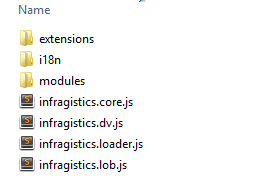
## Creating the Project

The first step is to create a new MVC4 project in VS2012. For the purposes of this example we will use the Basic template and the Razor view engine.

Once the project is created create a new IG folder under both the Scripts folder and the Content folder in the project. Now we need to copy in the local JavaScript and CSS files which can be found in the install folder in the js and css folders respectively.

## JavaScript File Structure

In the js folder you’ll find three folders and several js files:



How you use these depends on your needs. The infragistics.loader.js file will load in resources for you without you having to reference any other Infragistics JavaScript files. You will have to make sure that you include the i18n and modules folders though as these contain the JavaScript files for the controls that the loader will need to reference.

The infragistics.core, dv, and lob files are new combined files that replace the infragistics.js from previous versions of Ignite UI. These files combine several controls in to one file to minimize the amount of downloads that will be required. If you include these files you do not require the modules folder.

The extensions folder contains JavaScript files that are used to support other libraries. At this time the only files that reside in the folder are for the support of Knockout. The i18n folder contains files that support internationalization and are only needed if you require support for one of the provided cultures.

The rest of this example will follow the approach of using the igLoader.

## CSS File Structure

In the css folder you’ll fine two folders, themes and structure. Structure is used to setup the general layout needed for the controls. Themes provides special styling to apply a consistent look and feel across your application. Infragistics provides two themes by default: metro and Infragistics. You only need to copy in one theme.

## Adding the Infragistics.Web.Mvc Assembly

Now we need to add a reference to the Infragistics.Web.Mvc assembly. This is needed to be able to use the MVC wrappers for the Infragistics controls. You can find this assembly in the install folder in the MVC4\Bin folder. This file should be copied to the bin folder of your Project. Once this is done, right click the references node and add the assembly from its location in the bin. Then, right click the Infragistics.Web.Mvc node under reference and check the properties. Make sure that Copy Local is set to true.

## Setting Up a Layout Page

Open up the Views\Shared\\_Layout.cshtml file. Add the following line to include a reference to jQuery UI:

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

This should be placed after the line to include the jquery script bundle (line 13 by default). After this, add a reference to Infragistics.loader.js.

Add the following line to create a content section for the page:

@RenderSection("MainContent", required: false)

This should be placed after the scripts section. This step isn’t strictly necessary but it helps for organization of the page and makes sure that items can be added to the page in the correct order. Once you are done the \_Layout.cshtml file should appear similar to the following:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title</title>

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

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

</head>

<body>

@RenderBody()

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

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

<script src="@Url.Content("~/Scripts/IG/infragistics.loader.js")" type="text/javascript" ></script>

@RenderSection("scripts", required: false)

@RenderSection("MainContent", required: false)

</body>

</html>

## Creating the Model

Add a new class to the Models folder. In this example I am calling it TestModel. I use the following code:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace igGridLayoutMVC4Starter.Models

{

public class TestModel

{

public List<TestItem> Data { get; set; }

public TestModel()

{

Data = new List<TestItem>();

for (int i = 0; i < 5; i++)

{

Data.Add(new TestItem { ID = i, Name = "Item " + i, ItemDate = DateTime.Today.AddDays(-i) });

}

}

}

public class TestItem

{

public int ID { get; set; }

public string Name { get; set; }

public DateTime ItemDate { get; set; }

}

}

## Create the Controller

Add a new Controller to the Controllers folder called HomeController. Leave the template as Empty MVC controller.

At the top of the code file for the HomeController add a reference to the Models namespace. In the Index ActionResult return Data as IQueryable:

public class HomeController : Controller

{

public ActionResult Index()

{

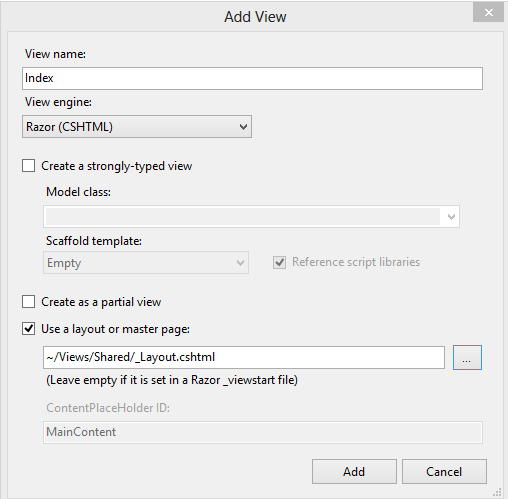
TestModel model = new TestModel();

return View(model.Data.AsQueryable());

}

}

Right click anywhere in the code for Index and choose Add View. Leave Use a layout or master page checked and choose the \_Layout.cshtml file we created above. Click on the Add button:



## Create the View

At the top of your view add a reference to the Infragistics.Web.Mvc namespace. After this, set up the model for the page as IQueryable<TestItem>:

@using Infragistics.Web.Mvc

@model IQueryable<igGridLayoutMVC4Starter.Models.TestItem>

Add a section called scripts. In here, add the igLoader and set its ScriptPath and CssPath to the IG folders we added above. Make sure that you call Render at the end to have the control created properly in the View. Since the LayoutManager is a CTP control we also need to explicitly tell the Loader to load the resources for this control. Normally when using the MVC wrapper the Resources for controls do not need to be included:

@section scripts{

@(Html.Infragistics().Loader()

.ScriptPath(Url.Content("~/Scripts/IG/"))

.CssPath(Url.Content("~/Content/IG/"))

.Resources("igLayoutManager")

.Render()

)

}

Next, add another section called MainContent. In here create the structure for the layout. For the purpose of the example we’ll have a div called layout. This has a nav section, a header area, and a center content area. In the content area, create an instace of the Grid bound to the Model data. For the sake of simplicity no features have been turned on for the grid:

@section MainContent{

<div id="layout">

<nav class="left">

<h1>Navigation here</h1>

</nav>

<div class="header">

<h1>Header goes here</h1>

</div>

<div class="center">

@(Html.Infragistics().Grid(Model)

.Width("100%").Height("100%")

.DataBind()

.Render()

)

</div>

</div>

}

Now, we have to set up the LayoutManager. At this time the LayoutManager is not included in the MVC wrapper so we have to instantiate everything through JavaScript.

First, add a script tag to the scripts section after the previous code we added for the Loader. In here, handle the ig.loader ready event, create the igLayoutManger, and set its leftWidth to 10% and rightWidth to 0%:

<script type="text/javascript">

$.ig.loader(function () {

$('#layout').igLayoutManager({

layoutMode: 'border',

borderLayout: {

leftWidth: "10%",

rightWidth: "0%"

}

});

});

</script>

If you want to style the areas of the LayoutManager then you may also want to add those styles to the scripts section:

<style type="text/css">

div#layout {

height: 300px;

}

div#layout .left {

background-color: #FFA72D;

color: #FFF;

height: 100%;

}

div#layout .header {

background-color: #2CBDF9;

color: #FFF;

}

</style>

## Finishing Up

At this point we should be able to compile and run our application. If everything is set up correctly you should see something like the following:

