

Creating a Spreadsheet-like Editable Grid with ComponentOne GridView for ASP.NET

Article ID: 1919
Applies To: WebGrid for ASP.NET
Author: Greg Lutz
Published On: 6/12/2009

Answer

Here at ComponentOne, we get asked all the time if our ASP.NET grids support editing data like a spreadsheet or like a WinForms grid control. In other words, developers want to know if end-users can edit all cells in all rows of the grid at once, rather than the clunky, one row at a time, editing method typically seen in ASP.NET grids. In this article I will go over an easy to understand approach to achieving this type of functionality using ComponentOne GridView for ASP.NET (C1GridView).

Part 1: Getting Started

Before you start, you'll need to have ComponentOne Studio for ASP.NET downloaded and installed. This includes the C1GridView control in the C1.Web.UI.Controls assembly.

Download Studio for ASP.NET

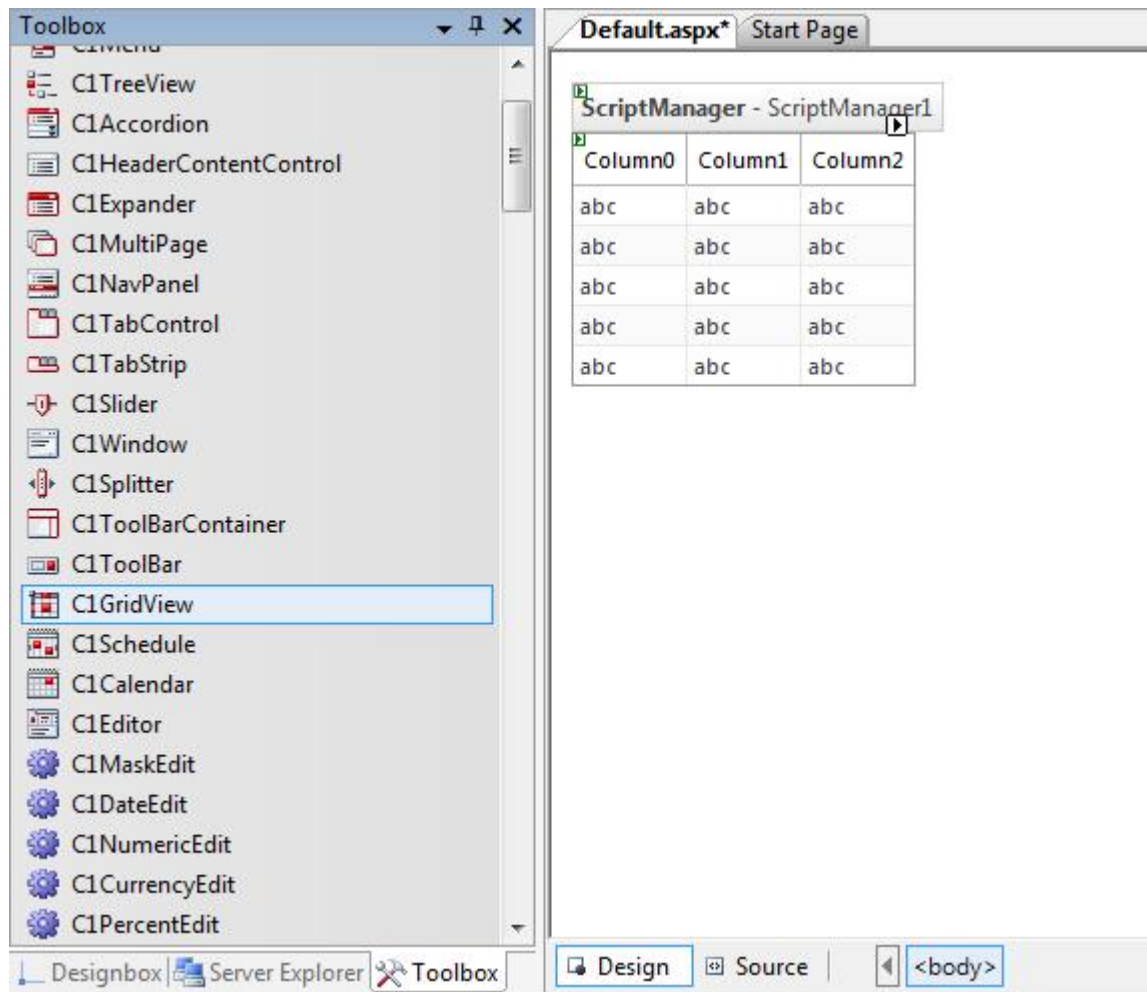
1. Create a new Web site project in Microsoft Visual Studio.

Note: C1GridView is supported under the 2.0 or 3.x Frameworks and can be used in either Visual Studio 2005 or 2008 environments. You can choose the framework and the environment. The code in this tutorial will be in C#.

2. If your app doesn't already have a ScriptManager on the form, drag and drop one from your Toolbox onto your page. It should be the first control within the <form> tag of your page.

Note: The ScriptManager control is part of the AJAX Extensions and may not already be installed on your machine. If you don't have the AJAX Extensions you can download them from <http://www.asp.net/ajax/>.

3. Drag and drop the **C1GridView** control from your Toolbox onto your page.



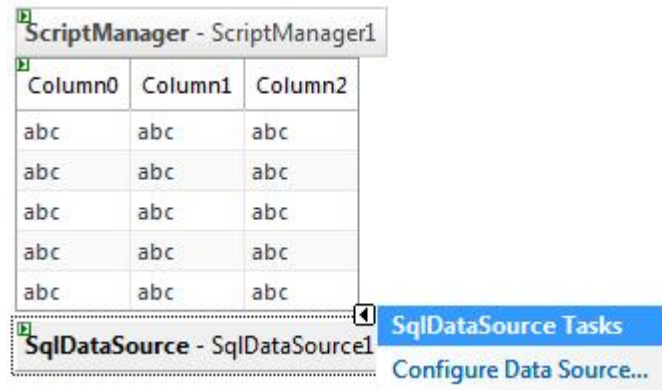
Note: If your site is .NET 2.0 then the reference added for C1GridView should be C1.Web.UI.Controls.2. If your site is .NET 3.x the reference should be C1.Web.UI.Controls.3. You will also need the corresponding C1.Web.UI and C1.Web.UI.Design assemblies added as references to your site. So once you've added C1GridView to your page, check the References directory for these assemblies. If you're missing any you can add them manually by going to "Website | Add Reference" and browse to **C:\Program Files\ComponentOneStudio.NET2.0\bin**.

4. Add a copy of the Nwind.mdb database to your site's directory.

This demo will be using a few of the tables from the Nwind database. You can add this to your Web site by right-clicking the App_Data folder in your Solution Explorer and selecting "Add Existing Item". Then browse to **C:\Program Files\ComponentOne Studio .NET2.0\Common** and select the **Nwind.mdb** file.

Part 2: Configuring the Data Source

1. From the Toolbox, drag and drop a SqlDataSource to your page.
2. Click the smart tag arrow on the upper-right corner of the SqlDataSource to open the "SqlDataSource Tasks" menu.
3. From the "SqlDataSource Tasks" menu select **Configure Data Source**.



4. On "Choose Your Data Source", select **Nwind.mdb** from the drop-down box.
5. Click **Next**.
6. Click **Next** again to save the connection string.
7. On "Configure the Select Statement", select the following columns from the Employees table: *EmployeeID*, *LastName*, *FirstName*, *Title*, and *HireDate*.

Configure Data Source - SqlDataSource1

Configure the Select Statement

How would you like to retrieve data from your database?

Specify a custom SQL statement or stored procedure

Specify columns from a table or view

Name:

Employees

Columns:

<input type="checkbox"/> *	<input checked="" type="checkbox"/> HireDate	<input type="checkbox"/> Extension	<input type="checkbox"/> Return only
<input checked="" type="checkbox"/> EmployeeID	<input type="checkbox"/> Address	<input type="checkbox"/> Photo	<input type="checkbox"/> WHER
<input checked="" type="checkbox"/> LastName	<input type="checkbox"/> City	<input type="checkbox"/> Notes	<input type="checkbox"/> ORDER
<input checked="" type="checkbox"/> FirstName	<input type="checkbox"/> Region	<input type="checkbox"/> ReportsTo	<input type="checkbox"/> Advanc
<input checked="" type="checkbox"/> Title	<input type="checkbox"/> PostalCode		
<input type="checkbox"/> TitleOfCourtesy	<input type="checkbox"/> Country		
<input type="checkbox"/> BirthDate	<input type="checkbox"/> HomePhone		

SELECT statement:

SELECT [EmployeeID], [LastName], [FirstName], [Title], [HireDate] FROM [Employees]

< Previous Next > Finish

8. On the same window, click the **Advanced** button.
9. On the "Advanced SQL Generation" window, check off the first item: Generate Insert, Update and Delete functions.
10. Click **Ok**, **Next**, and **Finish** wrapping up the data source configuration.

Part 3: Setting C1GridView properties.

Here we will set a few properties to make the grid's data visually appealing.

1. From the Properties window, set the **DataSourceID** property to the SqlDataSource we just created.
2. Set the **VisualStyle** property to **Office2007Blue** or any other style of your choice.
3. Set the **ShowFooter** property to **True**. Note that this is important for our Save button we'll add later.
4. Set the **RowHeader.Visible** property to **True**.

Now we have a real desktop-looking grid.

Part 4: Copy and Paste

Next we're going to replace all of the existing columns with TemplateColumns and bind the data in each column to a TextBox. This allows us to have editable cells, and by removing the borders of the TextBoxes we can create a real spreadsheet-like look to our grid. All of this can be done by clicking through the design-time Property Builder and editing the Templates; however, in this case working in the source is actually more convenient (and more paste-able).

1. Go to the Source view of the page and replace everything inside the <Columns> tag with the following code (including the Column tags)

```
<Columns>
<c1:C1BoundField DataField="EmployeeID" HeaderText="EmployeeID" ReadOnly="True" SortExpression="Emp
</c1:C1BoundField>
<c1:C1TemplateField HeaderText="Last Name" SortExpression="LastName">
<ItemTemplate>
<asp:TextBox ID="TxtLastName" runat="server" Text='<%= Bind("LastName") %>' BorderStyle="None" Width=
  OnTextChanged="TextBox_Changed"></asp:TextBox>
</ItemTemplate>
</c1:C1TemplateField>
<c1:C1TemplateField HeaderText="First Name" SortExpression="FirstName">
<ItemTemplate>
<asp:TextBox ID="TxtFirstName" runat="server" Text='<%= Bind("FirstName") %>' BorderStyle="None"
  Width="100%" OnTextChanged="TextBox_Changed"></asp:TextBox>
</ItemTemplate>
</c1:C1TemplateField>
<c1:C1TemplateField HeaderText="Title" SortExpression="Title">
<ItemTemplate>
<asp:TextBox ID="TxtTitle" runat="server" Text='<%= Bind("Title") %>' BorderStyle="None" Width="100"
  OnTextChanged="TextBox_Changed"></asp:TextBox>
</ItemTemplate> </c1:C1TemplateField> <c1:C1TemplateField HeaderText="Hire Date"
  SortExpression="HireDate">
<ItemTemplate>
<asp:TextBox ID="TxtHireDate" runat="server" Text='<%= Bind("HireDate") %>' BorderStyle="None" Width=
  OnTextChanged="TextBox_Changed"></asp:TextBox>
</ItemTemplate>
<FooterTemplate>
<asp:Button ID="BtnUpdate" runat="server" Text="Save" OnClick="Button1_Click" />
</FooterTemplate>
<FooterStyle HorizontalAlign="Right" />
</c1:C1TemplateField>
</Columns>
```

2. Now go to your code-behind page, default.aspx.cs. At the top of the page add the following reference.

```
using C1.Web.UI.Controls.C1GridView;
```

3. In your code-behind page, add the following code. We will use the affectedRows array to store which rows are modified when we save the changes made to the grid.

```
bool[] affectedRows;
protected void Page_Load(object sender, EventArgs e)
{
    affectedRows = new bool[C1GridView1.Rows.Count];
}
```

4. Next, we will add the TextChanged event for the TextBoxes. This event will fire when there are changes to be updated. We will mark each modified row's index as True in the affectedRows array. This allows us to optimize the saving procedure so we only send what needs updated to our database.

```
protected void TextBox_Changed(object sender, EventArgs e)
{
```

```

    TextBox tb = (TextBox)sender;
    C1GridViewRow row = (C1GridViewRow)tb.Parent.BindingContainer;
    affectedRows[row.RowIndex] = true;
}

```

5. Finally, we will add the Save button's click event. This event will fire-off an update command to our SqlDataSource for every affected row based upon the affectedRows array.

```

protected void Button1_Click(object sender, EventArgs e)
{
    for (int i = 0; i < C1GridView1.Rows.Count; i++)
    {
        if (affectedRows[i])
        {
            C1GridViewRow row = C1GridView1.Rows[i];
            SqlDataSource1.UpdateParameters["LastName"].DefaultValue = ((TextBox)row.FindControl("TxtLastNa
            SqlDataSource1.UpdateParameters["FirstName"].DefaultValue = ((TextBox)row.FindControl("TxtFirstN
            SqlDataSource1.UpdateParameters["Title"].DefaultValue = ((TextBox)row.FindControl("TxtTitle")).T
            SqlDataSource1.UpdateParameters["HireDate"].DefaultValue = ((TextBox)row.FindControl("TxtHireDat
            SqlDataSource1.UpdateParameters["EmployeeID"].DefaultValue = row.Cells[1].Text;
            SqlDataSource1.Update();
        }
    }
}

```

The code here is looping through each row finding ones that have been modified. For each modified row it grabs the values from all the textboxes across the column and enters them into the update parameters. Finally, it calls Update for each row as needed.

Now when you run the page you can edit any cell you want and with one post-back update all values in your database.

EmployeeID	Last Name	First Name	Title	Hire Date
1	Davolio	Nancy	Sales	5/1/1992 12:00:00 AM
2	Fuller	Andrew	Sales Manager	8/14/1992 12:00:00 AM
3	Leverling	Janet	Sales Representative	4/1/1992 12:00:00 AM
4	Peacock	Margaret	Sales Representative	5/3/1993 12:00:00 AM
5	Buchanan	Steven	Sales Manager	10/17/1993 12:00:00 AM
6	Suyama	Michael	Sales Representative	10/17/1993 12:00:00 AM
7	King	Robert	Sales Representative	1/2/1993 12:00:00 AM
8	Callahan	Lauren	Inside Sales Coordinator	3/5/1993 12:00:00 AM
9	Dodsworth	Anne	Sales Representative	11/15/1994 12:00:00 AM

Part 5: More

Why stop there? With C1GridView you can easily add so many features to your grid by just setting a bunch of properties. You can add sorting (AllowSorting = True), grouping (AllowGrouping = True), filtering (ShowFilter = True), column dragging (AllowColMoving = True), and more. To handle this fully-editable grid with these features we will have to make a small adjustment. If we move all of the code from the Button1_Click event to a separate "Save()" function, we can then call this Save function in the various C1GridView events: Sorting, Filtering, ColumnMoving and so on. By doing this we will automatically save the values just entered before performing the action on the grid.

Conclusion

This common approach to creating a fully-editable C1GridView detailed here can now be modified and enhanced to fit your needs. Whether you're using an ObjectDataSource or even a session DataSet, you can model the above approach around your data scheme. You can also use other editors instead of a TextBox inside the column EditTemplates.