Articles » Third Party Products » Third Party Products and Tools » General

# An Example to Use Kendo Grid

Dr. Song Li, 1 Nov 2013 CPOL
★★★★★ 4.80 (16 votes)

This article presents an example to use Kendo grid.

## Editorial Note

This article appears in the Third Party Products and Tools section. Articles in this section are for the members only and must not be used to promote or advertise products in any way, shape or form. Please report any spam or advertising.

**Download KendoGridExample.zip**

## Introduction

This article presents an example to use Kendo grid. Kendo provided us many web tools, such as "DatePicker", "NumericTextBox", and the grid. Among these web tools, the grid is the most involved web tool. This article is to show you how to use the Kendo grid. The example here uses the open source version Kendo. If you have already paid for a paid version and happy about what it offers you, this article is not for you, because your version is much easier to use. But if you are happy about working on the open source version, you can take a look at this article and find out how Kendo works in some more detail, and possibly with a little more coding.

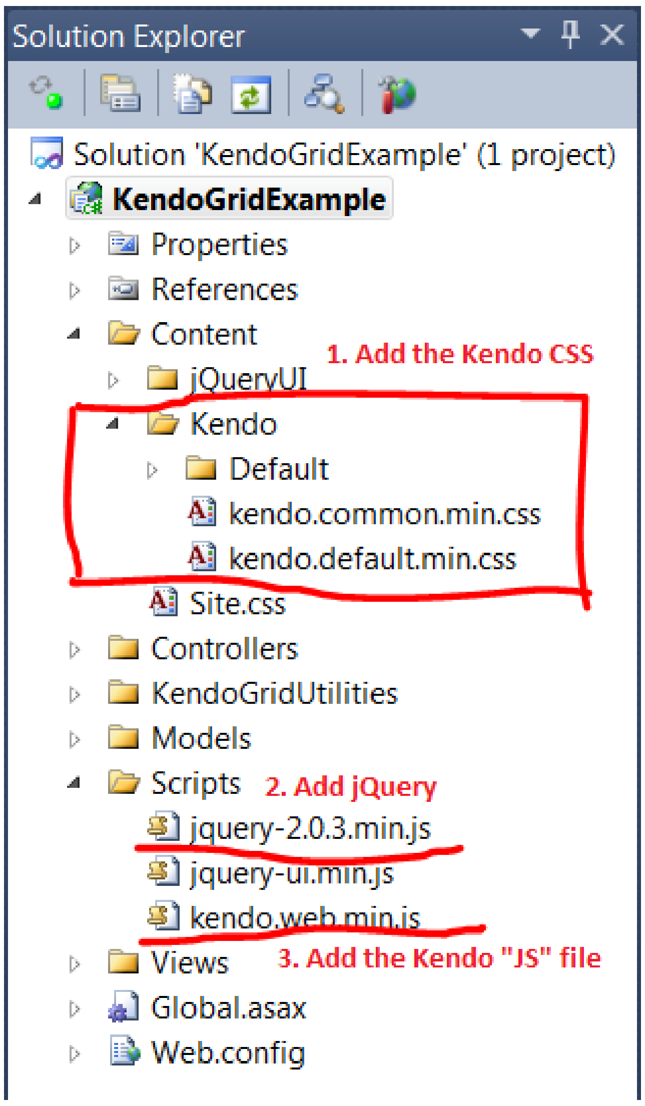
## Background

One of the most commonly used client side web controls is the "grid". There are many JavaScript based grid options. Among these options, depending on the features that you need, the Kendo grid is not necessarily the best. But it is fairly easy to use and fits most of the application scenarios. This article presents an example to use Kendo grid. The example is an MVC 2 application written in Visual Studio 2010. I am well aware that the most recent version of MVC is 4, but to keep the example in a lower version makes the readers easier to download the example and run it, in case they do not have the higher version of Visual Studio and the environment. In this example, I will show you the following:

- How to setup the Kendo Environment in your web application;
- How to implement a basic Keno grid;
- How to achieve sorting and filtering on the grid;
- How to create a basic event listener to handle the events such as double clicking on a grid row.

The example uses server binding, which I feel to be easier to manage when you have a relatively large and dynamic data set.

## Setup Kendo Environment in the Application

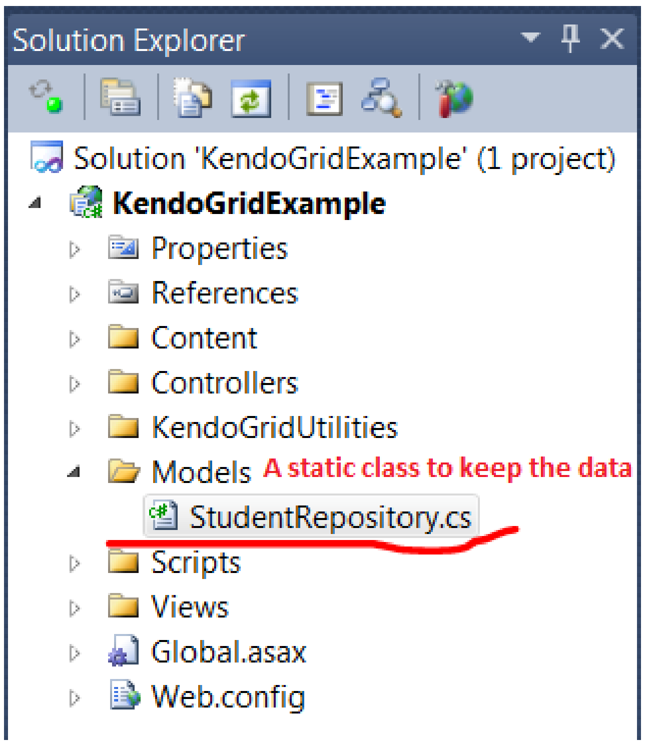


There are a couple of Kendo versions. The one used in this example is the open source version.

- You will need the "kendo.common.min.css" file. Beside this file, Kendo also provided us many options to control the look of the Kendo tools. In this example, I chose the default look, so I also included the "kendo.default.min.css" file and the image files associated to this css file in the "Default" folder.
- Kendo is built upon jQuery, so we will need the jQuery file "jquery-2.0.3.min.js".
- You will also need the "kendo.web.min.js" coming from Kendo in your project.

The version of Kendo used in this example is the open source version with version number "2013.2.716".

## The Data to Display in the Grid

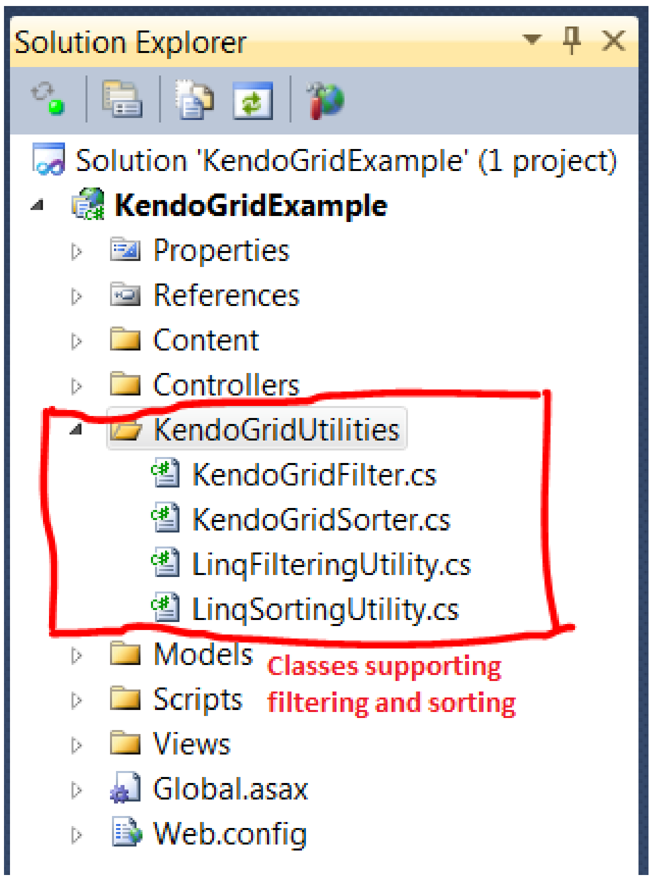


The "StudentRepository.cs" serves as the data repository for the data to be displayed in the Kendo grid.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace KendoGridExample.Models
{
    public class Student
    {
        public int? Id { get; set; }
        public string LastName { get; set; }
        public string FirstName { get; set; }
        public int Score { get; set; }
        public bool Active { get; set; }
    }

    public static class StudentRepository
    {
        public static List<Student> Students { get; private set; }
        static StudentRepository()
        {
            Func<char, string> rep4 = x =>
            {
                var result = new StringBuilder();
                for (var i = 0; i < 5; i++)
                {
                    result.Append(x);
                }
                return result.ToString();
            };

            Students = new List<Student>();
            var rand = new Random();
            for(var i = 1; i <= 1000; i++)
            {
                var m = (i - 1)%26;
                var lascii = m + 65;
                var student = new Student()
                {
                    Id = i,
                    LastName = rep4((char)lascii),
                    FirstName = " First Name " + i % 20,
                    Score = 60 + (int) Math.Round(rand.NextDouble()*40),
                    Active = i % 3 == 0
                };

                Students.Add(student);
            }
        }
    }
}
```

The static class "StudentRepository" will be initiated with 1000 randomly generated students. I will show you how to display these students in a Kendo grid in this example.

## The Classes Support Sorting and Filtering



The "KendoGridSorter.cs" and "LinqSortingUtility.cs" files support the sorting feature of the Kendo grid.

```
using System;
using System.Collections.Generic;
using System.Web;

namespace KendoGridExample.KendoGridUtilities
{
    public class KendoGridSorterCollection
    {
        public List<KendoGridSorter> Sorters { get; private set; }
        private KendoGridSorterCollection()
        {
            Sorters = new List<KendoGridSorter>();
        }

        public static KendoGridSorterCollection BuildEmptyCollection()
        {
            return new KendoGridSorterCollection();
        }

        public static KendoGridSorterCollection BuildCollection(HttpRequestBase request)
        {
            var collection = BuildEmptyCollection();

            var index = 0;
            while(true)
            {
                var sorter = new KendoGridSorter()
                {
                    Field = request.Params["sort[" + idex + "][field]"],
                    Direction = request.Params["sort[" + idex + "][dir]"]
                };

                if (sorter.Field == null) { break; }
                collection.Sorters.Add(sorter);
                idex++;
            }

            return collection;
        }
    }

    public class KendoGridSorter
    {
        public string Field { get; set; }
        public string Direction { get; set; }
    }
}
using System;
using System.Linq;

using System.Collections.Generic;

namespace KendoGridExample.KendoGridUtilities
{
    public static class LinqSortingUtility
    {
        public static IEnumerable<T> MultipleSort<T>(this IEnumerable<T> data,
          List<KendoGridSorter> sortExpressions)
        {
            if ((sortExpressions == null) || (sortExpressions.Count <= 0))
            {
                return data;
            }

            IEnumerable<T> query = from item in data select item;
            IOrderedEnumerable<T> orderedQuery = null;

            for (int i = 0; i < sortExpressions.Count; i++)
            {
                var index = i;
                Func<T, object> expression = item => item.GetType()
                                .GetProperty(sortExpressions[index].Field)
                                .GetValue(item, null);

                if (sortExpressions[index].Direction == "asc")
                {
                    orderedQuery = (index == 0)
                        ? query.OrderBy(expression)
                            : orderedQuery.ThenBy(expression);
                }
                else
                {
                    orderedQuery = (index == 0)
                        ? query.OrderByDescending(expression)
                            : orderedQuery.ThenByDescending(expression);
                }
            }

            query = orderedQuery;

            return query;
        }
    }
}
```

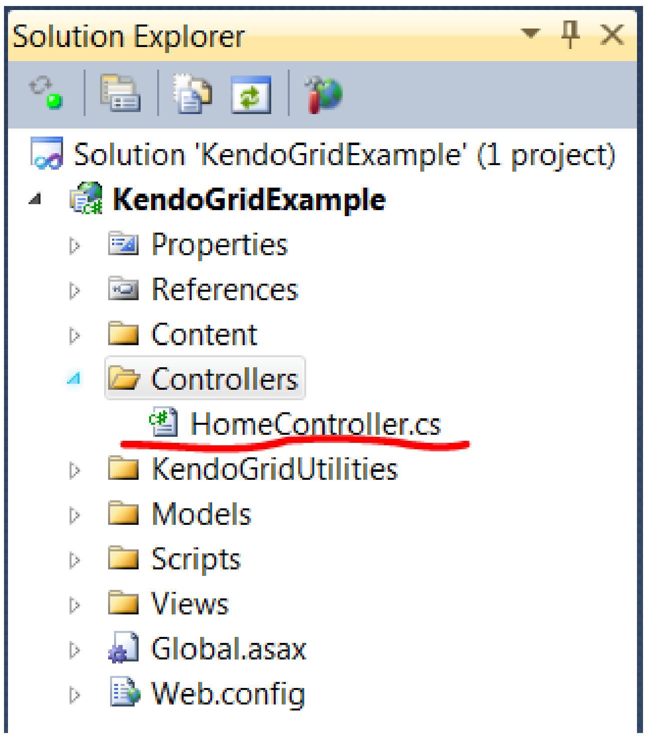
- The factory method "BuildCollection" in the "KendoGridSorterCollection" class builds an instance of the "KendoGridSorterCollection" class based on the "HttpRequestBase" object sent from the Kendo grid. When it is successful, the "KendoGridSorterCollection" object will have a list of the "KendoGridSorter" objects, which tell us how the users want to sort the grid data.
- The "MultipleSort" is an extension method on an "IEnumerable" collection. We can call this method by passing the list of the "KendoGridSorter" objects to sort the data.
- These two classes support sorting operations on multiple columns.

The "KendoGridFilter.cs" and "LinqFilteringUtility.cs" files support the filtering feature of the Kendo grid.

```
using System.Collections.Generic;
using System.Web;

namespace KendoGridExample.KendoGridUtilities
{
    public class KendoGridFilterCollection
    {
        public List<KendoGridFilter> Filters { get; private set; }
        private KendoGridFilterCollection()
        {
            Filters = new List<KendoGridFilter>();
        }

        public static KendoGridFilterCollection BuildEmptyCollection()
        {
            return new KendoGridFilterCollection();
        }

        public static KendoGridFilterCollection BuildCollection(HttpRequestBase request)
        {
            var collection = BuildEmptyCollection();

            var idex = 0;
            while (true)
            {
                var filter = new KendoGridFilter()
                {
                    Field = request.Params["filter[filters][" + idex + "][field]"],
                    Operator = request.Params["filter[filters][" + idex + "][operator]"],
                    Value = request.Params["filter[filters][" + idex + "][value]"]
                };

                if (filter.Field == null) { break; }
                collection.Filters.Add(filter);
                idex++;
            }

            return collection;
        }
    }

    public class KendoGridFilter
    {
        public string Field { get; set; }
        public string Operator { get; set; }
        public string Value { get; set; }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;

namespace KendoGridExample.KendoGridUtilities
{
    public static class LinqFilteringUtility
    {
        public static IEnumerable<T> MultipleFilter<T>(this IEnumerable<T> data,
          List<KendoGridFilter> filterExpressions)
        {
            if ((filterExpressions == null) || (filterExpressions.Count <= 0))
            {
                return data;
            }

            IEnumerable<T> filteredquery = from item in data select item;

            for (int i = 0; i < filterExpressions.Count; i++ )
            {
                var index = i;

                Func<T, bool> expression = item =>
                    {
                        var filter = filterExpressions[index];
                        var itemValue = item.GetType()
                            .GetProperty(filter.Field)
                            .GetValue(item, null);

                        if (itemValue == null)
                        {
                            return false;
                        }

                        var value = filter.Value;
                        switch (filter.Operator)
                        {
                            case "eq":
                                return itemValue.ToString() == value;
                            case "startswith":
                                return itemValue.ToString().StartsWith(value);
                            case "contains":
                                return itemValue.ToString().Contains(value);
                            case "endswith":
                                return itemValue.ToString().EndsWith(value);
                        }

                        return true;
                    };

                filteredquery = filteredquery.Where(expression);
            }

            return filteredquery;
        }
    }
}
```

- Similar to the "KendoGridSorterCollection" class, the factory method "BuildCollection" in the "KendoGridFilterCollection" class builds an instance of the "KendoGridFilterCollection" class based on the "HttpRequestBase" object sent from the Kendo grid. When it is successful, the "KendoGridFilterCollection" object will have the list of the "KendoGridFilter" objects, which tell us how the users want to filter the data.
- The "MultipleFilter" is an extension method on an "IEnumerable" collection. We can call this method by passing the list of the "KendoGridFilter" objects to filter the data.
- The "MultipleFilter" method only implemented the "eq", "startswith", "contains", and "endswith" operations. If you want to support more operations, you can modify this extension method.
- These two classes support filtering on multiple columns.

## The Server Side Code



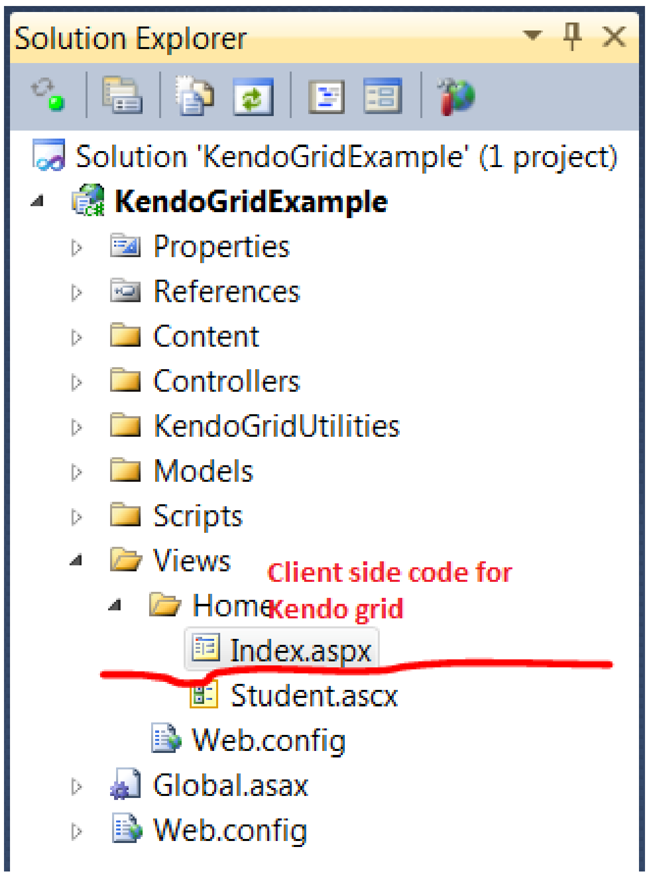
The example in this article uses server binding. The server will send the data to display in the grid. The "HomeController.cs" file is implemented as the following.

```
using System.Linq;
using System.Web.Mvc;
using KendoGridExample.KendoGridUtilities;
using KendoGridExample.Models;

namespace KendoGridExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

    public ActionResult GetAStudent(int id)
        {
            var student = StudentRepository.Students.Single(s => s.Id == id);
            return PartialView("Student", student);
        }

        public ActionResult LoadStudents(int page, int pageSize, int take
        , bool? activeOnly)
        {
            var sorterCollection = KendoGridSorterCollection.BuildCollection(Request);
            var filterCollection = KendoGridFilterCollection.BuildCollection(Request);

            var students = (!(activeOnly ?? false))
                                   ? StudentRepository.Students
                                   : StudentRepository.Students.Where(s => s.Active);
            var filteredStudents = students.MultipleFilter(filterCollection.Filters);
            var sortedStudents = filteredStudents.MultipleSort(sorterCollection.Sorters)
                        .ToList();
            var count = sortedStudents.Count();
            var data = (from v in sortedStudents.Skip((page - 1) * pageSize)
                                .Take(pageSize) select v).ToList();

            var jsonData = new { total = count, data };
            return Json(jsonData, JsonRequestBehavior.AllowGet);
        }
    }
}
```

- The "Index" method loads the application's main MVC view page.
- The "GetAStudent" method simply returns a partial view with the information of a single student. Since this article is about Kendo grid, I am not going to spend much time on returning partial views to the web page through Ajax calls. If you are interested, you can take a look at my earlier article "Load Clean HTML Fragments using jQuery and MVC".
- The "LoadStudents" method is the place where the server sends the list of the students to the Kendo grid to display. When Kendo grid retrieving the data from the server, it passes the information required by the server to load the data. In the "LoadStudents" method, the sorting and filtering information is

obtained from the "Request" object and then used to filter and sort the students. The Kendo grid does not need the "LoadStudents" method to return all the students. It only needs the list of the students to be displayed in the page. Along with the total count of the students, the data is return as a JSON result.

## The Client Side Code



To use the Kendo grid, we will need to link the required css and JavaScript files in the "Index.aspx" page.

```
<link type="text/css" rel="stylesheet"
    href="<%=Url.Content("~/Content/Site.css") %>" />
<link type="text/css" rel="stylesheet"
    href="<%=Url.Content("~/Content/kendo/kendo.common.min.css") %>" />
<link type="text/css" rel="stylesheet"
    href="<%=Url.Content("~/Content/kendo/kendo.default.min.css") %>" />
    <link type="text/css" rel="stylesheet"
    href="<%=Url.Content("~/Content/jQueryUI/jquery-ui.min.css") %>" />

<script type="text/javascript"
    src="<%=Url.Content("~/Scripts/jquery-2.0.3.min.js") %>"></script>
<script type="text/javascript"
    src="<%=Url.Content("~/Scripts/kendo.web.min.js") %>"></script>
<script type="text/javascript"
    src="<%=Url.Content("~/Scripts/jquery-ui.min.js") %>"></script>
```

We do not need the jQuery UI to use Kendo grid. The reason why I included the "jquery-ui.min.js" file is to use its "dialog" window to display the information for a single student in order to show you how to handle the double click event on a grid row. The html content of the "Index.aspx" page is the following.

```
<body>
    <div id="dialog" style="display: none"></div>
    <div>
        <div><input type="checkbox" id="chkShowActive" checked="checked"
                onclick="return GridExample.Refresh()"/>
            Show active students only</div>
        <div id="GridContainer"></div>
    </div>
</body>
```

- The "dialog" div will be used by the jQuery UI to show a dialog window to display the information of the single student.

- The "chkShowActive" checkbox will be used to show you how to refresh the Kendo grid and how to send the updated data to the server.
- The "GridContainer" will be used by Kendo to create the Kendo grid to display the information of the students.

The JavaScript code to create and manipulate the Kendo grid is the following.

```
var gridReadUrl = '<%=Url.Action("LoadStudents") %>';
var getStudentUrl = '<%=Url.Action("GetAStudent") %>';

var GridExample = function () {
    var initialize = function() {
        var container = $('#GridContainer').html('<div></div>');
        var grid = $('div', container);

        var filterOption = {
            extra: false,
            operators: {
                string: {
                    eq: "Is Equal To",
                    startswith: "Starts With",
                    contains: "Contains",
                    endswith: "Ends With"
                }
            }
        };

        var columnOptions = [
            {
                field: "Id",
                title: "Id",
                width: 100,
                type: "integer",
                filterable: false
            },
            {
                field: "LastName",
                title: "Last Name",
                filterable: filterOption
            },
            {
                field: "FirstName",
                title: "First Name",
                filterable: filterOption
            },
            { field: "Score", title: "Score", filterable: false },
            { field: "Active", title: "Active", filterable: false }
        ];

        var gridOptions = {
            dataSource: {
                transport: {
                    read: {
                        url: gridReadUrl,
                        type: "POST",
                        data: function() {
                            return {
                                activeOnly: $('#chkShowActive').is(':checked')
                            };
                        }
                    }
                },
                schema: { data: "data", total: "total", model: { id: "Id" } },
                serverPaging: true,
                serverSorting: true,
                serverFiltering: true
            },
            columns: columnOptions,
            height: 300,
            sortable: { mode: "multiple", allowUnsort: true },
            filterable: true,
            pageable: {
                pageSize: 30,
                refresh: true,
                messages: {
                    refresh: "Refresh the grid"
                }
            }
        };

        grid.kendoGrid(gridOptions)
            .delegate("tbody>tr", "dblclick", function(e) {
                var row = e.currentTarget;
                var selectedItem = grid.data("kendoGrid").dataItem(row);

                var ajax = $.ajax({
                    url: getStudentUrl,
                    type: "POST",
                    data: { id: selectedItem.Id },
                    dataType: "html"
                });

                ajax.success(function(data) {
                    var dialog = $('#dialog');
                    dialog.html(data);
                    dialog.dialog({
                        modal: true
                    });
                });
            });
    };

    var refresh = function () {
        var grid = $('#GridContainer>div');
        if (grid.length < 1) {
            return;
        }

        grid.data("kendoGrid").dataSource.read();
    };

    return {
        Initialize: initialize,
        Refresh: refresh
    };
} ();

$(document).ready(function () {
    GridExample.Initialize();
});
```

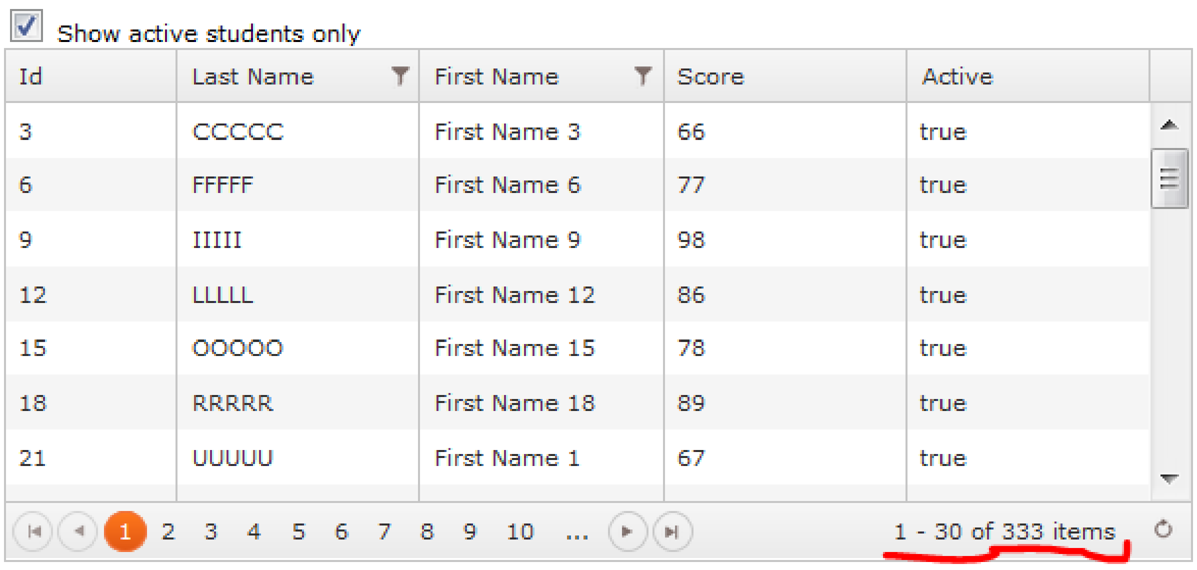
The "GridExample" variable is a JSON object, which exposes two functions, "Initialize" and "Refresh". The "Initialize" function initializes the Kendo grid and the "Refresh" function refreshes the grid. To initialize a Kendo grid, we can simply call the "kendoGrid" function on the jQuery object on the container div of the grid by passing a JSON object indicating how we want the grid to behave. The important information passed to the "kendoGrid" function includes the following:

- The url and the http method to load the data to display;
- The additional data to be passed to the server when loading the data from the server;
- The columns that we want to display in the grid and how we want them to be displayed;
- The dimensions of the grid and the paging information;
- The sorting and filtering configurations.

To handle the double click event on a grid row, we can associate a delegate to each row. In the delegate, we first retrieve the Id of the double clicked student and then issue an Ajax call to load a partial view with the information of the student. The partial view is then displayed in a jQuery UI dialog box.

## Run the Application

As simple as how Kendo is designed to be, we now finish the example application and we now have a fully functional Kendo grid. We can then test run it. When the web page is first loaded, an Ajax call is issued by Kendo to retrieve the information to display in the web page.

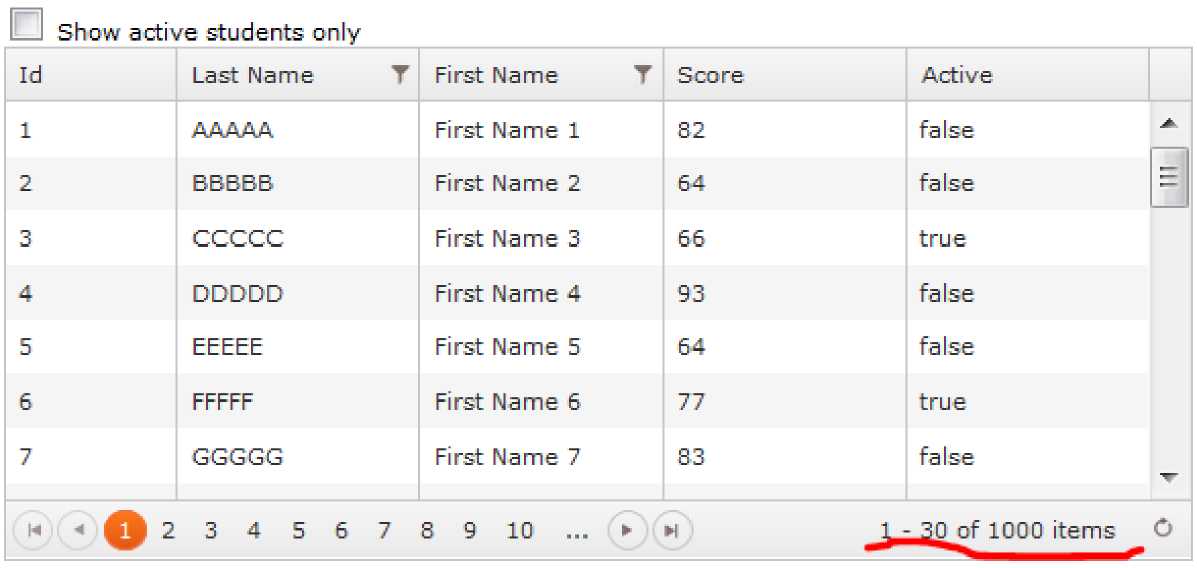


☑ Show active students only

| Id | Last Name | First Name | Score | Active |
|---|---|---|---|---|
| 3 | CCCCC | First Name 3 | 66 | true |
| 6 | FFFFF | First Name 6 | 77 | true |
| 9 | IIIII | First Name 9 | 98 | true |
| 12 | LLLLL | First Name 12 | 86 | true |
| 15 | OOOOO | First Name 15 | 78 | true |
| 18 | RRRRR | First Name 18 | 89 | true |
| 21 | UUUUU | First Name 1 | 67 | true |

1 2 3 4 5 6 7 8 9 10 ... 1 - 30 of 333 items

By default we only show the active students. If you uncheck the "Show active students only" checkbox, the grid is refreshed to show both active and inactive students.

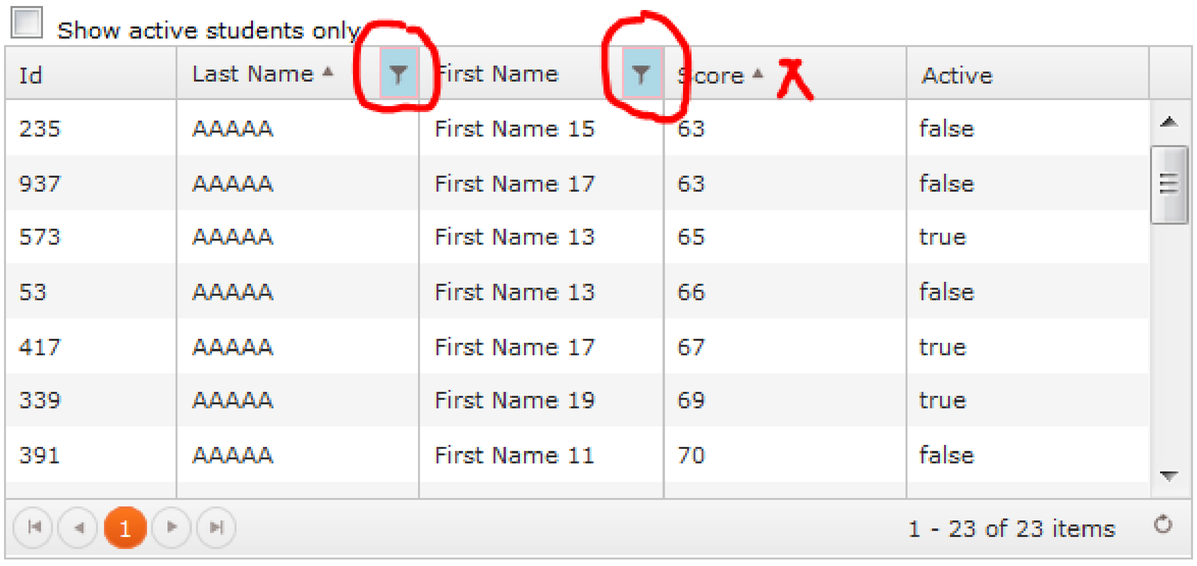


Show active students only

| Id | Last Name | First Name | Score | Active |
|---|---|---|---|---|
| 1 | AAAAA | First Name 1 | 82 | false |
| 2 | BBBBB | First Name 2 | 64 | false |
| 3 | CCCCC | First Name 3 | 66 | true |
| 4 | DDDDD | First Name 4 | 93 | false |
| 5 | EEEEE | First Name 5 | 64 | false |
| 6 | FFFFF | First Name 6 | 77 | true |
| 7 | GGGGG | First Name 7 | 83 | false |

1 2 3 4 5 6 7 8 9 10 ... 1 - 30 of 1000 items

We can then try the filtering and sorting capability of the Kendo grid by applying some filtering text and clicking on the header of some columns.

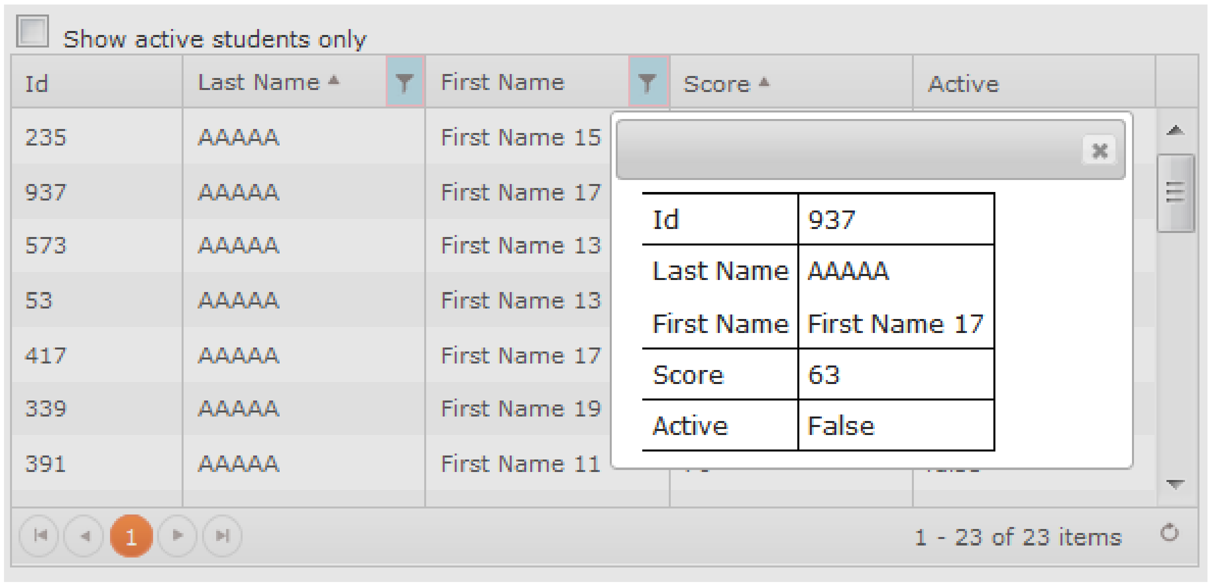


Show active students only

| Id | Last Name ▲ | First Name | Score ▲ | Active |
|---|---|---|---|---|
| 235 | AAAAA | First Name 15 | 63 | false |
| 937 | AAAAA | First Name 17 | 63 | false |
| 573 | AAAAA | First Name 13 | 65 | true |
| 53 | AAAAA | First Name 13 | 66 | false |
| 417 | AAAAA | First Name 17 | 67 | true |
| 339 | AAAAA | First Name 19 | 69 | true |
| 391 | AAAAA | First Name 11 | 70 | false |

1 1 - 23 of 23 items

If we double click on a row in the grid, the event is handled and the information of this student is loaded through a partial view and displayed in a jQuery UI dialog box.



Show active students only

| | |
|---|---|
| Id | 937 |
| Last Name | AAAAA |
| First Name | First Name 17 |
| Score | 63 |
| Active | False |

1 1 - 23 of 23 items

## How About Editing in a Kendo Grid?

The Kendo grid is not limited to display the data. It comes with a comprehensive set of editing capabilities. But I am not going to explore these capabilities in this example. From my experience, I found it is easier to implement my own popup dialog boxes to add new and edit existing data entries. We can easily add textboxes or other input controls in the popup to allow the users to change the data. We can use Ajax calls to send the data to the server to update the data source. If the Ajax call is successful, we can simply refresh the Kendo grid to display the updated data.

## Points of Interest

- The example showed you how to setup the environment in your application to use Kendo web controls. It also showed you how to implement some basic capabilities of the Kendo grid.
- The example implemented the sorting and filtering in Linq. If you have a database behind your application, you can choose to implement the sorting and filtering in your database. From my own test, I found that the extension methods provided by this article for the multiple-column sorting and filtering are very efficient.
- The example did not explore the editing capabilities provided by Kendo, since my personal preference is to implement my own dialog popup window to achieve the adding new item and editing an existing item capability.
- Kendo comes with different versions and some of the versions are paid versions. These versions should be much easier to use than the open source version, so I will strongly recommend you to check them out.
- I hope you like my postings and I hope this article can help you one way or the other.

## History

First revision - 10/31/2013

## License

## Share

## About the Author

**Dr. Song Li**

United States

I have been working in the IT industry for some time. It is still exciting and I am still learning. I am a happy and honest person, and I want to be your friend.

## Comments and Discussions

**8 messages** have been posted for this article Visit **http://www.codeproject.com/Articles/675879/An-Example-to-Use-Kendo-Grid** to post and view comments on this article, or click **here** to get a print view with messages.