# Getting Started

For more information on this topic, see: <http://docs.kendoui.com/getting-started/what-is-kendo>

Getting and adding KendoUI to your project.

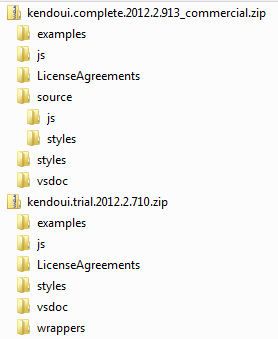
## Getting KendoUI

For more information on this topic, see: <http://docs.kendoui.com/getting-started/downloading-kendo>

KendoUI is divided into three parts: Web, Mobile, and DataViz.

The preferred method of getting KendoUI is to browse to <http://www.kendoui.com/download> and download KendoUI as a .zip file.

Depending on whether you are using the Trial version, or have purchased the full Commercial version, you will have a .zip file that follows one of these directory structures:



The \js and \styles folders in the root of the .zip file contain the “minified” versions of the KendoUI .js and .css files. If you have the commercial version of KendoUI, you may also notice the \source folder. This folder also contains \js and \styles folders, but these are the full, non-minified versions of the .js and .css. These would typically be used for local development and debugging, but the minified versions are the ones that should be deployed in a real application. For that reason, we will use the minified files in the remainder of this example project.

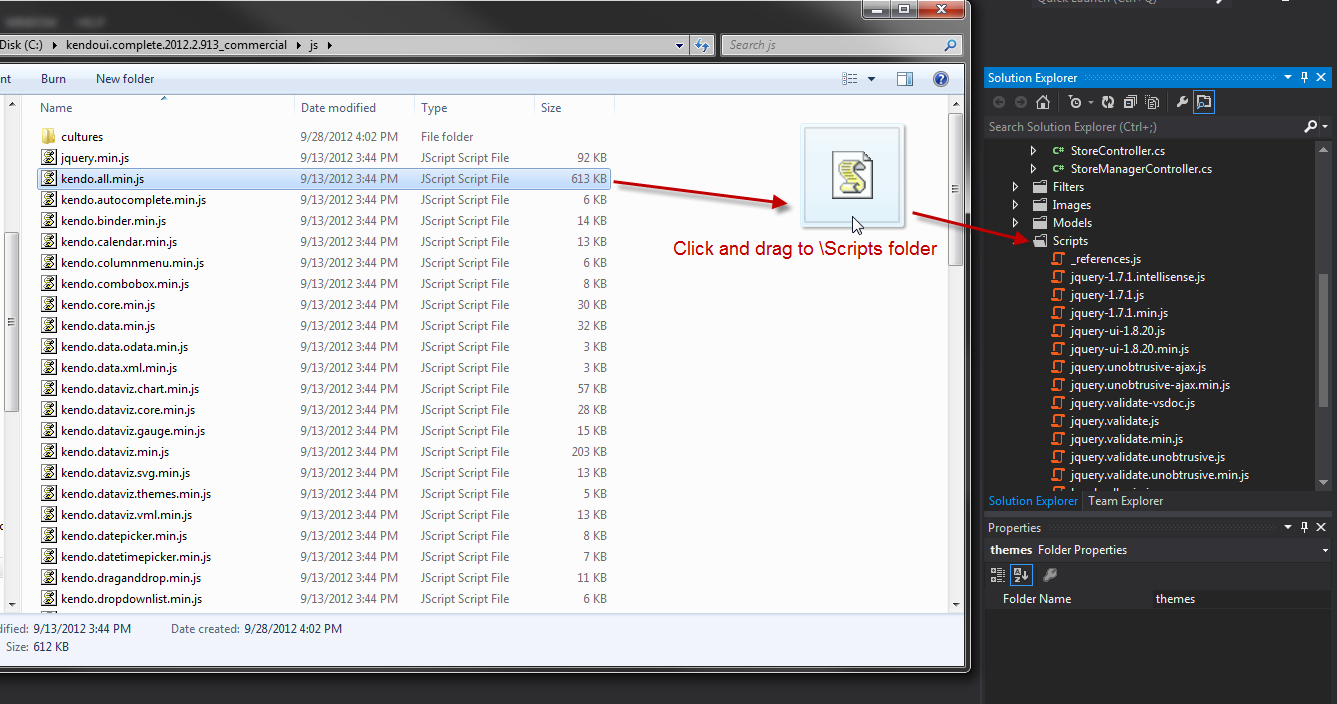
## Adding KendoUI to your ASP.NET MVC Project

For more information on this topic, see: <http://docs.kendoui.com/getting-started/using-kendo-with/aspnet-mvc/introduction>

To add KendoUI to your project, you just need to copy the files you need into your project. For a standard ASP.NET MVC project, files from the Kendo .zip \js folder are copied to ASP.NET MVC’s \Scripts folder, and files from the Kendo .zip \styles folder are copied to ASP.NET MVC’s \Content\themes\base folder.

If you know exactly which Kendo controls and styles you are going to use in your project, you can include individual JavaScript files for just those controls. This can provide a much smaller download for your end users. Optionally, you can use the single file “kendo.all.min.js” which contains the minified files for all of Web, Mobile, and DataViz. You do not need to add kendo.all.min.js *and* each individual kendo.\*.min.js file. Include just one or the other. For more information, see: <http://docs.kendoui.com/getting-started/javascript-dependencies>

When copying the files to your ASP.NET MVC project, you can simply drag and drop the files you want from Windows Explorer to the Solution Explorer in VisualStudio. This will automatically copy the files to your project’s directory, and add them to the project.



## VisualStudio Integration and Templates for ASP.NET MVC

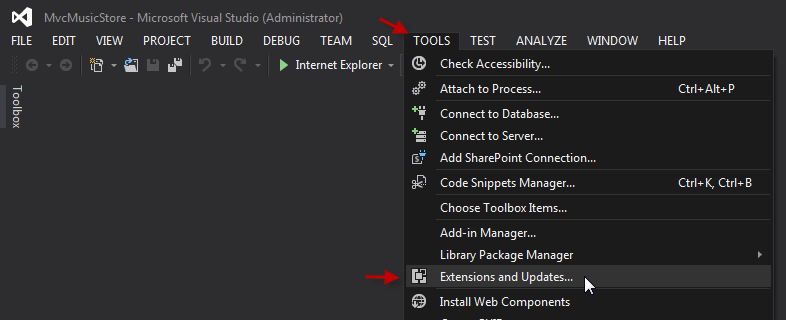
For more information on this topic, see: <http://docs.kendoui.com/getting-started/using-kendo-with/aspnet-mvc/introduction>

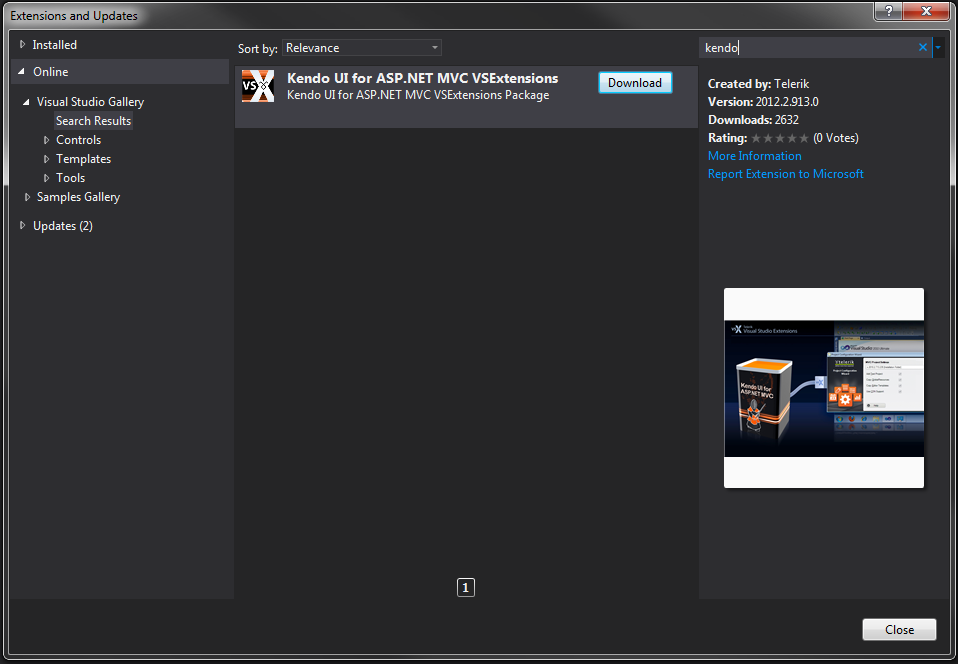
Optionally, Kendo UI provides Kendo UI Complete for ASP.NET MVC.

Kendo UI Complete for ASP.NET MVC is a set of HTML helpers which help you configure Kendo UI widgets by using server-side code in ASP.NET MVC applications. It also provides ready-to-use editor templates based on various Kendo UI widgets.

There is also a VisualStudio Extension available in the VisualStudio Gallery here: <http://visualstudiogallery.msdn.microsoft.com/65b78c2c-951e-43a8-bae7-f9039f59fb9b>

Or it can be added through VisualStudio by selecting “Tools | Extensions and Updates…” from the main menu, then searching for “kendo”.





If you would like to know more about this set of utilities, please refer to the above link.

## jQuery

Kendo is dependent on jQuery. When Kendo JavaScript files are included in an HTML page, jQuery must be included before Kendo.

# Creating a Consumer Web Site with KendoUI Web

## Include Kendo files in rendered Views

Add the CSS and JS to all rendered pages by adding entries to \Views\Shared\\_Layout.cshtml. We need to make sure that Kendo’s JavaScript files come after jQuery. For styling, we include kendo.common.min.css, plus one of the Kendo UI Themes. In this example we are using kendo.default.min.css, which is the Default theme.

<head>

<meta charset="utf-8" />

<title>@ViewBag.Title - Kendo UI Music Store</title>

<link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />

<meta name="viewport" content="width=device-width" />

@Styles.Render("~/Content/css")

<link href="~/Content/kendo.common.min.css" rel="stylesheet" /> // <-- Added Here.

<link href="~/Content/kendo.default.min.css" rel="stylesheet" /> // <-- Added Here.

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

...

@Scripts.Render("~/bundles/jquery")

<script src="~/Scripts/kendo.all.min.js"></script> // <-- Added Here.

@RenderSection("scripts", required: false)

</body>

It would also be good practice to use the MVC4 “bundler” capabilities to merge multiple Kendo JS files into a single file for client download. Since we are using the single kendo.all.min.js file in this example, we do not need to perform the bundling.

## Creating the main layout Header and Footer

The layout for our Music Store is contained in \Views\Shared\\_Layout.cshtml

The desired layout for our header is:



The two elements to note here are the menu and search box. For these elements, we will be using the Kendo [Menu](http://demos.kendoui.com/web/menu/index.html) and [AutoComplete](http://demos.kendoui.com/web/autocomplete/index.html) controls.

The basic markup for this header is:

<header>

<section>

@Html.Partial("\_LoginPartial")

</section>

<section>

<nav>

<ul id="menu">

</ul>

</nav>

<button />

<input type="text" />

<section>

</header>

### Add the main menu

Let’s use the Kendo Menu control to add a navigation menu to the Music Store.

To start, we can define the menu items as a <ul> with HTML and Razor:

<ul id="menu">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>Genres

@{Html.RenderAction("GenreMenu", "Store");}

</li>

<li>About</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul>

Converting this to a Kendo menu is easy. We just add the JavaScript:

$(document).ready(function () {

$("#menu").kendoMenu();

});

This uses a jQuery selector to get our <ul> tag with the id “menu,” and turns it into a Kendo UI menu. This type of kendo usage is sometimes called “explicit” widget initialization, since the element is directly targeted in JavaScript (We will see examples of implicit initialization, data-\* attributes and the MVVM pattern later on, but for our first control, let’s keep it simple.)

Next we can move the Genres list into a pull-down menu. Any <li> elements that in turn contain another list <ul> element are treated as a pull-down item. Let’s start with an empty pull-down menu item for our Genres:

<li>Genres

<ul>

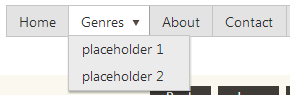
<li>placeholder 1</li>

<li>placeholder 2</li>

</ul>

</li>

This creates the pull-down item:



We can replace our “placeholder” <li> elements with an MVC partial that loads the list of Genres:

<li>Genres

<ul>

@{Html.RenderAction("GenreMenu", "Store");}

</ul>

</li>

This partial outputs a list of <li> elements that work as links to each Genre:

@model IEnumerable<MvcMusicStore.Models.Genre>

<ul>

@foreach (var genre in Model)

{

<li>@Html.ActionLink(genre.Name,

"Browse", "Store",

new { Genre = genre.Name }, null)

</li>

}

</ul>

Another ability of the Kendo Menu control is adding any content into a drop-down menu item. It does not have to be a list of selectable items, like the Genres. To add content to a menu item, Include a <ul> element, with a single <li> element. Within that element, use a <div> to specify the content. For example, let’s add this content for out “About” menu item:

<li>About

<ul>

<li>

<div class="k-content menu-content">

<img src="~/Images/kendo-logo.png" />

<p>Try Kendo UI yourself in the interactive Kendo UI Dojo!</p>

<p><a href="http://www.kendoui.com/dojo.aspx" target="\_blank">Launch Kendo UI Dojo</a></p>

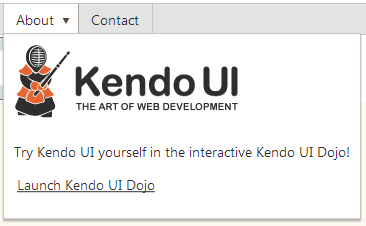
</div>

</li>

</ul>

</li>

This creates the <div> content displayed in a drop-down menu item:



### Adding the Search AutoComplete box.

Our site header contains a text box that can be used to search for artists and albums. We are going to use a Kendo AutoComplete control to display a list of possible matches to the user as they type.

This will also bind to a [remote data source](http://demos.kendoui.com/web/datasource/remote-data.html) to retrieve the auto-complete items.

We start with a normal <input> element, and initialize it with Kendo:

<input id="main-search"/>

$("#main-search").kendoAutoComplete({

filter: 'contains',

minLength: 3,

dataTextField: "Title",

placeholder: "Search music...",

dataSource: {

type: "odata",

serverFiltering: true,

serverPaging: true,

pageSize: 20,

transport: {

read: {

url: "/Api/Albums",

dataType: "json"

},

parameterMap: function (options, type) {

var paramMap = kendo.data.transports.odata.parameterMap(options);

delete paramMap.$inlinecount;

delete paramMap.$format;

return paramMap;

}

},

schema: {

data: function (data) {

return data;

},

total: function (data) {

return data.length;

}

}

},

template: "<img src='${data.AlbumArtUrl}' /><span>${data.Title}</span><span>${data.Artist.Name}</span>"

});

This JavaScript configures the input box to bind to a remote data source. Some of the notable parameters are:

*filter: ‘contains’* – Specifies that autocomplete results just need to contain the entered text, anywhere. The default is ‘starts with’.

*minLength: 3* – Indicates that the autocomplete dropdown will not display until the user has entered at least 3 characters.

*dataTextField: ‘Title’* – The name of the field in the data that is searched to find autocomplete results. In this example, we are showing autocomplete results for albums whose titles contain the text entered by the user.

*placeholder: “Search music…” –* This is the text thatis displayed in the text box as a placeholder, until the user clicks into the input box.

*dataSource: {}* – Here we configure the source for our autocomplete data. We have specified the URL to our WebAPI for Albums as the source. There is a lot going on in this data source, but most of it is enabling server-side filtering using OData.

*Template: “…”* – Each autocomplete item in the dropdown will be a <li> element, and within that element, include this template. If not specified, Kendo will put the text of the field specified by the *dataTextField* property into the <li>. We can use this template to make much nicer looking results. In this case, we are including the album cover art.

[TODO autocomplete screenshot]

## Creating the Home page.

For the main content of the “Home” page, we want to display a main banner image rotator, and 2 groups of Albums. One will be our “top sellers,” and the other is a “featured artist.” For this, we will be using the Kendo [ListView](http://demos.kendoui.com/web/listview/index.html) widget. For this view, we will also use a “declarative” approach to initializing the widgets, using the data-\* attributes, and the [Kendo MVVM](http://demos.kendoui.com/web/mvvm/index.html) framework.

### Displaying items in a ListView (with MVVM, templates, and binding to a remote data source)

To implement our lists of “featured artist” and “top selling” albums, we can start with some simple HTML markup to represent the 2 ListView widgets:

<section>

<h3>Featured Artist: <span data-bind="text: featuredArtistName"></span></h3>

<div data-role="listview" data-bind="source: featuredArtistAlbums" data-template="album-template">

</div>

</section>

<section>

<h3>Top Selling Albums:</h3>

<div data-role="listview" data-bind="source: topSellingAlbums" data-template="album-template">

</div>

</section>

We have set the “data-role” of each <div> to “listview” so that Kendo knows to transform these <div>s into ListView widgets. Each album will look the same between the 2 lists, so we chose to share a template. Both <div>s get the attribute “data-template” set to “album-template”. In addition, we also need to add the template itself:

<script id="album-template" type="text/x-kendo-template">

<div class="album">

<img src="#: AlbumArtUrl #" />

<span>#: Title #</span>

<span>#: Artist.Name #</span>

<span>#: Price #</span>

</div>

</script>

Note the special “type” of “text/x-kendo-template”. This is required for Kendo to be able to resolve the template. Also the “id” matches the template indicated by the <div> tags. More information on templates can be found [here](http://demos.kendoui.com/web/templates/index.html).

Finally, the “data-bind” property contains our list of binders that will be applied to this widget. In this case we are using the “source” binder, and binding it to the “featuredArtistAlbums” property of the view model. This means we need to create our view model in JavaScript, which is:

$(document).ready(function () {

var featuredArtist = "Metallica";

var viewModel = kendo.observable({

featuredArtistName: featuredArtist,

featuredArtistAlbums: new kendo.data.DataSource({

type: "odata",

serverFiltering: true,

serverPaging: true,

pageSize: 5,

filter: {

field: "Artist/Name",

operator: "eq",

value: featuredArtist

},

transport: {

read: {

url: "/Api/Albums",

dataType: "json"

},

parameterMap: function (options, type) {

var paramMap = kendo.data.transports.odata.parameterMap(options);

delete paramMap.$inlinecount;

delete paramMap.$format;

return paramMap;

}

},

schema: {

data: function (data) {

return data;

},

total: function (data) {

return data.length;

}

}

}),

topSellingAlbums: new kendo.data.DataSource({

transport: {

read: "/Api/Albums?popular=5"

}

})

});

kendo.bind("#body", viewModel);

});

Our viewModel is a Kendo [Observable](http://docs.kendoui.com/api/framework/observableobject) object, which facilitates the updating of properties and notifying the view when it needs to be redrawn. The last line of JavaScript tells kendo to apply bindings between the viewModel and the HTML element with the ID “body”, using normal jQuery selector syntax. In our view model, we have set “Metallica” as our featured artist. This text string will be shown in our <h3> header element, due to the data-binding:

<h3>Featured Artist: <span data-bind="text: featuredArtistName"></span></h3>

The featuredArtistAlbums and topSellingAlbums properties of the view model are both set to Kendo [DataSources](http://demos.kendoui.com/web/datasource/remote-data.html) that will pull remote data from our MVC4 WebAPI controllers. In the case of the featuredArtistAlbums data source, there is additional code specified that enabled server-side filtering by artist name.

### Image Rotator (with a custom binding)

The final item we need to add to your main page is an image rotator. Kendo does not actually provide an image rotator widget, however, the Kendo MVVM bindings are extendable, and so we will write our own [custom binding](http://demos.kendoui.com/web/mvvm/custom.html) to handle this.

We can start by setting up a property on our view model to hold the URLs for the images we want to rotate:

var viewModel = kendo.observable({

bannerImages: [

"/Content/Images/banner1.jpg",

"/Content/Images/banner2.jpg"

]

The bannerImages property is just a simple array of image URLs. Next we make our custom binding:

kendo.data.binders.rotateImages = kendo.data.Binder.extend({

init: function (element, bindings, options) {

kendo.data.Binder.fn.init.call(this, element, bindings, options);

var binding = this.bindings["rotateImages"];

var target = $(element);

binding.rotateDelay = target.data("rotate-delay");

binding.imageIndex = 0;

binding.setRotationTimeout = function () {

var imageArray = binding.get();

target.attr("src", imageArray[binding.imageIndex]);

binding.imageIndex = binding.imageIndex + 1;

if (binding.imageIndex >= imageArray.length) {

binding.imageIndex = 0;

}

setTimeout(binding.setRotationTimeout, binding.rotateDelay);

};

binding.setRotationTimeout();

},

refresh: function () {

var binding = this.bindings["rotateImages"];

binding.imageIndex = 0;

},

});

Functionally, this is a fairly simple binding. All it does is use a JavaScript setTimeout() to periodically change the “src” attribute of an <img> element. However, it does demonstrate a way to retain data in the binding by adding properties to the object instance, and a way to pass in additional data, in this case the delay time. We use a second data-\* property on the element, “data-rotate-delay”, and the custom binding can read this data property with jQuery: $(element).data(“rotate-delay”).

Now we can make our HTML that uses our custom binding:

<img src="/Content/Images/home-showcase.png" data-bind="rotateImages: bannerImages" data-rotate-delay="5000" />

We have indicated that this <img> element will be bound to our custom “rotateImages” binding, using the “bannerImages” property of the view model as its data. Through use of the additional data-\* attribute, we have also indicated that our image will update every 5000 milliseconds.

# Creating an Administrative Panel with KendoUI Kendo Grid and DataViz Charts

# Creating a Consumer Mobile Site KendoUI Mobile