 2;
13. dr["Name"] = "Mary";
14. dr["Age"] = 30;
15. dt.Rows.Add(dr);
16. ds.Tables.Add(dt);

3. Bind the GridView control to the dataset.

**C# Code:**

1. GridView1.DataSource = ds.Tables[0];
2. GridView1.DataBind();

4. Enable sorting and paging in the GridView control.

**C# Code:**

1. GridView1.AllowSorting = **true**;
2. GridView1.AllowPaging = **true**;
3. GridView1.PageSize = 5;

5. Run the application and see the data displayed in the **GridView** control.

Conclusion:

The **GridView** data presentation interface in C# is strong and flexible, allowing you to display data in a tabular manner with sorting, paging, and editing options. Data from a database or other data sources are frequently displayed using it in web applications. In this article, we explored **GridView** in C# in detail, including its features, properties, and how to use it to display data. We also provided an example of using **GridView** in C# to display data from a dataset. With its rich set of features and properties, **GridView** in C# is an excellent choice for displaying data in web applications.

**What is repeater control in asp.net and c#**

The Repeater control in ASP.NET is a data-bound control that allows you to create custom templates for displaying data from a data source. It provides a flexible and fine-grained way to define the presentation of data, making it a versatile choice for displaying data in a tabular or list format on a web page.

Here are some key features and characteristics of the Repeater control in ASP.NET and C#:

1. **Custom Templates**: The Repeater control allows you to define custom templates for various parts of the control, such as the item template for individual data items, header template for column headings, and footer template for summary information.
2. **Data Binding**: You can bind the Repeater control to a data source, such as a database, a collection, or an array of objects. The control will automatically iterate over the data source and render the templates for each item.
3. **Data Source Flexibility**: The data source can be any IEnumerable, including datasets, data tables, lists, arrays, or custom objects. This flexibility allows you to work with various data structures.
4. **Data Binding Expressions**: You can use data binding expressions, like **<%# Eval("ColumnName") %>**, to extract and display data from the data source within the templates.
5. **Complete Control**: Unlike some other data-bound controls like GridView or DataList, the Repeater provides complete control over the generated HTML markup. This makes it suitable for creating highly customized and semantically rich HTML structures.

Here's a simple example of using a Repeater control in ASP.NET and C#:

aspxCopy code

<asp:Repeater ID="myRepeater" runat="server"> <ItemTemplate> <div> <h3><%# Eval("ProductName") %></h3> <p>Price: <%# Eval("Price", "{0:C}") %></p> </div> </ItemTemplate> </asp:Repeater>

In the code-behind (C#), you would bind data to the Repeater control:

csharpCopy code

protected void Page\_Load(object sender, EventArgs e) { if (!IsPostBack) { // Replace this with your data retrieval logic (e.g., from a database or collection) var data = YourDataAccessClass.GetData(); myRepeater.DataSource = data; myRepeater.DataBind(); } }

In this example, the Repeater control is used to display a list of products. The **Eval** method is used within the item template to extract and display data from the data source.

The Repeater control provides a lot of flexibility for creating custom data presentation, but it also requires more manual coding compared to some of the other data-bound controls like GridView or DataList. Depending on your specific requirements, the Repeater may be a good choice for your ASP.NET web application.