Grid.MVC Documantation

Installation

You can easy install the component to you ASP.NET MVC 3/4 project by installing nuget package. In the package manager console (View-Other windows - package manager console) type the following command:

install-package Grid.Mvc

After that installer makes the following changes to your project:

1. Reference GridMvc.dll - base implementation of Grid.Mvc
2. Adds Views/Shared/Grid.cshtml, Views/Shared/GridPager.cshtml - views for grid
3. Adds Content/gridmvc.css - default style sheet for the grid
4. Adds Scripts/gridmvc.js, Scripts/gridmvc.min.js - Grid.Mvc scripts.

Also you can download binaries and make this changes manually.

Quick start with Grid.Mvc

To start displaying Grid you need to retrieve the collection of Model items in your project, for example:

Your model class:

```
public class Foo
{
        public string Title {get; set;}
        public string Description {get;set;}
}
```

Now, imagine that you have a controller action which retrieve a strongly-typed collection of Model property and pass it to the View:

```
public ActionResult Index()
{
       var items = fooRepository.GetAll();
       return View(items);
}
```

Now you need to render your items collection in the View.

To render GridMvc - you can use Html helper extension.

First, reference GridMvc.Html namespace in the view:

```
@using GridMvc.Html
```

Render Grid.Mvc on the page:

```
@using GridMvc.Html

@Html.Grid(Model).Columns(columns =>
           {
```

```
                columns.Add(foo => foo.Title).Titled("Custom column
title").SetWidth(110);
                columns.Add(foo => foo.Description).Sortable(true);
           }).WithPaging(20)
```

By default, this html helper method render "_Grid.cshtml" partial view in your Views/Shared folder. If you want to render other view - specify his name in 'viewName' parameter, like this:

```
@using GridMvc.Html

@Html.Grid(Model,"_MyCustomGrid").Columns(columns =>
          {
                columns.Add(foo => foo.Title).Titled("Custom column
title").SetWidth(110);
                columns.Add(foo => foo.Description).Sortable(true);
          }).WithPaging(20)
```

For more documentation about column options, please see: Custom columns.

In the last step you need to ensure, that Grid.Mvc stylesheet and scripts registred in your page. Grid.Mvc scripts uses jQuery to provide client side functionality. You need register them too.

In your _Layout.cshtml file:

```
        <script src="@Url.Content("~/Scripts/jquery.min.js")"
type="text/javascript"> </script>
        <link href="@Url.Content("~/Content/Gridmvc.css")" rel="stylesheet"
type="text/css" />
        <script src="@Url.Content("~/Scripts/gridmvc.min.js")"
type="text/javascript"> </script>
```

If your project has own table styles you can use them. By default Grid.Mvc builds layout compatible with Bootstrap 3 css framework - http://getbootstrap.com/

Paging

How it works

You need to pass collection of the items, that should be displayed in the grid. After that grid will "cut" only displaying page.

If your data source is database (for example) you need to pass IQueryable<T> collection to the grid. Grid.Mvc uses IQueryable<T> interface to construct query expressions to your data collection.

When you go to some page the grid invokes .Skip(N).Take(N) methods and when you sort data the grid invokes OrderBy or OrderByDescending methods etc.

If you pass IQueryable collection produced by ORM (like Entity Framework) - the ORM will translate query to database (SQL queries). In this case Grid.Mvc queries only one page of data, that currently displayed.

How to configure

To enable paging of you Grid can call "WithPaging" method in your View and specify page size and other parameters:

```
@Html.Grid(Model).Columns(columns =>
                    {
                        columns.Add(foo => foo.Title).Titled("Custom column
title").SetWidth(110);
                        columns.Add(foo => foo.Description).Sortable(true);
                    }).WithPaging(20)
```

Also if you derive you own Grid - to enable paging set 'EnablePaging' property to 'true'. Use 'Pager' object to configure paging options in your Grid:

```
public class FooGrid : Grid<Foo>
{
    public FooGrid(IQueryable<TestModel> items) : base(items)
    {
              Columns.Add(foo => foo.Title).Sortable(true);
        Columns.Add(foo => foo.Description).Sortable(true);

        EnablePaging = true;
        Pager.PageSize = 10;
    }
}
```

Pager options

Property name	Property type	Description
PageSize	Int32	Setup the page size for displaying items in the grid

Custom columns

You can define your custom column by calling method Columns.Add in your Grid class. For example:

```
Columns.Add(o => o.Customers.CompanyName)
                .Titled("Company Name")
                .Sortable(true)
                .SetWidth(220);
```

Method Titled int this case setup the column header text, if you do not setup this, Grid take the name of property (CompanyName).

Also you can call 'Insert' method if you want to add column at specified position:

```
Columns.Insert(0, o => o.Customers.CompanyName)
                .Titled("Company Name")
```

You can construct the displaying value:

```
Columns.Add(o => o.Employees.LastName)
                .Titled("Employee")
                .RenderValueAs(o => o.Employees.FirstName + " " +
o.Employees.LastName)
```

Not connected columns

Sometimes you need to add column, that renders some content, but don't have a base model property (for sorting, filtering). In this case you need to use empty contructor of Add method to create column:

```
Columns.Add().RenderValueAs(model => "Item " + model.Id);
```

When you create not connected column - you must specify RenderValueAs method to specify which content will be render in this column. Also sorting and filtering will not works in this columns.

Hidden columns

By default all columns adding to the grid are visible. If you want, that column not displaying in the grid view - specify that in the Add method:

```
Columns.Add(o => o.Id, true);
```

In this case you will not see this column, but you can get values at the client side (javascript). Note: you can't sort hidden columns.

Sorting

If you want to enable sorting of your column - just call Sortable(true) method of added column:

```
Columns.Add(o => o.Employees.LastName)
                .Titled("Employee")
                .Sortable(true);
```

Sorting will be implemented by field that you specify at the Add method (o.Employees.LastName).

If you pass the ordered collection of items in the Grid constructor and want to display this by default - you want to specify the initial sorting options:

```
Columns.Add(o => o.OrderDate)
                .Titled("Date")
                .Sortable(true)
                .SortInitialDirection(GridSortDirection.Descending);
```

Auto generating columns

This feature of Grid.Mvc provides functionality to automatically creates columns from public properties of your model class.

To do this you need to call AutoGenerateColumns method of Grid<T> class or Grid html helper. After that Grid.Mvc adds columns for each public property:

```
@Html.Grid(Model).AutoGenerateColumns()
```

If you want to exclude some properties from auto generation or customize his properties, you need to use Data annotations (please see Data annotations)

Column settings

Property name	Description	Example
Titled	Setup the column header text	Columns.Add(x=>x.Name).Titled("Name of product");
Encoded	Enable or disable encoding column values	Columns.Add(x=>x.Name).Encoded(false);
Sanitized	If encoding is disabled sanitize column value from XSS attacks	Columns.Add(x=>x.Name).Encoded(false).Sanitize(false);
SetWidth	Setup width of current column	Columns.Add(x=>x.Name).SetWidth("30%");
RenderValueAs	Setup delegate to render column values	Columns.Add(x=>x.Name).RenderValueAs(o => o.Employees.FirstName " " o.Employees.LastName)
Sortable	Enable or disable sorting for current column	Columns.Add(x=>x.Name).Sortable(true);
SortInitialDirection	Setup the initial sort deirection of the column (need to enable sorting)	Columns.Add(x=>x.Name).Sortable(true).SortInitialDirection(GridSortDirection.Descending);
SetInitialFilter	Setup the initial filter of the column	Columns.Add(x=>x.Name).Filterable(true).SetInitialFilter(GridFilterType.Equals, "some name");
ThenSortBy	Setup ThenBy sorting of current column	Columns.Add(x=>x.Name).Sortable(true).ThenSortBy(x=>x.Date);
ThenSortByDescending	Setup ThenByDescending sorting of current column	Columns.Add(x=>x.Name).Sortable(true).ThenSortBy(x=>x.Date).ThenSortByDescending(x=>x.Description);

Filterable	Enable or disable filtering feauture on the column	Columns.Add(x=>x.Name).Filterable(true);
SetFilterWidgetType	Setup filter widget type for rendering custom filtering user interface	Columns.Add(x=>x.Name).Filterable(true).SetFilterWidgetType("MyFilter");
Format	Format column value	Columns.Add(o => o.OrderDate).Format("{0:dd/MM/yyyy}")
Css	Apply css classes to the column	Columns.Add(x => x.Number).Css("hidden-xs")

Data annotations

You can custome Grid and columns setting, using data annotations. In other words you can mark properties of you model class as grid columns, specify column options and call "AutoGenerateColumns" method. Grid.Mvc automatically create columns as you describe in your annotations.

There are some attributes for this:

1. **GridTableAttribute** - applies for model class and specify options for Grid (paging options...);
2. **GridColumnAttribute** - applies for model public property and tell that this property is grid column with set of properties;
3. **GridHiddenColumn** - applies for model public property and tell that this property is grid hidden column;
4. **NotMappedColumnAttribute** - applies for model public property and tell that this property is NOT a column. If property has this attribute - Grid.Mvc will not Add this column to column collection;

For example model class with data annotations:

```
[GridTable(PagingEnabled = true, PageSize = 20)]
public class Foo
{
    [GridColumn(Title = "Name title", SortEnabled = true, FilterEnabled = true)]
    public string Name { get; set; }

    [GridColumn(Title = "Active Foo?")]
    public bool Enabled { get; set; }

    [GridColumn(Title = "Date", Format = "{0:dd/MM/yyyy}")]
    public DateTime FooDate { get; set; }

    [NotMappedColumn]
    public byte[] Data { get; set; }
}
```

This class describe that grid table must contains 3 columns (Name, Enabled and FooDate) with custom options.

Than you can render this table in the View:

```
@Html.Grid(Model).AutoGenerateColumns()
```

Grid.Mvc generate columns based on your data annotations when AutoGenerateColumns method is invoked. It means that you can add custom columns after or before this method, for example:

```
@Html.Grid(Model).AutoGenerateColumns().Columns(columns=>columns.Add(foo=>foo.Chil
d.Price))
```

Also you can overwrite grid options using WithPaging method.

Client side

Client side object model allows you to retrieve column values of selected by user row in the grid.

After installing Grid.Mvc package (see: Installation) Scripts/common.mvc.ui.grid.js was added to your project. If you want to use client object model you need to attach jQuery (http://jquery.com) and gridmvc.min.js, like this:

```
<script src="@Url.Content("~/Scripts/jquery.min.js")"
type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/gridmvc.min.js")"
type="text/javascript"></script>
```

When you rendering your grid you need to specify grid client name, using Named method:

```
      @Html.Grid(Model).Named("ordersGrid").Columns(columns =>
                    {
                        columns.Add(foo => foo.Title).Titled("Custom column
title");
                        columns.Add(foo => foo.Description).Sortable(true);
                    })
```

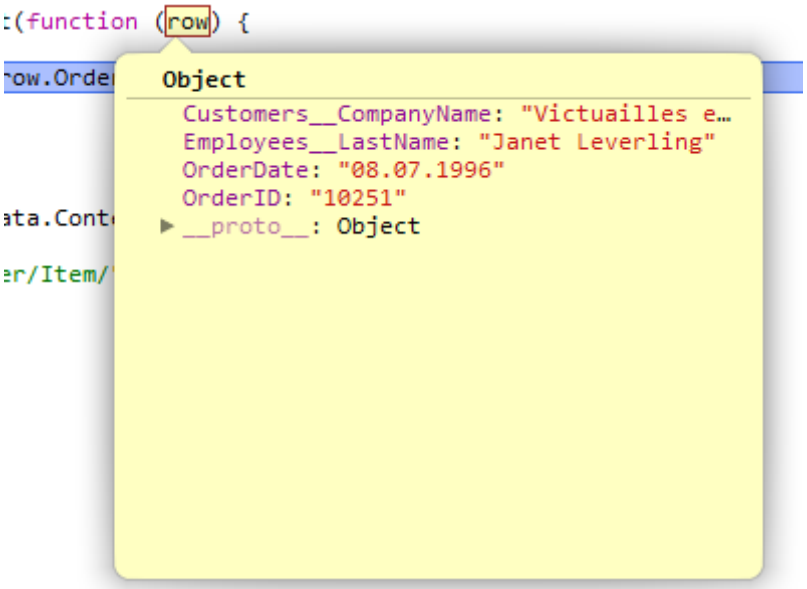
Grid.mvc.js - provides filtering functionality on the client side and row selecting.
The script register singleton object named 'pageGrids' that contain all grids registred on the page.

To retrive user selected row values, register script:

```
<script>
    $(function () {
        pageGrids.ordersGrid.onRowSelect(function (e) {
            alert(e.row.OrderID);
        });
    });
</script>
```

Method onRowSelect register event when user select Grid row.

'e.row' parameter contains all column values of selected row. You can retrive them by calling his properties (names is the same with your model in C#):



Filtering

Since Grid.Mvc 2.0 you can enable filtering option for your columns. To enable this functionality use Filterable method:

```
        Columns.Add(o => o.Customers.CompanyName)
                  .Titled("Company Name")
                  .Filterable(true)
                  .Width(220);
```

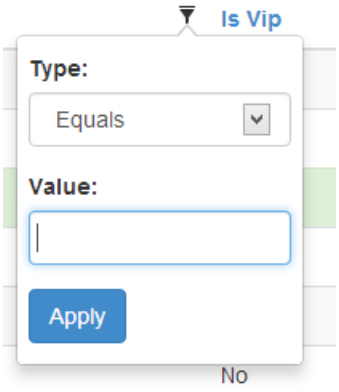
After that you can filter this column.
Grid.Mvc support several types of columns (specified in Add method):

1. System.String
2. System.Int32
3. System.Int64
4. System.Boolean
5. System.DateTime
6. System.Decimal
7. System.Byte
8. System.Double
9. System.Single

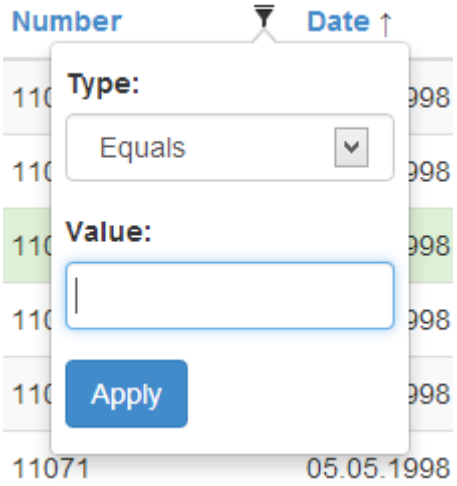
Also supports nullable types of this list.

To build more user friendly interface Grid.Mvc has different filter widget for this types.

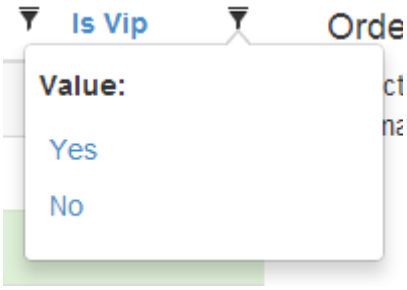
- **TextFitlerWidget** - provides filter interface for text columns (System.String). This means that if your column has text data - Grid.Mvc render specific filter interface:



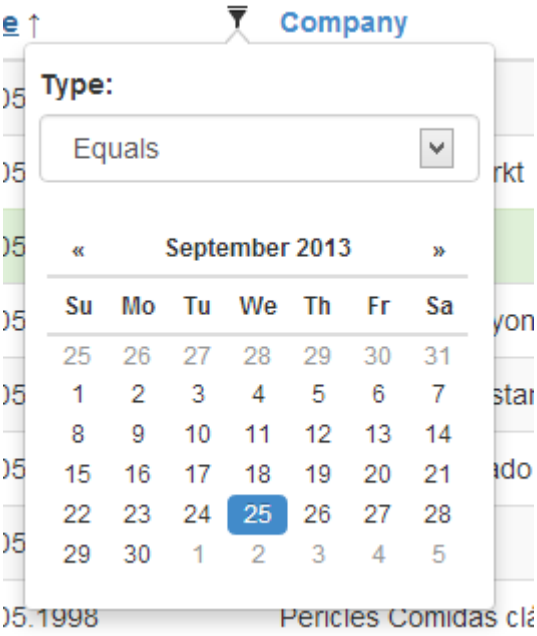
- **NumberFilterWidget** - provides filter interface for number columns (System.Int32, System.Decimal etc.)



- BooleanFilterWidget - provides filter interface for boolean columns (System.Boolean):



- DateTimeFilterWidget - provides filter interface for datetime columns (System.DateTime). To render DatePicker control you need setup Grid.Mvc datepicker (https://www.nuget.org/packages/Grid.Mvc.DatePicker/) on the page.



Also you can create your own filter widget.

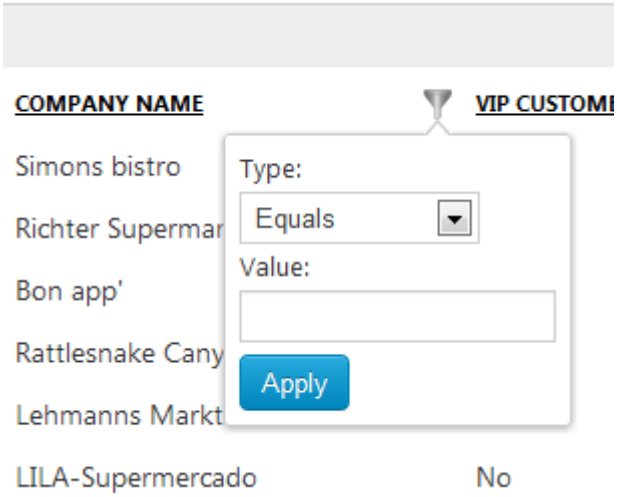
Creating custom filter widget

If you need to create custom user interface to filter specific grid columns - you can do it with Grid.Mvc. To do this you need:

1. Create specific javascript object, that will render your interface;
2. Setup custom filter widget name with 'SetFilterWidgetType' function;
3. Register custom filter on the client side

Introduce

Image that you display the column with customer names. By default Grid.Mvc render filter interface for text column and she will looks like:

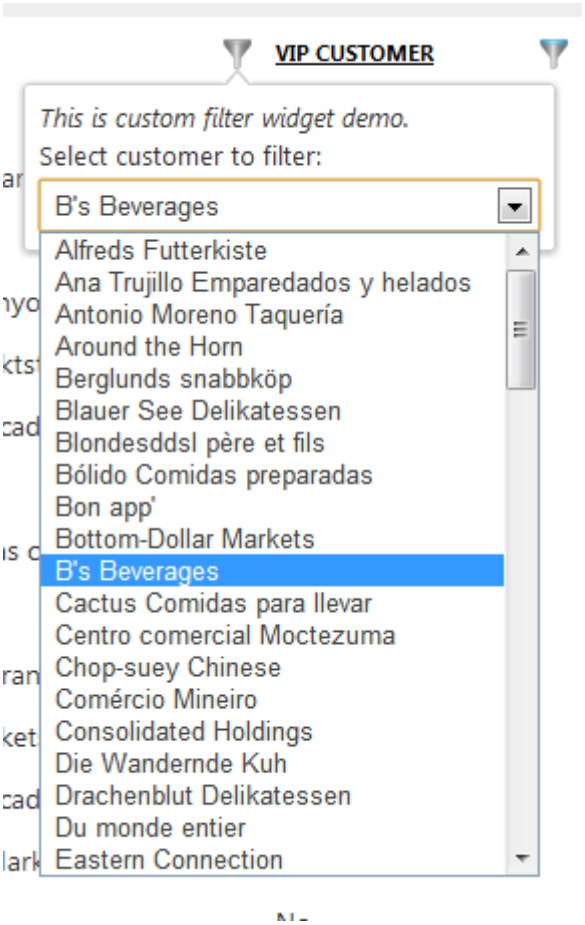


By you want to provide more specific filter interface for this column. For example, you want that user picks customer from customer's list, like:

Creating javascript widget object

Your javascript object should have the following functions:

1. **getAssociatedTypes** - returns the array of filter types, that associated with current widget;
2. **showClearFilterButton** - returns true/false and specify whether render 'Clear filter' button for this widget;
3. **onRender** - this function invokes by Grid.Mvc and tell that you need to render your widget, This method invokes once, when user first time click on filter button;



4. **onShow** - this function invokes by Grid.Mvc and tell that you widget was shown to user (open popup window);

   Below is the example of filter widget (you can find it in the sample web app in the source):

```
/***
* CustomersFilterWidget - Provides filter user interface for customer name column
in this project
* This widget onRenders select list with avaliable customers.
*/

function CustomersFilterWidget() {
    /***
    * This method must return type of registered widget type in
'SetFilterWidgetType' method
    */
    this.getAssociatedTypes = function () {
        return ["CustomCompanyNameFilterWidget"];
    };
```

```
    /***
    * This method invokes when filter widget was shown on the page
    */
    this.onShow = function () {
        /* Place your on show logic here */
    };

    this.showClearFilterButton = function () {
        return true;
    };
    /***
    * This method will invoke when user was clicked on filter button.
    * container - html element, which must contain widget layout;
    * lang - current language settings;
    * typeName - current column type (if widget assign to multipile types, see:
getAssociatedTypes);
    * values - current filter values. Array of objects [{filterValue: '',
filterType:'1'}];
    * cb - callback function that must invoked when user want to filter this
column. Widget must pass filter type and filter value.
    * data - widget data passed from the server
    */
    this.onRender = function (container, lang, typeName, values, cb, data) {
        //store parameters:
        this.cb = cb;
        this.container = container;
        this.lang = lang;

        //this filterwidget demo supports only 1 filter value for column column
        this.value = values.length > 0 ? values[0] : { filterType: 1, filterValue:
"" };

        this.renderWidget(); //onRender filter widget
        this.loadCustomers(); //load customer's list from the server
        this.registerEvents(); //handle events
    };
    this.renderWidget = function () {
```

```
        var html = '<p><i>This is custom filter widget demo.</i></p>\
                    <p>Select customer to filter:</p>\
                    <select style="width:250px;" class="grid-filter-type
customerslist form-control">\
                    </select>';
        this.container.append(html);
    };
    /***
    * Method loads all customers from the server via Ajax:
    */
    this.loadCustomers = function () {
        var $this = this;
        $.post("/Home/GetCustomersNames", function (data) {
            $this.fillCustomers(data.Items);
        });
    };
    /***
    * Method fill customers select list by data
    */
    this.fillCustomers = function (items) {
        var customerList = this.container.find(".customerslist");
        for (var i = 0; i < items.length; i++) {
            customerList.append('<option ' + (items[i] == this.value.filterValue ?
'selected="selected"' : '') + ' value="' + items[i] + '">' + items[i] +
'</option>');
        }
    };
    /***
    * Internal method that register event handlers for 'apply' button.
    */
    this.registerEvents = function () {
        //get list with customers
        var customerList = this.container.find(".customerslist");
        //save current context:
        var $context = this;
        //register onclick event handler
        customerList.change(function () {
```

```
            //invoke callback with selected filter values:
            var values = [{ filterValue: $(this).val(), filterType: 1 /* Equals */
}];
            $context.cb(values);
        });
    };

}
```

This example loads customer's list from the server via ajax, build and render layout in 'onRender' method (when users first time clicking on filter button).

Setup custom filter type

Filter type - is a string, that specify, which filter widget should render on the column. By default he's setup to field .Net type. If you want to override this type you need to use SetFilterWidgetType function:

```
columns.Add(o => o.Customers.CompanyName).Titled("Company Name")
                                    .Sortable(true)
                                    .Filterable(true)

.SetFilterWidgetType("CustomCompanyNameFilterWidget");
```

Filter type must contains in getAssociatedTypes array of your JS object in the step 1.

Register filter widget

In the last step you need to add custom widget in the Grid.Mvc filters collection. You can do this by using 'addFilterWidget'. Adds the following script after gridmvc.min.js:

```
<script>
        pageGrids.ordersGrid.addFilterWidget(new CustomersFilterWidget());
</script>
```

Where CustomersFilterWidget - object defined in step 1.

Setup initial column filtering

Sometime you need to setup initial column filtering after page loads first time. After that user can work this pre-filtered grid or clear/change filter settings.

You can do this, using SetInitialFilter method:

```
columns.Add(o => o.Customer.CompanyName)
        .Titled("Company Name")
        .ThenSortByDescending(o => o.OrderID)
        .SetInitialFilter(GridFilterType.StartsWith, "a")
```

After that you need to tell Grid.Mvc to apply initial settings by passing **grid-init** parameter to query string:

http://<url>/**?grid-init=1**

Overview the problem

Grid.Mvc uses query string to pass filtering, sorting and paging settings to the server side.
If you place multiple grids on the page - that query string parameters will be applied to all grids on the page. There are ways to resolve this conflicts.

Columns

If your page grids have the same column names, that filtering and sorting settings will be applied to both grids. In this case you need to give grid columns unique internal names. You can do that using overloaded Columns.Add method:

```
columns.Add(o => o.Number, "CustomNumberName")
                .Titled("Number")
                .Sortable(true)
                .SetWidth(220);
```

In this case grid.mvc will be pass new internal column name to query string like:

?grid-column=**CustomNumberName**&grid-dir=0 - sorting
?grid-filter=**CustomNumberName**1Blauer%20See%20Delikatessen - filtering

This settings will applied only to added column.

Paging

To resolve paging setting conflict you need to specify custom grid pager query parameter name:

```
@Html.Grid(Model).Columns(columns =>
{
    ...
})..WithPaging(15, 6,"grid1-page")
```

After that grid.mvc will be pass new query string parameter:

/?**grid1-page**=2

Client side

If you want to use Client side (javascript) api that you need to give each grid unique client side ID.

```
@Html.Grid(Model).Named("myGrid").Columns(columns =>
{
...
})
```

After that you can access to added grid throght **pageGrids** object:

```
pageGrids.myGrid.addFilterWidget(new CustomersFilterWidget());
```

Render button, checkbox etc in the grid cell

To render custom html markup in the grid cell you should use **RenderValueAs** method. Also you need to disable default encoding and satinizing cell values, using **Encoded** and **Sanitized** method.

Button

```
columns.Add()
        .Encoded(false)
        .Sanitized(false)
        .SetWidth(30)
        .RenderValueAs(o => @<button type="submit">Submit</button>);
```

Checkbox

```
columns.Add()
        .Encoded(false)
        .Sanitized(false)
        .SetWidth(30)
        .RenderValueAs(o => Html.CheckBox("checked", false));
```

Custom layout

You can render any custom layout, using razor @helper:

```
@helper CustomRenderingOfColumn(Order order)
{
    if (order.Customer.IsVip)
    {
    <text>Yes</text>
    }
    else
    {
    <text>No</text>
    }
}
```

```
@Html.Grid(Model)..Columns(columns =>
{
        columns.Add(o => o.Customer.IsVip)
                .Titled("Vip customer")
                .SetWidth(150)
                .RenderValueAs(o => CustomRenderingOfColumn(o));
})
```

Overview

As you now Grid.Mvc uses **IQueryable<T>** interface to construct query expressions to your data collection.
When you go to some page the grid invokes **.Skip(N).Take(N)** methods and when you sort data the grid invokes **OrderBy** or **OrderByDescending** methods etc.
If you pass IQueryable collection produced by ORM (like Entity Framework), in this case the ORM will generate query to database. In this case Grid.Mvc queries only one page of data, that currently displayed.
If you want to see how it works, you can download the source code and learn it.

The problem

This mechanism works fine with large amount of data in the table, because Grid.Mvc loads items, that need to be displayed. But in real projects may exist there following moment:
The model item, displayed in the grid is different from data, stored in the table, and some properties need to be instantiated by other services.
I try to explain on the following example: we have some **FooDto** items, stored in the database, and querying it, using ORM:

```
public class FooDto
{
    public int Id { get; set; }
    public string Title { get; set; }
}
```

And structure, that we want to display in the grid:

```
public class FooGridItem
{
    public int Id { get; set; }
    public string Title { get; set; }
    public bool CanEdit { get; set; }
}
```

Property **CanEdit** tells that current user can edit this foo item, and we need to show edit button. This property should be initialized from IPermissionManager class:

```
    public interface IPermissionManager
    {
        bool DoesCurrentUserCanEditFoo(int id);
    }
```

Controller:

```
        public ActionResult List()
        {
            var items = _fooRepository.GetAll() /*returns IQueryable<T>*/
                                      .Select(fooDto => new FooGridItem
                                          {
                                              Id = fooDto.Id,
                                              Title = fooDto.Title
                                          });
            return View(items);
        }
```

View:

```
@Html.Grid(Model).Columns(columns =>
      {
          columns.Add(f => f.CanEdit)
              .RenderValueAs(f => f.CanEdit ? Html.ActionLink("Edit", "Edit") :
MvcHtmlString.Create(string.Empty));

          columns.Add(f => f.Title);
      }).Sortable().WithPaging(20)
```

But how we can initialize **CanEdit** property? We don't know which items will be displayed on the page, because the source collection can be filtered, sorted and paged by Grid.Mvc infrastructure.
You can do like this:

```
            //_permissionManager injected via ctor DI
     foreach (var fooGridItem in items) //Query all from DB!!!
            {
                fooGridItem.CanEdit =
_permissionManager.DoesCurrentUserCanEditFoo(fooGridItem.Id);
            }
```

But, in this case you query all items from database. If you have many items in the Foo table it can produce large latency lag. Also method **DoesCurrentUserCanEditFoo** may take large amount of time. Ok, we need to initialize this property only for displayed items.

Solution

There are several ways to solve this problem. But the best way is to override**GetItemsToDisplay** method of **Grid<T>** class (option available since Grid.Mvc 2.3.0). This method returns items, that need to be displayed on the page.
At first create class that derived from **Grid<T>**:

```
    public class FooGrid : Grid<FooGridItem>
    {
        private readonly IPermissionManager _permissionManager;
        private IEnumerable<FooGridItem> _displayingItems;

        public FooGrid(IQueryable<FooGridItem> items, IPermissionManager
permissionManager)
            : base(items)
        {
            _permissionManager = permissionManager;
        }

        /// <summary>
        /// Grid.Mvc invokes this method when wants to retrive items, which to be
displayed
        /// </summary>
        protected override IEnumerable<FooGridItem> GetItemsToDisplay()
```

```
        {
            // Grid.Mvc may call this methods multiple times. You may cache
process result in the field;
            if (_displayingItems != null)
                return _displayingItems;

            // Get items that Grid.Mvc wants to display
            _displayingItems = base.GetItemsToDisplay().ToList();

            // Initialize fields only for displaying items via service reference:
            foreach (var displayedItem in _displayingItems)
            {
                displayedItem.CanEdit =
_permissionManager.DoesCurrentUserCanEditFoo(displayedItem.Id);
            }

            return _displayingItems;
        }
    }
```

Now you can create the FooGrid instance and render it the view, the same way as at the top of this article:

```
        public ActionResult List()
        {
            var items = _fooRepository.GetAll() /*returns IQueryable<T>*/
                                      .Select(fooDto => new FooGridItem
                                          {
                                              Id = fooDto.Id,
                                              Title = fooDto.Title
                                          });
            return View(new FooGrid(items, _permissionManager));
        }
```

Migrate From Grid.Mvc 2.x to 3.x

There was some changes, that break backward capability during migration from 2.x to 3.x grid version. To update your version - just install new grid package (replace all package files with the new one). If you made any changes with grid template files - it will be lost, I recomend to save this changes, before updating.

Layout

Grid.mvc builds other layout (compatible with bootstrap 3) and if you has any custom styles for grid - they may not apply for Grid.Mvc 3. In this case - change your style rules (selectors etc)

Custom filter widgets

Grid.Mvc 3 support multiple filter values per one column, also filter widget need to return array of filter values. Please see updated article Creating custom filter widget

Client side

Now onRowSelect method pass Event argument intead of row. Use Event.row property to get row object:

```
pageGrids.ordersGrid.onRowSelect(function (e) {
console.log(e.row);
});
```

Custom renderers

Grid renderers interface are splited to cell renderers and header renderers. If you have custom header renderers use **IGridColumnHeaderRenderer**