**ASP.NET MVC 3–Working with WebGrid (Part 2)**

**Introduction**

On my previous post I shown how to display simple data using the new WebGrid feature. On this post I’ll continue and show a little bit more how can we configure the WebGrid and customize it.

**WebGrid Properties**

The WebGrid have a set of properties that allow us to customize it, I’ll cover here only the most common/important ones.

* **Source** – Where your data come from. Usually the model passed by the controller action.
* **DefaultSort** – Here you can define how the data will be sorted. Just inform the column name here. Example “Id” or “Name”.
* **RowsPerPage** – Quantity of records that will be shown on table.
* **CanPage** – Allow the paging.
* **CanSort** – Allow the sorting by clicking on column title.

Here is an initial sample:

1: var gridItens = new WebGrid(source: Model,

2: defaultSort: "Title",

3: rowsPerPage: 15,

4: canPage: true,

5: canSort: true);

This way we have a WebGrid receiving data from our model (Passed by the action) default sorting by the title with 15 rows per page and with paging and sorting. Easy!

**FieldNamePrefix**

An important property is the FieldNamePrefix, as you can have multiple WebGrids on the same view. This property allow us to have a prefixed string before the query string parameters. We can better check this feature by checking these two images:

[image](http://weblogs.asp.net/blogs/andrebaltieri/image_4005CDED.png)

Note that we have a parameter called “sort”, but there`s nothing specifying which WebGrid this sort belongs to. If we use the FieldNamePrefix, we can specify it, and now you can clearly see from which WebGrid the sort parameter belongs to:

[image](http://weblogs.asp.net/blogs/andrebaltieri/image_0E4AB098.png)

Now we have a prefix before the parameter sort (gridItens\_), and this way we can have multiple WebGrids on the same view and the sort or paging of each grid will not affect the other grids.

Here is the code:

1: var gridItens = new WebGrid(source: Model,

2: defaultSort: "Title",

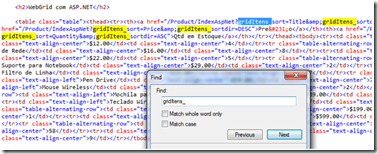
3: rowsPerPage: 15,

4: canPage: true,

5: canSort: true,

6: fieldNamePrefix: "gridItens\_");

To finish, just take a look at the HTML generated. I highlighted the fieldNamePrefix string defined on this image:

[](http://weblogs.asp.net/blogs/andrebaltieri/image_0A10B306.png)

**PageFieldName**

This property is similar to the FieldNamePrefix, but this just work for paging action. In this case the keyword “page” that represents the current grid paging will be replaced by the string you define on this parameter.

Lets take a look on the two next images to better understand how it works:

[image](http://weblogs.asp.net/blogs/andrebaltieri/image_74C65BDC.png)  
This is a sample without using the PageFieldName, and as we can see, the default parameter is “page”.

[image](http://weblogs.asp.net/blogs/andrebaltieri/image_7BB5E894.png)  
This is a sample using the PageFieldName and we can see that the parameter “Page” was replaced by “inside”.

The code to do this is here:

1: var gridItens = new WebGrid(source: Model,

2: defaultSort: "Title",

3: rowsPerPage: 5,

4: canPage: true,

5: canSort: true,

6: fieldNamePrefix: "gridItens\_",

7: pageFieldName: "inside");

**SelectionFieldName**

Similar as the other ones, this parameter allow us to customize the parameter (Passed on the query string) to refer the selected row in the WebGrid. As the effect is the same, I’ll just show the difference between using and not this option.

On this image, we have a fixed string called “row” that tell us which row is selected on the WebGrid. If for any reason you need to change this value, you can just use the SelectionFieldName property to redefine it.

[image](http://weblogs.asp.net/blogs/andrebaltieri/image_19AB3067.png)

The result will be similar to this:

[image](http://weblogs.asp.net/blogs/andrebaltieri/image_43CAB882.png)

And the code to do this is here:

1: var gridItens = new WebGrid(source: Model,

2: defaultSort: "Title",

3: rowsPerPage: 5,

4: canPage: true,

5: canSort: true,

6: fieldNamePrefix: "gridItens\_",

7: pageFieldName: "inside",

8: selectionFieldName: "selectedRow");

As the other properties (SortFieldName and SortDirectionFieldName) have the similar usage, I’ll not describe it here. But if you got how to use those above, you’ll be able to use the other ones.

**Styling your grid**

This is an important part and I have to assume that I liked the ways we have to customize the WebGrid. As I work close to my customer, the appearance of our applications should be light and rich, and I have to confess I was afraid I can’t do it with WebGrid.

I’m happy to say I was wrong =D

The WebGrid have a set of properties that allow us to customize it easily using CSS and following the WebStandards. I just think it could be better if it have an HtmlAttribute option, that we can input some attributes like “Id”, important when we are working with jQuery. But I think it will come on next versions.

So, let’s take a look on the properties to do this customization.

1: gridItens.GetHtml(

2: tableStyle: "webgrid",

3: headerStyle: "webgrid-header",

4: footerStyle: "webgrid-footer",

5: alternatingRowStyle: "webgrid-alternating-row",

6: selectedRowStyle: "webgrid-selected-row",

7: rowStyle: "webgrid-row-style")

Not that I used a set of properties (Easy to understand) to apply to each one a style. For the entire table I used the tableStyle property, for the table header the tableHeader and etc…

Now, on my CSS file (Content/Site.css) I’ll input the styles for this grid as follow:

1: .webgrid

2: {

3: width: 50%;

4: border: 0px;

5: border-collapse: collapse;

6: }

7:

8: .webgrid a

9: {

10: color: #000;

11: }

12:

13: .webgrid-header

14: {

15: padding: 6px 5px;

16: text-align: center;

17: background-color: #e8eef4;

18: border-bottom: 2px solid #3966A2;

19: height: 40px;

20:

21: border-top: 2px solid #D6E8FF;

22: border-left: 2px solid #D6E8FF;

23: border-right: 2px solid #D6E8FF;

24: }

25:

26: .webgrid-footer

27: {

28: padding: 6px 5px;

29: text-align: center;

30: background-color: #e8eef4;

31: border-top: 2px solid #3966A2;

32: height: 30px;

33:

34: border-bottom: 2px solid #D6E8FF;

35: border-left: 2px solid #D6E8FF;

36: border-right: 2px solid #D6E8FF;

37: }

38:

39: .webgrid-alternating-row

40: {

41: height: 30px;

42: background-color: #f2f2f2;

43: border-bottom: 1px solid #d2d2d2;

44:

45: border-left: 2px solid #D6E8FF;

46: border-right: 2px solid #D6E8FF;

47: }

48:

49: .webgrid-row-style

50: {

51: height: 30px;

52: border-bottom: 1px solid #d2d2d2;

53:

54: border-left: 2px solid #D6E8FF;

55: border-right: 2px solid #D6E8FF;

56: }

57:

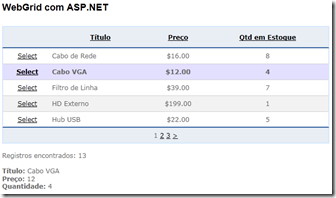
58: .webgrid-selected-row

59: {

60: font-weight: bold;

61: }

The final result is it:

[](http://weblogs.asp.net/blogs/andrebaltieri/image_51D981B2.png)

**Customizing Columns**

As you could see on the previous image, my columns are in a different language (Portuguese – Brazil), so let’s check how I did that.

On the WebGrid, we have a property called Columns that receive a list of Column. Here we will customize the grid columns.

The following code shows how to create columns on the WebGrid:

1: columns: gridItens.Columns(

2:

3: gridItens.Column(

4: columnName: "Title",

5: header: "Título",

6: style: "text-align-left"),

7:

8: gridItens.Column(

9: columnName: "Price",

10: header: "Preço",

11: style: "text-align-center",

12: format: (item) => string.Format("{0:C}", item.Price)),

13:

14: gridItens.Column(

15: columnName: "Quantity",

16: header: "Qtd em Estoque",

17: style: "text-align-center")

18: )

We have four options here and they are:

* **columnName** – Used to know which member of our model will be displayed. Price, Name, Id, etc…
* **header** – The text that will be displayed on the column header.
* **style** – CSS style of the column. In this example, you can see that for some columns I want they text aligned to center, so I applied the “text-align-center” style to them.
* **format** – Used to format the displayed string.

Just to better understand the **format** property, here is how it was displayed after use the format “{0:C}” (For currency).

[](http://weblogs.asp.net/blogs/andrebaltieri/image_07229418.png)

**Selecting a row**

Yes, we can select a row using the WebGrid and manipulate the data =D

To do this, we need to include an “special” column and use the method GetSelectItem([optional]string text) which allow us to get the row selected. If we simply call this method we will have the default text “Select”, but it can be changed by simply informing a string when call the method like GetSelectedItem(“[ + ]”) or GetSelectedItem(“Details”).

We can also use a column from our model by informing the property to be shown like this GetSelectLink(item.Title).

Here is the base code to do this and some variants.

1: columns: grid.Columns(

2: grid.Column(

3: header:"",

4: style: "text-align-center",

5: format:(item) => item.GetSelectLink(item.Title)),

6:

7: grid.Column(

8: header: "",

9: style: "text-align-center",

10: format: (item) => item.GetSelectLink("Custom Text")),

11:

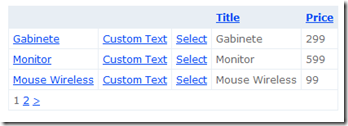
12: grid.Column(

13: header: "",

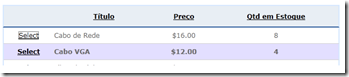
14: style: "text-align-center",

15: format: (item) => item.GetSelectLink()),

And the result is here:

[](http://weblogs.asp.net/blogs/andrebaltieri/image_19F717C2.png)

And here is a sample of a selected row:

[](http://weblogs.asp.net/blogs/andrebaltieri/image_56CB9994.png)

With the row selected, now we start a process to read its data.

Note that this is only a way to do this, and may not be the most performatively way to do this.

Here is the code to get the value from a selected row and capture the item:

1: CustomWebGrid.Models.Product product = new CustomWebGrid.Models.Product();

2: if (gridItens.HasSelection)

3: {

4: product = (CustomWebGrid.Models.Product)gridItens.Rows[gridItens.SelectedIndex].Value;

5: }

6: else

7: {

8: product = (CustomWebGrid.Models.Product)gridItens.Rows[0].Value;

9: }

Note that the grid returns an object (In this case a product), and with this information we can display the details of the product easily, or manipulate the data as you want.

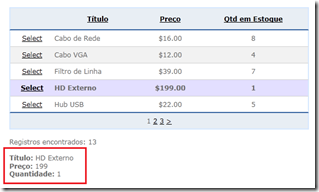
1: <b>Title:</b> < %: product.Title % ><br />

2: <b>Price:</b> < %: product.Price % ><br />

3: <b>Quantity:</b> < %: product.Quantity % ><br />

4: <br />

Here is the final result:

[](http://weblogs.asp.net/blogs/andrebaltieri/image_6CD20526.png)

Note that I displayed the amount of records found also =)   
I did it using the property “gridItens.TotalRowCount”.

It’s simple, it’s easy, it’s light, it’s ASP.NET MVC =D

**André Baltieri   
MSN:** andrebaltieri@hotmail.com | **Twitter:** @andrebaltieri   
**Blog:** <http://weblogs.asp.net/andrebaltieri>

Inside .NET Users Group Leader   
<http://www.insidedotnet.com.br/>

Published Tuesday, November 02, 2010 7:32 PM by [andrebaltieri](http://weblogs.asp.net/members/andrebaltieri.aspx)http://i1.asp.net/wan/default/images/common/star-left-on.gifhttp://i1.asp.net/wan/default/images/common/star-right-on.gifhttp://i1.asp.net/wan/default/images/common/star-left-on.gifhttp://i1.asp.net/wan/default/images/common/star-right-on.gifhttp://i1.asp.net/wan/default/images/common/star-left-on.gifhttp://i1.asp.net/wan/default/images/common/star-right-on.gifhttp://i1.asp.net/wan/default/images/common/star-left-on.gifhttp://i1.asp.net/wan/default/images/common/star-right-on.gifhttp://i1.asp.net/wan/default/images/common/star-left-on.gifhttp://i1.asp.net/wan/default/images/common/star-right-on.gif

Filed under: [ASP.NET MVC 3](http://weblogs.asp.net/andrebaltieri/archive/tags/ASP.NET+MVC+3/default.aspx), [ASP.NET MVC](http://weblogs.asp.net/andrebaltieri/archive/tags/ASP.NET+MVC/default.aspx)