INSTANT

Short | Fast | Focused

Kendo UI Grid

Learn an amazing JavaScript framework that will boost the look and function of your tabular data

James R. Lamar

[PACKT] PUBLISHING

Instant Kendo UI Grid

Learn an amazing JavaScript framework that will boost the look and function of your tabular data

James R. Lamar

PACKT PUBLISHING

BIRMINGHAM - MUMBAI

Instant Kendo UI Grid

First published: July 2013

Production Reference: 1220713

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84969-913-6

www.packtpub.com

Credits

Author

James R. Lamar

Reviewers

Nirbhay Anand

Omkar Patil

Acquisition Editor

Kartikey Pandey

Commissioning Editor

Llewellyn Rozario

Meeta Rajani

Technical Editor

Dennis John

Copy Editor

Aditya Nair

Project Coordinator

Esha Thakker

Proofreader

Paul Hindle

Production Coordinator

Conidon Miranda

Cover Work

Conidon Miranda

About the Author

James R. Lamar began work in design over 15 years ago and transitioned into web development in 2007. He is a self-taught programmer with experience in ColdFusion, PHP, and ASP.NET. He has worked in all aspects of software development for companies such as FedEx, Yusen/NYK Logistics, and the U.S. Navy Recruiting Command. He is also the owner and Chief Software Engineer of Nautilus Technology Solutions Inc.

James thrives on learning as much as he can about engineering software that is elegant and intuitive.

I would like to thank Packt Publishing and Telerik/Kendo UI for making this book possible. I would like to thank Matt Gifford for his support and encouragement throughout the development of this book. I would also like to give the sincerest thanks to my wife and best friend Aimee for being my greatest source of encouragement and wisdom.

About the Reviewers

Nirbhay Anand has done his Masters in Computer Application and is also a Microsoft Certified Technology Specialist with 7 years of experience in Software Product Development. Nirbhay has developed software for different domains such as Investment Banking, Online Retailing, Digital Media, Warehousing, and IM Security.

He started his career with the Indian branch of Lightbulb Technologies, a Houston-based software product development company. Here he worked on IM Security, setting up and implementing CA servers and two-way certificate-based authentication using WCF Service. Post this, he worked with "The 5th Dimension", a startup that conceptualized, designed, and developed software in the areas of Investment Banking and Online Retailing of designer creations based in Pune, India. At The 5th Dimension, Nirbhay was deeply involved in enhancing functionalities of a highly complex Warehouse Management System for a Canadian third-party Logistics company.

Presently, he is associated with Synechron Technologies, where he is working on projects involving various technologies such as Kendo, MVC, and WCF.

He has a passion for writing blogs (`http://nirbhay-mcts.blogspot.com/`) and learning new technologies.

I would like to thank my wife Vijeta and daughter Navya for supporting me on my late evening work at home. I would also like to thank all my friends and family for their never-ending support.

Omkar Patil is currently working as a Senior Architect for the Global Technology and Architecture group of SunGard. He has about 14 years of experience in architecture, design, and development of web applications. Starting with server-side Enterprise Java, he has shifted his focus to frontend development of web and mobile applications in the last few years. His current skillset consists of JavaScript, jQuery, Kendo UI, AngularJS, and Node.js.

In his spare time, he enjoys reading, tinkering with new technologies, and contributing to open source software. He is a committer for an Angular-Kendo project that integrates Kendo UI with AngularJS (`https://github.com/kendo-labs/angular-kendo`).

I would like to thank my wife Anu for her support and encouragement and simply for bearing me being constantly glued to the computer screen for the past 11 years.

---

www.packtpub.com

Support files, eBooks, discount offers and more

You might want to visit `www.packtpub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.packtpub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.packtpub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.packtpub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.packtpub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

This book introduces you to an amazing JavaScript framework that will boost the look and function of your tabular data. Kendo Grids are the perfect fit if you need a powerful set of tools to let users manipulate data or even if you just want a nice way to dress up existing tables.

Instant Kendo UI Grid is a practical, hands-on guide that will provide dozens of working examples and also serve as a reference for customizing your grids in no time.

What this book covers

See what Kendo UI Grids can do (Simple), demonstrates how to transform a simple table into a Kendo Grid.

How to change your theme (Simple), demonstrates how to change your Kendo theme.

Using built-in sorting (Simple), demonstrates the built-in sorting feature of Kendo Grids.

Using built-in paging (Simple), demonstrates the built-in paging feature of Kendo Grids.

Using built-in grouping (Simple), demonstrates the built-in grouping feature of Kendo Grids.

Using conditional JavaScript (Simple), demonstrates how to use JavaScript and templates to create dynamic content inside your Kendo Grid.

Customizing and formatting column data (Simple), demonstrates how to customize and format the data in a Kendo Grid column.

Filtering column data (Simple), demonstrates how to allow the user to filter the data in a Kendo Grid column.

Working with aggregates (Intermediate), demonstrates how to work with aggregates; for example, counting the number of items in a column.

Getting a handle on toolbar templates (Intermediate), demonstrates how to use custom toolbars in your grid.

Customizing grid rows (Intermediate), demonstrates how to use row templates to customize rows in a Kendo Grid.

Creating a grid within a grid (Advanced), demonstrates how to create a Kendo Grid within a grid.

Working with remote data (Advanced), demonstrates how to work with remote data such as cross-domain JSONP.

Batch editing (Advanced), demonstrates how to allow the user to make batch edits in the grid.

Inline editing (Advanced), demonstrates how to allow the user to make inline edits in the grid.

Working with user events (Advanced), demonstrates how to work with user events in the grid.

Advanced API example (Advanced), demonstrates how to get the most out of Kendo Grids by tapping directly into the Kendo API.

What you need for this book

To run the examples in this book, the following software are required:

- A text editor of any kind. I recommend Sublime Text 2.
- A web browser of any kind that is compatible with jQuery. I recommend Chrome, Firefox, or Safari.

Who this book is for

This book is for anyone with some basic HTML, CSS, and JavaScript experience. Intermediate and advanced users will find several helpful examples as well. If your work predominantly involves designing or developing, then this book is for you.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "Reopen the same file you used or created from the `1-initializing.html` exercise file".

A block of code is set as follows:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.metro.
min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
```

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

Instant Kendo UI Grid

Welcome to *Instant Kendo UI Grid*. This book promises to introduce you to an amazing framework that can boost the look and function of your tabular data. Kendo Grids are the perfect fit if you need a powerful set of tools to let users manipulate data, or even if you just want a nice way to dress up the existing tables.

See what Kendo UI Grids can do (Simple)

This recipe demonstrates how to transform a simple table into a Kendo Grid.

Getting ready

You will need to download the Kendo Web framework. You also have the option of downloading the exercise files that accompany this book, which can help you get started or check your work as you follow along. You can download the commercial or open source version of Kendo Web at `http://www.kendoui.com/download.aspx`. You'll want to extract the minified versions of the files referenced below to their respective folders. Also, make sure to copy at least the folders named `Default` and `Metro` from the `styles` folder in the Kendo download. Put your Kendo `js` and `styles` folders in another folder on the root named `kendo`.

How to do it...

If you've downloaded the exercise files, open `1-initializing.html` in your editor. If you do not have access to the exercise files, copy the following code into your preferred text editor or IDE. I prefer to use Sublime Text 2 (`http://www.sublimetext.com/`) as my text editor, which is free to try.

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 1 - See What Kendo UI Grids Can
Do</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
     $(document).ready(function(){
        $("#myGrid").kendoGrid();
     });
   </script>
<table id="myGrid">
   <thead>
      <tr>
         <th>First Name</th>
         <th>Last Name</th>
         <th>Rank</th>
      </tr>
   </thead>
   <tbody>
      <tr>
         <td>Jim</td>
         <td>Kirk</td>
         <td>Captain</td>
      </tr>
      <tr>
         <td>Jim</td>
         <td>Raynor</td>
         <td>Captain</td>
```

```
        </tr>
        <tr>
           <td>Sarah</td>
           <td>Kerrigan</td>
           <td>Ghost Recon</td>
        </tr>
     </tbody>
</table>
</body>
</html>
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

How it works...

A Kendo Grid is basically a table or structured set of data that is transformed by the Kendo JavaScript engine. Kendo stylesheets will help make the Kendo Grid look pretty, but it is the jQuery-based JavaScript engine that does the magic. Kendo also utilizes HTML5 as part of its framework, which we will demonstrate later.

The real key to making this example work is telling Kendo what `div` or `table` is to be used to make a Kendo Grid. You tell Kendo by using the jQuery syntax, usually by matching the ID or class of the table, followed by calling the `kendoGrid()` function as shown in the following line of code:

```
$("#myGrid").kendoGrid();
```

There's more...

The reader should already be familiar with CSS and HTML, but if you want more information about jQuery or JavaScript, there are a plethora of resources available. There are also a number of fine books available from Packt Publishing (`http://www.packtpub.com`) that will serve as an invaluable resource for you in your design/development career. I also recommend the W3 Schools website (`http://www.w3schools.com`) and the jQuery website (`http://jquery.com`) itself as a good starting point.

Caution!

If you choose to follow along by typing out the code yourself instead of using the exercise files and copying and pasting, be sure to reference your `jquery.js` file *before* your `kendo.web.min.js` file. The Kendo library is dependent on jQuery, so jQuery must be loaded first.

More info

The open source and commercial versions of Kendo Web will also contain the full source code, located in the folder named `src`. You'll want to use the minified versions for production, but you may want to use the full version for development. Minified versions of CSS or JavaScript are compacted to save space and will not provide helpful information if you have an error or want to make your own customizations.

Disclaimer

While many applications may qualify to legally use the open source version of Kendo Web under GPLv3, you are strongly encouraged to have any application reviewed by your own legal authority. To be safe, you can always purchase the commercial version of Kendo Web, which has the added benefit of providing support and regular updates to the framework.

How to change your theme (Simple)

This recipe demonstrates how to change your Kendo theme.

Getting ready

Reopen the same file you used or created from the `1-initializing.html` exercise file. In this recipe, you'll be making a simple change to realize how easy it is to change the theme of your grid.

How to do it...

In the `<head>` section of your document, change the reference to `kendo.default.min.css` to any of the other themes included in the `style` folder with your Kendo download. The following is an example that should be easy to see once you save your changes and refresh the page in the browser:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
```

```
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.metro.
min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
```

How it works...

Without referencing a Kendo theme stylesheet, your grid will essentially look like a plain table with borders. Kendo generates the same classes every time a grid is initialized, so the theme simply defines the style of those classes. If you look inside any particular theme folder, for example `Metro`, you'll notice that Kendo uses sprites to keep the file size and load time down.

Using built-in sorting (Simple)

This recipe demonstrates the built-in sorting feature of Kendo Grids.

Getting ready

Open `2-sorting.html` in your editor or create a new HTML file with the same name.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 2 - Using Built-in Sorting</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
     $(document).ready(function(){
        $("#myGrid").kendoGrid({
         sortable: {
```

```
            mode: "multiple", // a value of "single" would only allow
the user to sort one column at a time
            allowUnsort: true
         }
        });
     });
   </script>
<table id="myGrid">
   <thead>
      <tr>
         <th>First Name</th>
         <th>Last Name</th>
         <th>Rank</th>
      </tr>
   </thead>
   <tbody>
      <tr>
         <td>Jim</td>
         <td>Kirk</td>
         <td>Captain</td>
      </tr>
      <tr>
         <td>Jim</td>
         <td>Raynor</td>
         <td>Captain</td>
      </tr>
      <tr>
         <td>Sarah</td>
         <td>Kerrigan</td>
         <td>Ghost Recon</td>
      </tr>
      <tr>
         <td>Jean-Luc</td>
         <td>Picard</td>
         <td>Captain</td>
      </tr>
      <tr>
         <td>Steve</td>
         <td>Rogers</td>
         <td>Captain</td>
   </tr>
   </tbody>
</table>
</body>
</html>
```

How it works...

When you add the `sortable` parameter to the grid, Kendo automatically includes all of the client-side functionality you need to sort the columns of your grid. The `mode` and `allowUnsort` parameters provided to the `sortable` parameter are optional. You can also use the default sort functionality by simply defining `sortable` as `true`:

```
$("#myGrid").kendoGrid({
sortable: true
});
```

Using built-in paging (Simple)

This recipe demonstrates the built-in paging feature of Kendo Grids.

Getting ready

Open `3-paging.html` in your editor or create a new HTML file with the same name.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 3 - Using Built-in Paging</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
  $(document).ready(function(){
     $("#myGrid").kendoGrid({
      pageable: {
         pageSize: 2,
         previousNext: true, // default true
```

```
            numeric: true, // default true
            buttonCount: 2, // default 10, controls how many buttons are
   shown which represent pages of data
            refresh: true, // default false
            input: true, // default false
            messages: {
               display: "{0} - {1} of {2} records",
               empty: "Nothing to display",
               page: "Page",
               of: "of {0}",
               itemsPerPage: "records per page",
               first: "Go to the first page",
               previous: "Go to the previous page",
               next: "Go to the next page",
               last: "Go to the last page",
               refresh: "Refresh"
            }
         }
        });
     });
   </script>
   <table id="myGrid">
      <thead>
         <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Rank</th>
         </tr>
      </thead>
      <tbody>
         <tr>
            <td>Jim</td>
            <td>Kirk</td>
            <td>Captain</td>
         </tr>
      <tr>
            <td>Jim</td>
            <td>Raynor</td>
            <td>Captain</td>
         </tr>
         <tr>
            <td>Sarah</td>
            <td>Kerrigan</td>
            <td>Ghost Recon</td>
```

```
        </tr>
        <tr>
           <td>Jean-Luc</td>
           <td>Picard</td>
           <td>Captain</td>
        </tr>
        <tr>
           <td>Steve</td>
           <td>Rogers</td>
           <td>Captain</td>
        </tr>
     </tbody>
  </table>
  </body>
  </html>
```

How it works...

Like with the sorting feature, you can simply give the `pageable` parameter a value of `true`, but this code shows several examples of refining the paging feature of the grid.

Using built-in grouping (Simple)

This recipe demonstrates the built-in grouping feature of Kendo Grids.

Getting ready

Open `4-grouping.html` and `personData.js` in your editor. If you don't have the exercise files, create a blank document for each and follow the set of instructions in the following section.

How to do it...

1. First, create a new JavaScript file in a new folder named `js`. Then, copy the following code into the JavaScript file and save it:

```
var personData = [{
    FirstName : "Jim",
    LastName : "Kirk",
    Rank : "Captain",
    DOB: "03/22/2233",
    PersonId : 1
}, {
```

```
        FirstName : "Jim",
        LastName : "Raynor",
        Rank : "Captain",
        DOB: "06/23/2470",
        PersonId : 2
    }, {
        FirstName : "Jean-Luc",
        LastName : "Picard",
        Rank : "Captain",
        DOB: "07/13/2305",
        PersonId : 3
    }, {
        FirstName : "Sarah",
        LastName : "Kerrigan",
        Rank : "Ghost Recon",
        DOB: "09/07/2471",
        PersonId : 4
    }, {
        FirstName : "Steve",
        LastName : "Rogers",
        Rank : "Captain",
        DOB: "07/04/1922",
        PersonId : 5
    }];
```

2. After creating your JavaScript file, copy the following code into your new HTML file:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
<script src="js/personData.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 4 - Using Built-in Grouping</
h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
  $(document).ready(function() {
```

```
            var myDataSource = new kendo.data.DataSource({
             data: personData,
             group: { field: "Rank" },
                schema: {
                    model: {
                        fields: {
                            FirstName: { type: "string" },
                            LastName: { type: "string" },
                            Rank: { type: "string" },
                            PersonId: { type: "number" }
                        }
                    }
                }
          } );
            $("#myGrid").kendoGrid({
             dataSource: myDataSource,
             groupable: true // allows the user to alter what field
the grid is grouped by
            });
       });
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

You've probably noticed that we are no longer using the actual HTML table as our datasource. We are actually defining our datasource by manually building a JavaScript array with the relevant columns and data we had in the table that was used previously. In this example, we are defining `kendo.data.DataSource`, which expects its own structure of data in order to accurately populate the grid. More information about things you can further refine in the Kendo datasource can be found in the Kendo documentation. We will cover several of them in this book.

We also replaced our HTML table with a simple `<div>` tag with an ID equal to `myGrid`. Next, we separated defining our Kendo datasource from defining our Kendo Grid. This is a best practice that will be followed throughout the rest of the book.

Using conditional JavaScript (Simple)

This recipe demonstrates how to use JavaScript and Kendo templates to create dynamic content inside your Kendo Grid.

Getting ready

If you've downloaded the exercise files, open `5-javascript.html` in your editor or create a new HTML file with the same name.

How to do it...

Copy the following code into your new HTML file:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
<script src="js/personData.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 5 - Using Conditional JavaScript</
h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
  $(document).ready(function() {
        var myDataSource = new kendo.data.DataSource({
         data: personData,
            schema: {
                model: {
                    fields: {
                        FirstName: { type: "string" },
                        LastName: { type: "string" },
                        Rank: { type: "string" },
                        PersonId: { type: "number" }
                    }
                }
```

```
            }
        } );
          $("#myGrid").kendoGrid({
           dataSource: myDataSource,
           columns: [
                 "FirstName",
                 { field: "Rank", template: kendo.template(
$("#rankTemplate").html() )  }
             ]
          });
    });
</script>
<script type="text/x-kendo-template" id="rankTemplate">
    #if( Rank == 'Ghost Recon') {#
       #= Rank# !!!
    #} else {#
       #= Rank#
    #}#
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

You should have noticed several new things in this code sample. First, we actually define the columns in our grid, not just in the datasource model. Second, we have simply referenced the name of the field we wanted in the column. As a result, Kendo will render the string representation of the data and substitute a column header that matches the column name. Templates give us granular control of the presentation of each column.

Lastly, we are using a more advanced template for `Rank` to demonstrate how you can use JavaScript to process and evaluate the content of your grid before rendering it to the browser. Notice that the # symbol sets apart the JavaScript code included in the `<script>` tag. It's easier if you separate your JavaScript as is done in this example, as Kendo also looks at `#=` symbols to output a column field as literal text.

Customizing and formatting column data (Simple)

This recipe demonstrates how to customize and format the data in a Kendo Grid column.

Getting ready

Open `6-customizing.html` in your editor or create a new document.

How to do it...

Copy the following code into your new HTML file:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
<script src="js/personData.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 6 - Customizing and Formatting
Column Data</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
  $(document).ready(function() {
        var myDataSource = new kendo.data.DataSource({
         data: personData,
            schema: {
                model: {
                    fields: {
                        FirstName: { type: "string" },
                        LastName: { type: "string" },
                        Rank: { type: "string" },
                        DOB: { type: "date" },
                        PersonId: { type: "number" }
                    }
                }
```

```
            }
        } );
          $("#myGrid").kendoGrid({
           dataSource: myDataSource,
           columns: [
                 { field: "Name", title: "Full Name", width: "300px",
template: '#=FirstName# #=LastName#' },
              { field: "DOB", title: "Date of Birth", width: "300px",
format: "{0:yyyy-MM-dd}" },
                 { field: "Rank", title: "Rank", width: "300px" }
             ]
          });
    });
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

As you may have noticed, in this example, we chose to define our `DOB` field in our datasource for us to be able to use it in the grid. We also added a `format` parameter that accepts a filter pattern for our data. In this case, we want to format the date so that the user can properly sort the dates. We also concatenated the `FirstName` and `LastName` fields and gave that column a title of `Full Name`. Lastly, we added a `width` parameter to each column definition.

Filtering column data (Simple)

This recipe demonstrates how to allow the user to filter the data in a Kendo Grid column.

Getting ready

Open `7-filtering.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
```

```
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
<script src="js/personData.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 7 - Filtering Column Data</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
  $(document).ready(function() {
        var myDataSource = new kendo.data.DataSource({
         data: personData,
            schema: {
                model: {
                    fields: {
                        FirstName: { type: "string" },
                        LastName: { type: "string" },
                        Rank: { type: "string" },
                        DOB: { type: "date" },
                        PersonId: { type: "number" }
                    }
                }
            }
      } );
        $("#myGrid").kendoGrid({
         dataSource: myDataSource,
         filterable: {
            extra: false, // true if a second "AND/OR"  filter is
needed
            operators: {
               string: {
                  startswith: "Starts with",
                  eq: "Equals",
                  neq: "Not equal to"
               }
            }
         },
         columns: [
               { field: "Name", title: "Full Name", width: "300px",
template: '#=FirstName# #=LastName#', filterable: false },
            { field: "DOB", title: "Date of Birth", width: "300px",
format: "{0:yyyy-MM-dd}", filterable: { ui: "datepicker" } },
```

```
                { field: "Rank", title: "Rank", width: "300px" }
            ]
        });
    });
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

In this recipe, we defined the grid as `filterable` and passed some additional parameters to our `filterable` parameter to control how many filters there are and what kind of text is shown when the user tries to filter the column. We also specified two special scenarios in our column definitions. We made the `FullName` column unfilterable and then we made the `DOB` column filterable using a Kendo `DatePicker` object.

There's more...

There is a lot more that you can do with filtering, such as using predefined dropdowns or multiple search parameters. However, a word of caution is to always keep it simple for your users. Although you might not have a problem with logic such as "starts with *** and does not equal ***", some users may find it confusing. No matter how you choose to implement your grid, filtering can be a very helpful tool for users to parse through large amounts of data.

Working with aggregates (Intermediate)

This recipe demonstrates how to work with aggregates; for example, counting the number of items in a column.

Getting ready

Open `8-aggregate.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
```

```
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
<script src="js/personData.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 8 - Working with Aggregates</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
  $(document).ready(function() {
        var myDataSource = new kendo.data.DataSource({
         data: personData,
            schema: {
                model: {
                    fields: {
                        FirstName: { type: "string" },
                        LastName: { type: "string" },
                        Rank: { type: "string" },
                        DOB: { type: "string" },
                        PersonId: { type: "number" }
                    }
                }
            },
         aggregate: [
            { field: "Name", aggregate: "count" }
         ]
      } );
        $("#myGrid").kendoGrid({
         dataSource: myDataSource,
         columns: [
               { field: "Name", title: "Full Name", width: "300px",
template: '#=FirstName# #=LastName#', footerTemplate: "Total Count:
#=count#" },
            { field: "DOB", title: "Date of Birth", width: "300px",
format: "{0:yyyy-MM-dd}" },
            { field: "Age", title: "Age", width: "300px", template:
kendo.template( $("#ageTemplate").html() ), headerTemplate: "Age,
Assuming Year is 2500"  },
               { field: "Rank", title: "Rank", width: "300px" }
          ]
        });
   });
```

```
</script>
<script type="text/x-kendo-template" id="ageTemplate">
   #age = DOB;#
   #age = 2500 - age.substring(age.length - 4, age.length);#
   #=age#
</script>

<div id="myGrid"></div>
</body>
</html>
```

How it works...

In this example, we added the `aggregate` parameter to our datasource and specified that we wanted to make the record count of the `Name` column available. If we were working with a numeric datatype, we could also specify that we wanted the sum, min, max, or average. Unfortunately, we are limited to those variables when outputting to `footerTemplate`, such as the one we are using on the `Name` column.

You'll also notice that I added a column named `Age`, which is calculated dynamically from `DOB`. It's worth mentioning again that you cannot access dynamic variables in `footerTemplate` like you can in `template`. So, in this case, if you wanted to output the average age in the footer template of the `Age` column, you couldn't do it using Kendo aggregates.

In order to make the `Age` calculations work, we have to specify in the datasource that the `DOB` column is of type string and not date. The `format` parameter for `DOB` works the same either way.

Getting a handle on toolbar templates (Intermediate)

This recipe demonstrates how to use custom toolbars in your grid.

Getting ready

Open `9-toolbar.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
<script src="js/personData.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 9 - Getting a Handle on Toolbar
Templates</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
  $(document).ready(function() {
        var myDataSource = new kendo.data.DataSource({
         data: personData,
            schema: {
                model: {
                    fields: {
                        FirstName: { type: "string" },
                        LastName: { type: "string" },
                        Rank: { type: "string" },
                        DOB: { type: "date" },
                        PersonId: { type: "number" }
                    }
                }
            }
      } );
        $("#myGrid").kendoGrid({
         dataSource: myDataSource,
         toolbar: [
             { template: kendo.template($("#toolbarTemplate").html())
}
```

```
            ],
            columns: [
                  { field: "Name", title: "Full Name", width: "300px",
template: '#=FirstName# #=LastName#' },
                { field: "DOB", title: "Date of Birth", width: "300px" },
                  { field: "Rank", title: "Rank", width: "300px" }
              ]
           });
   });
   function toolClick() {
      alert('You clicked me!');
   }
</script>
<script id="toolbarTemplate" type="text/x-kendo-template">
   <a class="k-button" href="\#" onclick="return toolClick()">Custom
Toolbar</a>
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

There are several ways to add a toolbar to a Kendo Grid, but this recipe demonstrates the most accessible and extensible way. We simply added the `toolbar` parameter and passed it the `toolbarTemplate` template that we created. We could have simply included the markup we used in the `toolbarTemplate` template as a string argument to the `toolbar` parameter. A toolbar can be just about anything you want, but in this case, we simply made it a button that fires some JavaScript function.

There's more...

There is a nice example on the Kendo Grid demo website that shows how to bind the Kendo Grid to a Kendo dropdown. There are also ways to specify the `create`, `save`, `cancel`, and `destroy` toolbars, but we will use those later in the book when we perform **Create Read Update Delete** (**CRUD**) operations in our grid.

Customizing grid rows (Intermediate)

This recipe demonstrates how to use row templates to customize the rows in a Kendo Grid.

Getting ready

Open `10-rowtemplate.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
<script src="js/personData.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 10 - Customizing Grid Rows</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
  $(document).ready(function() {
        var myDataSource = new kendo.data.DataSource({
         data: personData
      } );
        $("#myGrid").kendoGrid({
         dataSource: myDataSource,
         rowTemplate: kendo.template($("#rowTemplate").html())
        });
   });
</script>
<table id="myGrid">
<thead>
<tr>
<th>
```

```
                Name
</th>
<th>
                Date of Birth
</th>
<th>
                Rank
</th>
</tr>
</thead>
<tbody>
<tr>
<td colspan="3"></td>
</tr>
</tbody>
</table>
<script id="rowTemplate" type="text/x-kendo-tmpl">
<tr>
<td>
<a href="http://wikipedia.org/wiki/#: data.FirstName #_#: data.
LastName #">#: data.FirstName # #: data.LastName #</a>
</td>
<td>
           #: data.DOB #
</td>
<td>
           #: data.Rank #
</td>
</tr>
</script>
</body>
</html>
```

How it works...

In this example, we actually removed several items from our grid and datasource, such as the model schema and the column model parameters. We initialized the grid from an empty table and added the `rowTemplate` parameter. Next, we defined what that row template would look like in the Kendo template below our table. We can't use conditional or dynamic JavaScript in this row template, but we can still use the data intelligently, just as we have here by creating `Wikipedia.org` links.

There's more...

In the next recipe, we will work through an example of using detailed templates to create a Kendo Grid within a grid. Detailed templates will definitely give us some more control, but if all you need is to insert some images or links, using a row template works just fine.

Creating a grid within a grid (Advanced)

This recipe demonstrates how to create a Kendo Grid within a grid.

Getting ready

Open `11-gridwithingrid.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 11 - Creating a Grid Within a
Grid</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
$(document).ready(function() {
    var outerServiceURL = "http://gonautilus.com/kendogen/KENDO.
cfc?method=getArtists";
   var innerServiceURL = "http://gonautilus.com/kendogen/KENDO.
cfc?method=getArt";
   var myDataSource = new kendo.data.DataSource({
        transport: {
            read: {
                url: outerServiceURL,
```

```
                dataType: "JSONP"
            }
        }
    });
    $("#myGrid").kendoGrid({
        dataSource: myDataSource,
        sortable: true,
        detailInit: detailInit,
        columns: [{ field: "ARTISTID", title: "Artist ID"},
            { field: "FIRSTNAME", title: "First Name"},
            { field: "LASTNAME", title: "Last Name"},
            { field: "EMAIL", title: "Email"},
            { field: "PHONE", title: "Phone Number"}]

    });
    function detailInit(e) {
        $("<div/>").appendTo(e.detailCell).kendoGrid({
            dataSource: {
                transport: {
                    read: {
                        url: innerServiceURL,
                        dataType: "JSONP"                    }
                },
      filter: { field: "ARTISTID", operator: "eq", value: e.data.
ARTISTID }
            },
            scrollable: false,
            sortable: false,
            columns: [
            { field: "ARTID", title: "Art ID"},
            { field: "ARTNAME", title: "Art Name"},
            { field: "DESCRIPTION", title: "Description"},
            { field: "PRICE", title: "Price", template: '#= kendo.
toString(PRICE,"c") #'},
            { field: "LARGEIMAGE", title: "Large Image"},
            { field: "MEDIAID", title: "Media ID"},
            { field: "ISSOLD", title: "Sold"}]
        });
    }
});
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

In this recipe, we define the `detailInit` parameter and pass to it the function that is also called `detailInit`. The `detailInit` function we created defines an inner Kendo Grid that binds to the `ARTISTID` column of the row of the outer Kendo Grid from which the `detailInit` function is called. We bind the inner grid to the outer grid using the `filter` parameter and by specifying the name of the column and passing the value of that particular column using `e.data`.

Working with remote data (Advanced)

This recipe demonstrates how to work with remote data such as cross-domain JSONP.

Getting ready

Open `12-remote.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 12- Working with Remote Data</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
$(document).ready(function () {
   var serviceURL = "http://gonautilus.com/kendogen/KENDO.
cfc?method=";
   var myDataSource = new kendo.data.DataSource({
   transport: {
      read:  {
```

```
            url: serviceURL + "getArt",
            dataType: "JSONP"
        }
    },
    pageSize: 20,
    schema: {
        model: {
            id: "ARTISTID",
            fields: {
                ARTID: { type: "number" },
                ARTISTID: { type: "number" },
                ARTNAME: { type: "string" },
                DESCRIPTION: { type: "CLOB" },
                PRICE: { type: "decimal" },
                LARGEIMAGE: { type: "string" },
                MEDIAID: { type: "number" },
                ISSOLD: { type: "boolean" }
                }
            }
        }
    } );
    $("#myGrid").kendoGrid({
        dataSource: myDataSource,
        pageable: true,
        sortable: true,
        columns: [
              { field: "ARTID", title: "Art ID"},
              { field: "ARTISTID", title: "Artist ID"},
              { field: "ARTNAME", title: "Art Name"},
              { field: "DESCRIPTION", title: "Description"},
              { field: "PRICE", title: "Price", template: '#= kendo.
toString(PRICE,"c") #'},
              { field: "LARGEIMAGE", title: "Large Image"},
              { field: "MEDIAID", title: "Media ID"},
              { field: "ISSOLD", title: "Sold"}]
        } );
    } );
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

This example shows you how to access a JSONP remote datasource. JSONP allows you to work with cross-domain remote datasources. The JSONP format is like JSON except it adds padding, which is what the "P" in JSONP stands for. The padding can be seen if you look at the result of the AJAX call being made by the Kendo Grid. It simply responds back with the callback argument that is passed and wraps the JSON in parentheses.

You'll notice that we created a `serviceURL` variable that points to the service we are calling to return our data. On line 19, you'll see that we are calling the `getArt` method and specifying the value of `dataType` as `JSONP`. Everything else should look familiar.

There's more...

Generally, the most common format used for remote data is JavaScript Object Notation (JSON). You'll find several examples of using ODATA on the Kendo UI demo website. You'll also find examples of performing create, update, and delete operations on that site. We will be using the demo services that Kendo UI has set up for our next few recipes as we explore user editing in the grid.

Outputting JSON with ASP MVC

In an ASP MVC or ASP.NET application, you'll want to set up your datasource like the following example. ASP has certain security requirements that force you to use `POST` instead of the default `GET` request when making AJAX calls. ASP also requires that you explicitly define the value of `contentType` as `application/json` when requesting JSON. By default, when you create a service as ASP MVC that has `JsonResultAction`, ASP will nest the JSON data in an element named `d`:

```
var dataSource = new kendo.data.DataSource({
    transport: {
        read: {
            type: "POST",
            url: serviceURL,
            dataType: "JSON",
            contentType: "application/json",
            data: serverData
        },
        parameterMap: function (data, operation) {
            return kendo.stringify(data);
        }
    },
    schema: {
        data: "d"
    }
});
```

Batch editing (Advanced)

This recipe demonstrates how to allow the user to make batch edits in a grid.

Getting ready

Open `13-batchedit.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 13- Batch Editing</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
$(document).ready(function () {
   var serviceURL = "http://demos.kendoui.com/service";
   var myDataSource = new kendo.data.DataSource({
      transport: {
         read:  {
            url: serviceURL + "/Products",
            dataType: "jsonp"
         },
         update: {
            url: serviceURL + "/Products/Update",
            dataType: "jsonp"
         },
         destroy: {
            url: serviceURL + "/Products/Destroy",
            dataType: "jsonp"
         },
```

```
            create: {
               url: serviceURL + "/Products/Create",
               dataType: "jsonp"
            },
            parameterMap: function(options, operation) {
               if (operation !== "read" && options.models) {
                  return {models: kendo.stringify(options.models)};
               }
            }
         },
         batch: true,
         pageSize: 10,
         schema: {
            model: {
               id: "ProductID",
               fields: {
                  ProductID: { editable: false, nullable: true },
                  ProductName: { validation: { required: true } },
                  UnitPrice: { type: "number", validation: { required:
true, min: 1} },
                  Discontinued: { type: "boolean" },
                  UnitsInStock: { type: "number", validation: { min: 0,
required: true } }
               }
            }
         }
      });
      $("#myGrid").kendoGrid({
         dataSource: myDataSource,
         navigatable: true,
         pageable: true,
         editable: true,
         toolbar: [ { name: "create", text: "Add Product" } , "save",
"cancel"],
         columns: [
            { field: "ProductName", title: "Product Name" },
{ field: "UnitPrice", title: "Unit Price", template: '#= kendo.
toString(UnitPrice,"c") #' },
            { field: "UnitsInStock", title: "Units In Stock" },
            { field: "Discontinued" },
            { command: "destroy", title: " ", width: 120 }]
      });
});
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

There are a couple of things to pay attention to in this recipe. First, we have defined four distinct transport calls in our datasource. Obviously, each of these represents our CRUD functions. Depending on the command that the user performs, Kendo will fire one of the four transport operations.

There is some magic going on in lines 34-38, where Kendo appends all of the proposed changes by the user in the URL via the `models` argument. On the server side, you'll want to perform some kind of URL decode of the data coming from the client. Another thing to remember when working with batch data is that you are getting back a list of arrays, so you'll need to loop through all of them to make sure you properly update all of the data that the user has submitted.

We also introduced validation in the datasource schema. So, instead of just identifying the type of each column, we can also specify `min`, `max`, `nullable`, `required`, and other types of validation. You'll notice in the grid column definition that we included a command column that allows the user to request a row of data to be deleted.

Lastly, it's worth noting that we added a parameter called `batch` to our datasource and a parameter called `editable` to our grid. Both of these parameters need to be included, because the default is `false`.

Inline editing (Advanced)

This recipe demonstrates how to allow the user to make inline edits in the grid.

Getting ready

Open `14-inlineedit.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
```

```
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 14 - Inline Editing</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
$(document).ready(function () {
   var serviceURL = "http://demos.kendoui.com/service";
   var myDataSource = new kendo.data.DataSource({
      transport: {
         read:  {
            url: serviceURL + "/Products",
            dataType: "jsonp"
         },
         update: {
            url: serviceURL + "/Products/Update",
            dataType: "jsonp"
         },
         destroy: {
            url: serviceURL + "/Products/Destroy",
            dataType: "jsonp"
         },
         create: {
            url: serviceURL + "/Products/Create",
            dataType: "jsonp"
         },
         parameterMap: function(options, operation) {
            if (operation !== "read" && options.models) {
               return {models: kendo.stringify(options.models)};
            }
         }
      },
      batch: true,
      pageSize: 10,
      schema: {
         model: {
            id: "ProductID",
            fields: {
```

```
                ProductID: { editable: false, nullable: true },
                ProductName: { validation: { required: true } },
                UnitPrice: { type: "number", validation: { required:
true, min: 1} },
                Discontinued: { type: "boolean" },
                UnitsInStock: { type: "number", validation: { min: 0,
required: true } }
             }
          }
       }
   });
   $("#myGrid").kendoGrid({
      dataSource: myDataSource,
      navigatable: true,
      pageable: true,
      editable: "inline",
      toolbar: [ { name: "create", text: "Add Product" } ],
      columns: [
         { field: "ProductName", title: "Product Name" },
{ field: "UnitPrice", title: "Unit Price", template: '#= kendo.
toString(UnitPrice,"c") #' },
         { field: "UnitsInStock", title: "Units In Stock" },
         { field: "Discontinued" },
         { command: ["edit", "destroy"], title: " ", width: 200
}]
   });
});
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

In this recipe, only the datasource differs from the previous example. Here we set the value of `editable` to `inline` rather than just `true`. We also removed the `save` and `cancel` button values from the toolbar; as you can see, they are not needed. With inline editing enabled, the user is restricted to editing one row at a time.

Working with user events (Advanced)

This recipe demonstrates how to work with user events in the grid.

Getting ready

Open `15-events.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
<script src="js/personData.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 15 - Working with User Events</h3>
<p><a href="index.html">Home</a></p>
<script type="text/javascript">
  $(document).ready(function() {
        var myDataSource = new kendo.data.DataSource({
         data: personData,
            schema: {
                model: {
                    fields: {
                        FirstName: { type: "string" },
                        LastName: { type: "string" },
                        Rank: { type: "string" },
                        DOB: { type: "date" },
                        PersonId: { type: "number" }
                    }
                }
            }
      } );
```

```
        $("#myGrid").kendoGrid({
         dataSource: myDataSource,
         change: showChange,
         selectable: "multiple cell",
         columns: [
               { field: "Name", title: "Full Name", width: "300px",
template: '#=FirstName# #=LastName#' },
            { field: "DOB", title: "Date of Birth", width: "300px" },
               { field: "Rank", title: "Rank", width: "300px" }
          ]
        });
   });
   function showChange(e) {
      var selected = $.map(this.select(), function(item) {
         return $(item).text();
      });
      alert("Selected: " + selected.length + " item(s), [" + selected.
join(", ") + "]");
   }
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

In this recipe, we define the `change` parameter in our grid and pass it the `showChange` function that we created. All the `showChange` function does is push an alert to the browser telling the user how many cells they have selected and what data those cells hold. There are many uses for this type of functionality. You could use it to just log what selections a user has made or you could create a pop up of an image related to the data in that row.

You may also have noticed that we added a parameter called `selectable` and passed it the value of `multiple cell`. This just makes this recipe more practical when a user clicks on individual and multiple cells.

There's more...

You also have the `dataBound` and `dataBinding` parameters available to fire when those events are triggered in the grid. All kinds of special validations and manipulations can be applied using these parameters.

Advanced API example (Advanced)

This recipe demonstrates how to get the most out of Kendo Grids by tapping directly into the Kendo API.

Getting ready

Open `16-api.html` in your editor or create a new document.

How to do it...

Copy the following code into your new document:

```
<!DOCTYPE html>
<html>
<head>
<title>Kendo UI Grid How-to</title>
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
common.min.css">
<link rel="stylesheet" type="text/css" href="kendo/styles/kendo.
default.min.css">
<script src="kendo/js/jquery.min.js"></script>
<script src="kendo/js/kendo.web.min.js"></script>
</head>
<body>
<h3 style="color:#4f90ea;">Exercise 16 - Advanced API Example</h3>
<p><a href="index.html">Home</a></p>
<button id="expandAll" href="#">Expand All</button><button
id="collapseAll" href="#">Collapse All</button>
<script type="text/javascript">
  $(document).ready(function() {
      $('#collapseAll').hide(); // MAKE SURE ONLY ONE BUTTON SHOWS ON
LOAD
        var outerServiceURL = "http://gonautilus.com/kendogen/KENDO.
cfc?method=getArtists";
      var innerServiceURL = "http://gonautilus.com/kendogen/KENDO.
cfc?method=getArt";
      var myDataSource = new kendo.data.DataSource({
         transport: {
            read: {
               url: outerServiceURL,
               dataType: "JSONP"
            }
         }
```

```
        });
        $("#myGrid").kendoGrid({
           dataSource: myDataSource,
           sortable: true,
           detailInit: detailInit,
           columns: [{ field: "ARTISTID", title: "Artist ID"},
                 { field: "FIRSTNAME", title: "First Name"},
                 { field: "LASTNAME", title: "Last Name"},
                 { field: "EMAIL", title: "Email"},
                 { field: "PHONE", title: "Phone Number"}]

        });
        function detailInit(e) {
           $("<div/>").appendTo(e.detailCell).kendoGrid({
              dataSource: {
                 transport: {
                    read: {
                       url: innerServiceURL,
                       dataType: "JSONP",
                    }
                 },
                 filter: { field: "ARTISTID", operator: "eq", value:
e.data.ARTISTID }
              },
              scrollable: false,
              sortable: false,
              columns: [
                 { field: "ARTID", title: "Art ID"},
                 { field: "ARTNAME", title: "Art Name"},
                 { field: "DESCRIPTION", title: "Description"},
                 { field: "PRICE", title: "Price", template: '#= kendo.
toString(PRICE,"c") #'},
                 { field: "LARGEIMAGE", title: "Large Image"},
                 { field: "MEDIAID", title: "Media ID"},
                 { field: "ISSOLD", title: "Sold"}]
           });
        }
        expandCollapseGrid('#myGrid');
     });
     function expandCollapseGrid(gridId) {
        $("#expandAll").click(function (event) {
           event.preventDefault();
           var grid = $(gridId).data("kendoGrid");
           grid.expandRow(grid.tbody.find("tr.k-master-row"));
```

```
            $("#expandAll").hide();
            $("#collapseAll").show();
        });
        $("#collapseAll").click(function (event) {
            event.preventDefault();
            var grid = $(gridId).data("kendoGrid");
            grid.collapseRow(grid.tbody.find("tr.k-master-row"));
            $("#expandAll").show();
            $("#collapseAll").hide();
        });
    }
</script>
<div id="myGrid"></div>
</body>
</html>
```

How it works...

In this recipe, we build on exercise 11 by adding the `expandAll` and `collapseAll` buttons. This can be especially helpful with a larger set of data that users may want to see all at once. Adding the HTML buttons to expand and collapse was easy; what we did next was create a function called `expandCollapseGrid` to make them functional. We used several API methods here, including `data`, `expandRow`, and `collapseRow`. Lastly, we used jQuery syntax to get at all of the particular rows contained in the `tbody` section of the grid to identify the elements we wanted to execute the methods on. The `expandCollapseGrid` function is built in a way that you should be able to reuse it on any Kendo Grid that has some type of grouping.

There's more...

There are hundreds of other examples of using API methods to control the Kendo Grid. The complete documentation can be found at `http://docs.kendoui.com/api/web/grid`. You can also filter your results in the textbox at the top of the page, which may not be apparent at first glance. Also, be sure to check out the documentation on the Kendo datasource, as that is just as much a part of the grid. Datasource documentation can be found at `http://docs.kendoui.com/api/framework/datasource`.

Thank you for buying
Instant Kendo UI Grid

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

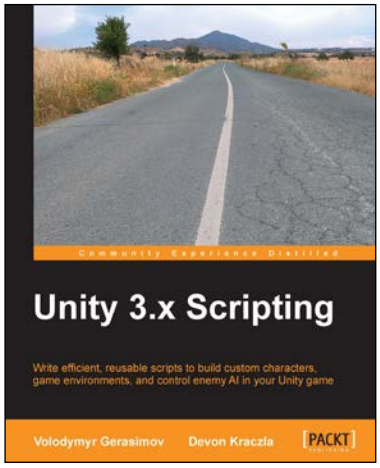
Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] PUBLISHING

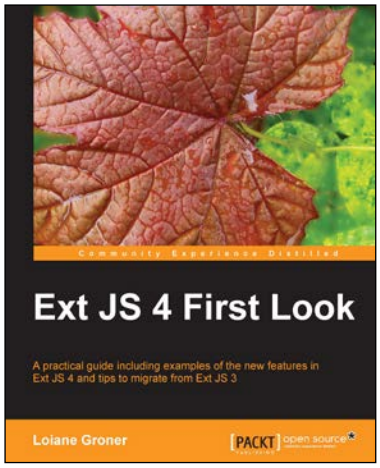


Unity 3.x Scripting

ISBN: 978-1-849692-30-4 Paperback: 292 pages

Write efficient, reusable scripts to build custom characters, game environments, and control enemy All in your Unity game

1. Make your characters interact with buttons and program triggered action sequences.
2. Create custom characters and code dynamic objects and players' interaction with them.
3. Synchronize movement of character and environmental objects.
4. Add and control animations to new and existing characters.



Ext JS 4 First Look

ISBN: 978-1-849516-66-2 Paperback: 340 pages

A practical guide including examples of the new features in Ext JS 4 and tips to migrate from Ext JS 3

1. Migrate your Ext JS 3 applications easily to Ext JS 4 based on the examples presented in this guide.
2. Full of diagrams, illustrations, and step-by-step instructions to develop real word applications.
3. Driven by examples and explanations of how things work.