**Highsoft Internal Documentation**

[Overview](#h.uisam5mkefts)

[CodeGenerator](#h.3el8jd5u0iy9)

[CodeAddOns](#h.ice8omfsj0ff)

[CodeGeneration](#h.l9y6asvejvdc)

[CodeTemplates](#h.g1fduj1ieqz0)

[Additional Relevant files](#h.qpwedu50giol)

[Generating Enums](#h.uqjgp7ep0pc)

[Generating Properties](#h.a6h8aqfu1dwl)

[Generating Classes](#h.zfqup5v8bwwu)

[Highcharts Class Library MVC\_Highcharts](#h.54h6j1azqjwz)

[CodeAddOns](#h.3jml7oks9xud)

[CodeGeneration](#h.exnu5u5j9m32)

[CompileOn.cs](#h.7ku0kra7iuf)

[Highcharts.cs](#h.87e74qmu2e6k)

[HighchartsRendering.cs](#h.mwotnhnn3q8h)

[copygeneratedfiles.bat](#h.apc3e8ropl3x)

[HighstockClass Library MVC\_Highstock](#h.4u6u8wwia9bd)

[CodeAddOns](#h.vqneyfghp73q)

[CodeGeneration](#h.axnlyv3ylb28)

[CompileOn.cs](#h.tt39uopgsmlf)

[Highstock.cs](#h.s2r3fue8t4en)

[HighstockRendering.cs](#h.xa0mxv2cih7s)

[copygeneratedfiles.bat](#h.oj3b6j4de77r)

[MVC\_Demo](#h.a8zcxdbiqamg)

[References](#h.hd287piyqtpn)

[Areas](#h.zdhmapl0xr1q)

[Controllers](#h.ipdb8az868rt)

[Views](#h.iebec9ikamg3)

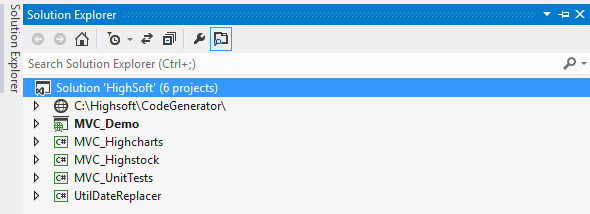
[Content](#h.j9ntfyyv2jtu)

[Downloads](#h.9x3q9ayzgyo7)

[MVC\_UnitTests](#h.6d746l1z2ou7)

# Overview

The Highsoft ASP.NET MVC solution currently consists of 6 projects:



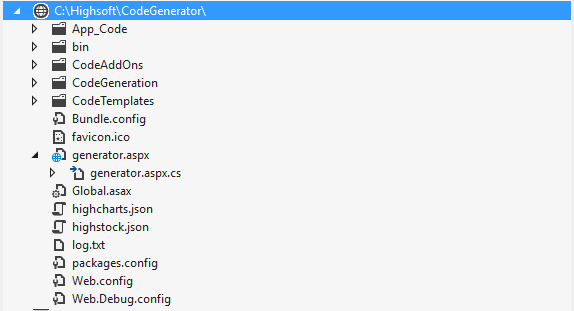
Each project with brief description

1. **“CodeGenerator”** is a web-application the contains the actual code generator. It reads the JSON documentation file (of Highcharts, Highstock, etc) and generates the ASP.NET MVC code of the library. The reason it is a web-site is that it was originally developed as a demo / test project for Highsoft and needed to be shown online. In the future it can be changed to a Console Application
2. **MVC\_Demo** - this is the actual demo projects that shows the nearly ~80 demos of the product and can be currently seen here- <http://highchartsaspnetmvc.azurewebsites.net/highcharts/demo>. It servers two puprposes - it is the online demo of the product which customers try online and the project itself when zipped is what is actually shipped to the customer as a Trial Build.
3. **MVC\_Highcharts** - this library project is the actual generated C# code from the CodeGenerator. However it also contains additional code that cannot be autogenerated, for example MVC stub code of the library, add-ons, databinding and general ASP.NET logic that cannot be inferred from the JSON file.
4. **MVC\_Highstock**, same as MVC\_Highcharts, however for the Highstock product.
5. **MVC\_UnitTest** - a set of unit tests for the MVC\_Highcharts and MVC\_Highstock projects - general library logic, JSON options generation, etc
6. **UtilDateReplacer** - a helper project that is used in the build process to generate a dummy file with the current time (timestamp). This is used by the trial products to check when the trial expires.

A more detailed information on each project follows

# CodeGenerator

The CodeGenerator project has the following structure:

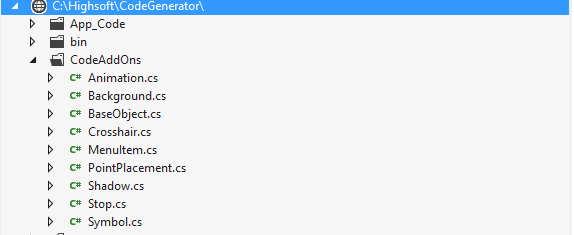


All logic related to converting the JSON help file to code is located in the generator.aspx.cs file. This may later be changed to a more complex structure as we include other products (Stock, Maps) and platforms (e.g. WebForms)

The important folders are

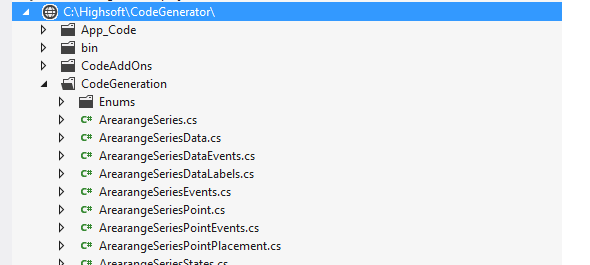
## CodeAddOns

C# source code files that are customly written (not autogenerated) for properties and types that do not have a direct match in the JSON file to C# types, for objects that are not fully documented in the JSON file (e.g. Animation has several properties that are documented in the description but not as actual separate items in the JSON, as well as some auxiliary file as BaseObject needed for serialization



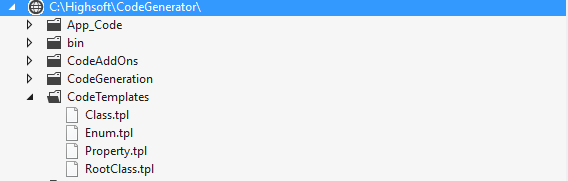
## CodeGeneration

The CodeGeneration folder contains all the automatically generated files from the JSON file to actual C# code. The enumerations are separated in a subfolder **Enums**. Each object from the JSON file has its own C# match entry with its respective name using the JSON entry **fullname** attribute



## CodeTemplates

The CodeTemplates folder contains several templates for generating code. They include Enum.tpl, Class.tpl, Property.tpl, etc (others may be added as needed and additional subfolders may be introduced later on for different platforms)



The template commands how the JSON item will be persisted as a file, for example the property template contains the following:

*/// <summary>*

*/// {HighTemplate.Comment}*

*/// </summary>*

*public {HighTemplate.Type} {HighTemplate.Name} { get; set; }*

*private {HighTemplate.Type} {HighTemplate.Name}\_DefaultValue { get; set; }*

The variables in curly braces {} are placeholders which the **CodeGenerator** replaces with the values for the respective JSON file entry, e.g. the **{HighTemplate.Comment}** is taken from the JSON entry description property, where as **{HighTemplate.Type}** is calculated (e.g. Number is mapped to Int or Float, etc)

## Additional Relevant files

The highcharts.json and highstock.json files are the actual JSON file the help uses to generate its help UI and are taken directly from the Highcharts website



## Generating Enums

A typical entry from the json files is:

*{"name":"series<area>--step","fullname":"series<area>.step","title":"step","parent":"series<area>","isParent":false,"returnType":"String","defaults":"false","values":"[\"left\", \"center\", \"right\"]","since":"1.2.5","description":"Whether to apply steps to the line. Possible values are <code>left</code>, <code>center</code> and <code>right</code>. Prior to 2.3.5, only <code>left</code> was supported.","demo":"<a href=\"http://jsfiddle.net/gh/get/jquery/1.7.2/highslide-software/highcharts.com/tree/master/samples/highcharts/plotoptions/line-step/\" target=\"\_blank\">Different step line options</a>","deprecated":false},*

The CodeGenerator undetstands that this is an enumeration (since it has multiple “values”) and uses the Enum.tpl template to generate the following C# source file:

*using System;*

*using System.Collections.Generic;*

*using System.Linq;*

*using System.Text;*

*namespace Highsoft.Web.Mvc.Charts*

*{*

*/// <summary>*

*/// Whether to apply steps to the line. Possible values are <code>left</code>, <code>center</code> and <code>right</code>. Prior to 2.3.5, only <code>left</code> was supported.","demo":"<a href=\"http://jsfiddle.net/gh/get/jquery/1.7.2/highslide-software/highcharts.com/tree/master/samples/highcharts/plotoptions/line-step/\" target=\"\_blank\">Different step line options</a>*

*/// </summary>*

*public enum AreaSeriesStep*

*{*

*False,*

*Left,*

*Center,*

*Right*

*}*

*}*

Depending ot the type of the JSON item (enum, class, property) the respective template file is used. The logic for generating classes and properties is of course more complex than enums

## Generating Properties

The template for properties is:

/// <summary>

/// {HighTemplate.Comment}

/// </summary>

public {HighTemplate.Type} {HighTemplate.Name} { get; set; }

private {HighTemplate.Type} {HighTemplate.Name}\_DefaultValue { get; set; }

{HighTemplate.Comment} comes directly from the “description” property of the JSON file. It will be used in the Visual Studio autocomplete for reference (will show as the user types in the property using autocomple) and in the Help generation for the online help.

{HighTemplate.Type} comes from the “type” property of the JSON file, however CodeGenerator performs mapping between types, e.g. string => String, number => Int or Float, etc. Some types that lack direct matches (e.g. Object | String) are customly mapped in CodeGenerator. In addition, if a type is a complex type (has children) it is deemed an object and the type becomes the name of the object (using the fullname property of the JSON entry - e.g. fullname=”series<area>.plotLines” becomes the complex object AreaSeriesPlotLines.

{HighTemplate.Name} - The name of the property maps the JSON fullname to actual property name in the C# files.

One thing to note - we define two properties - one public (the property itself) and one private (the default values of the property). This will later on be used to determine if the property is different from its default values - if this is the case, the property is persisted in the JSON options. This is needed so that we make sure only the necessary options are persisted.

## Generating Classes

The class template is contained in the Class.tpl file (too long to be pasted here). A class is introduced for each JSON entry that has children (isParent = true). It contains a list of all its properties, as well as three default methods:

ToHashtable() - for each property in the class, if the current value is different from the default, it is added to a Hashtable, which is then used for JSON serialization

IsDirty() - return a boolean if the class needs to be serialized to JSON (if ToHashtable() has elements)

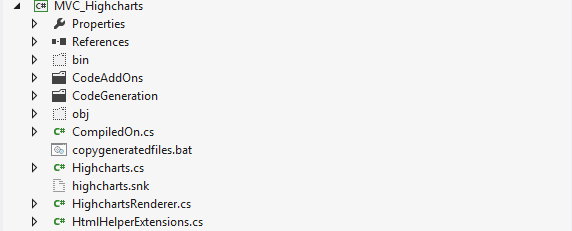
ToJSON() - uses the ToHashtable() and IsDirty() to return the actual JSON string of the options of the class serialized. If IsDirty() is false, to return an empty string

A class may consist of enums, properties and other classes as well, each of the nested classes has the ToHashtable(), IsDirty() and ToJSON() methods, thus representing the actual hierachy from the JSON help file.

# Highcharts Class Library MVC\_Highcharts

The **MVC\_Highcharts** library is the actual ASP.NET MVC library that is generated.by the **CodeGenerator** project. This is a standalone library the produces the DLL that is the product itself - **Highcharts.Web.Mvc.dll**. The DLL is then used by the customer by referencing it in their own project.

The current structure of the MVC\_Highcharts project is the following:



The important folders are:

## CodeAddOns

This is the very same folder from the CodeGenerator project (see the reference for it in the CodeGennerator / CodeAddOns section). It is needed to define the extra C# definitions for files that cannot be generated by the JSON file.

## CodeGeneration

This is the very same folde from the CodeGenerator project ((see the reference for it in the CodeGennerator / CodeAddOns section). These are the actual generated files by the CodeGenerator project.

## CompileOn.cs

This is a static file that is needed for the trial expiration of the library (time-bomb). It is static in this project and contains only this:

*using System.Collections.Generic;*

*using System.Linq;*

*using System.Text;*

*namespace Highsoft.Web.Mvc*

*{*

*internal static class CompiledOn*

*{*

*internal static DateTime CompilationDate = new DateTime(2099, 1, 1);*

*}*

*}*

The 2099 set as year means that in the static compilation, the library never expires. However in the build process (explained later), this is changed to the current date, e.g. Oct, 6th, 2015 and then used to calculate when the trial expires (currently set to 30-days)

## Highcharts.cs

This is a partial file that cotains just a few internal functions needed for the correct functioning of the product. The actual auto-generated Highcharts.cs file is in the CodeGeneration folder.

## HighchartsRendering.cs

Used for checking the trial expiration, e.g.

*#if (LICENSED == false)*

*if (DateTime.Now > CompiledOn.CompilationDate.AddDays(30))*

*{*

*return "This is a trial version of Highcharts for ASP.NET MVC which has expired.<br> Please, contact sales@highcharts.com for purchasing the product or for trial extension.";*

*}*

*#endif*

and also usd for the actual HTML rendering of the chart (javascript, <div> container, etc)

## copygeneratedfiles.bat

The purpose of this .BAT file is to automate the copying of the generated files from the CodeGenerator project to the MVC\_Highcharts project. Current definition is:

*rmdir /s /q CodeAddOns*

*rmdir /s /q CodeGeneration\Enums*

*rmdir /s /q CodeGeneration*

*echo f | xcopy /f /y ..\CodeGenerator\CodeAddOns CodeAddOns\*

*echo f | xcopy /f /y ..\CodeGenerator\CodeGeneration\Highcharts CodeGeneration\*

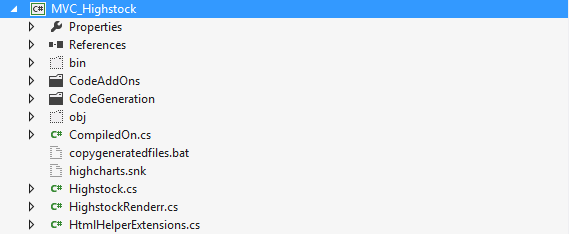
*echo f | xcopy /f /y ..\CodeGenerator\CodeGeneration\Highcharts\Enums CodeGeneration\Enums\*

Whenever the CodeGenerator project is run and files are generated, the copygeneratedfiles.bat files needs to be run to copy the generated files to the MVC\_Highcharts project, and then the project must be recompiled. After that the build files must be run to generate the deliverables (.ZIP files)

# HighstockClass Library MVC\_Highstock

The **MVC\_Highstock** library is the actual ASP.NET MVC Highstocl library that is generated.by the **CodeGenerator** project when the highstock.json file is used. This is a standalone library the produces the DLL that is the product itself - **Highstock.Web.Mvc.dll**. The DLL is then used by the customer by referencing it in their own project.

The current structure of the MVC\_Highstock project is the following:



The important folders are:

## CodeAddOns

This is the very same folder from the CodeGenerator project (see the reference for it in the CodeGennerator / CodeAddOns section). It is needed to define the extra C# definitions for files that cannot be generated by the JSON file.

## CodeGeneration

This is the very same folde from the CodeGenerator project (see the reference for it in the CodeGennerator / CodeAddOns section). These are the actual generated files by the CodeGenerator project.

## CompileOn.cs

This is a static file that is needed for the trial expiration of the library (time-bomb). It is static in this project and contains only this:

*using System.Collections.Generic;*

*using System.Linq;*

*using System.Text;*

*namespace Highsoft.Web.Mvc*

*{*

*internal static class CompiledOn*

*{*

*internal static DateTime CompilationDate = new DateTime(2099, 1, 1);*

*}*

*}*

The 2099 set as year means that in the static compilation, the library never expires. However in the build process (explained later), this is changed to the current date, e.g. Oct, 6th, 2015 and then used to calculate when the trial expires (currently set to 30-days)

## Highstock.cs

This is a partial file that contains just a few internal functions needed for the correct functioning of the product. The actual auto-generated Highstock.cs file is in the CodeGeneration folder.

## HighstockRendering.cs

Used for checking the trial expiration, e.g.

*#if (LICENSED == false)*

*if (DateTime.Now > CompiledOn.CompilationDate.AddDays(30))*

*{*

*return "This is a trial version of Highstock for ASP.NET MVC which has expired.<br> Please, contact sales@highcharts.com for purchasing the product or for trial extension.";*

*}*

*#endif*

and also used for the actual HTML rendering of the chart (javascript, <div> container, etc)

## copygeneratedfiles.bat

The purpose of this .BAT file is to automate the copying of the generated files from the CodeGenerator project to the MVC\_Highcharts project. Current definition is:

*rmdir /s /q CodeGeneration\Enums*

*rmdir /s /q CodeGeneration*

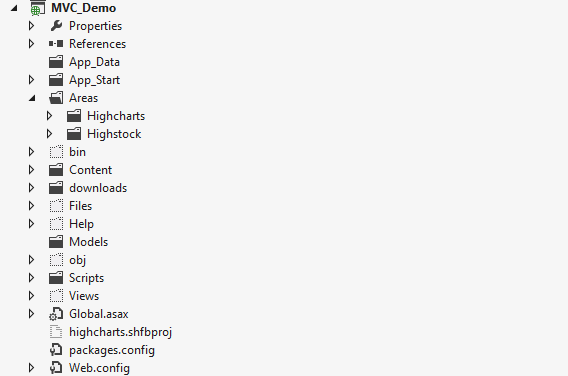
*echo f | xcopy /f /y ..\CodeGenerator\CodeGeneration\Highstock CodeGeneration\*

*echo f | xcopy /f /y ..\CodeGenerator\CodeGeneration\Highstock\Enums CodeGeneration\Enums\*

Whenever the CodeGenerator project is run and files are generated, the copygeneratedfiles.bat files needs to be run to copy the generated files to the MVC\_Highstock project, and then the project must be recompiled. After that the build files must be run to generate the deliverables (.ZIP files)

# MVC\_Demo

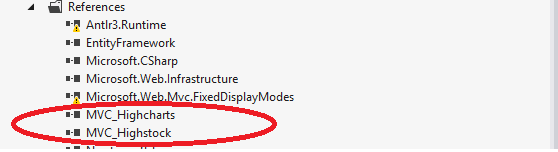
The MVC demo project contains all the demos for all the products that are shipped. It currently contains the Highcharts product demos (around 80) and a few Highstock demos (still in very early alpha stage). The current project structure is the following:



The important folders and files are the following:

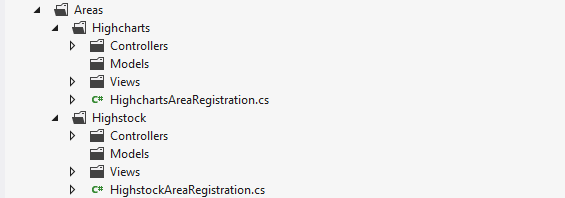
## References

There are many refenreces (DLL files) in the MVC\_Demo project, and almost all of them are generated automatically by Visual Studio .NET. The two references that are of interest are the MVC\_Highcharts and MVC\_Highstock references that refer to the actual libraries we are developing. Customers should also add a reference to one of these (or both) in their own projects:



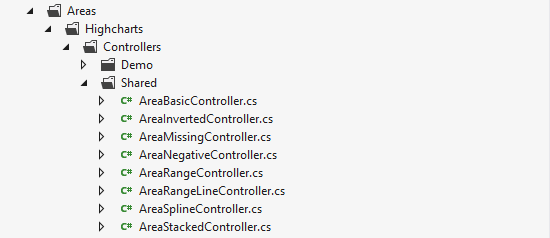
## Areas

Areas are an ASP.NET MVC term that allows for easier separation of Model, View and Controller for different logical parts of the website. We will have a different “Area” for all different products - Highcharts, Highstock and Highmaps.



### Controllers

Controllers contain the code for each different example (C# code). They are located in the following location:



There are currently around 80 different controllers for all different examples online. A typical controller contains the following code:

*public partial class SharedController : Controller*

*{*

*public ActionResult AreaBasic()*

*{*

*List<double?> usaValues = new List<double?> {*

*null, null, null, null, null, 6, 11, 32, 110, 235, 369, 640,*

*1005, 1436, 2063, 3057, 4618, 6444, 9822, 15468, 20434, 24126,*

*27387, 29459, 31056, 31982, 32040, 31233, 29224, 27342, 26662,*

*26956, 27912, 28999, 28965, 27826, 25579, 25722, 24826, 24605,*

*24304, 23464, 23708, 24099, 24357, 24237, 24401, 24344, 23586,*

*22380, 21004, 17287, 14747, 13076, 12555, 12144, 11009, 10950,*

*10871, 10824, 10577, 10527, 10475, 10421, 10358, 10295, 10104 };*

*List<double?> russiaValues = new List<double?> {*

*null, null, null, null, null, null, null, null, null, null,*

*5, 25, 50, 120, 150, 200, 426, 660, 869, 1060, 1605, 2471, 3322,*

*4238, 5221, 6129, 7089, 8339, 9399, 10538, 11643, 13092, 14478,*

*15915, 17385, 19055, 21205, 23044, 25393, 27935, 30062, 32049,*

*33952, 35804, 37431, 39197, 45000, 43000, 41000, 39000, 37000,*

*35000, 33000, 31000, 29000, 27000, 25000, 24000, 23000, 22000,*

*21000, 20000, 19000, 18000, 18000, 17000, 16000 };*

*List<AreaSeriesData> usaData = new List<AreaSeriesData>();*

*List<AreaSeriesData> russiaData = new List<AreaSeriesData>();*

*usaValues.ForEach(p => usaData.Add(new AreaSeriesData { Y = p }));*

*russiaValues.ForEach(p => russiaData.Add(new AreaSeriesData { Y = p }));*

*ViewData["usaData"] = usaData;*

*ViewData["russiaData"] = russiaData;*

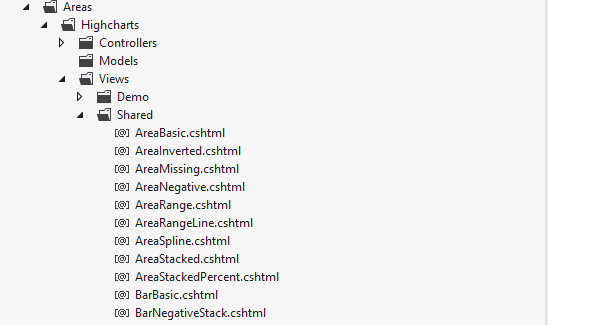
*return View();*

*}*

The rough overview of the code - just get the data for the chart points from database (or other source) and pass them to the view for display:

### Views

Views contain the HTML for each different example The current views structure is the following



Theare are around 80 views for each different example for the Highcharts project. A typical view contains the following code:

*<script type="text/javascript">*

*function formatXAxis() {*

*return this.value; // clean, unformatted number for year*

*}*

*function formatYAxis() {*

*return this.value / 1000 + 'k';*

*}*

*</script>*

*@(Html.Highsoft().Highcharts(*

*new Highcharts*

*{*

*Chart = new Chart*

*{*

*Width = 1087,*

*Height = 400*

*},*

*Title = new Title*

*{*

*Text = "US and USSR nuclear stockpiles"*

*},*

*Subtitle = new Subtitle*

*{*

*Text = "Source: <a href='http://thebulletin.metapress.com/content/c4120650912x74k7/fulltext.pdf'>thebulletin.metapress.com</a>"*

*},*

*XAxis = new List<XAxis>*

*{*

*new XAxis*

*{*

*AllowDecimals = false,*

*Labels = new XAxisLabels*

*{*

*Formatter = "formatXAxis"*

*}*

*}*

*},*

*YAxis = new List<YAxis>*

*{*

*new YAxis*

*{*

*Title = new YAxisTitle*

*{*

*Text = "Nuclear weapon states"*

*},*

*Labels = new YAxisLabels*

*{*

*Formatter = "formatYAxis"*

*}*

*}*

*},*

*Tooltip = new Tooltip*

*{*

*PointFormat = "{series.name} produced <b>{point.y:,.0f}</b><br/>warheads in {point.x}"*

*},*

*PlotOptions = new PlotOptions*

*{*

*Area = new PlotOptionsArea*

*{*

*PointStart = 1940,*

*Marker = new PlotOptionsAreaMarker*

*{*

*Enabled = false,*

*Symbol = "circle",*

*Radius = 2,*

*States = new PlotOptionsAreaMarkerStates*

*{*

*Hover = new PlotOptionsAreaMarkerStatesHover*

*{*

*Enabled = true*

*}*

*}*

*}*

*}*

*},*

*Series = new List<Series>*

*{*

*new AreaSeries*

*{*

*Name = "USA",*

*Data = @ViewData["usaData"] as List<AreaSeriesData>*

*},*

*new AreaSeries*

*{*

*Name = "USSR/Russia",*

*Data = @ViewData["russiaData"] as List<AreaSeriesData>*

*}*

*}*

*}*

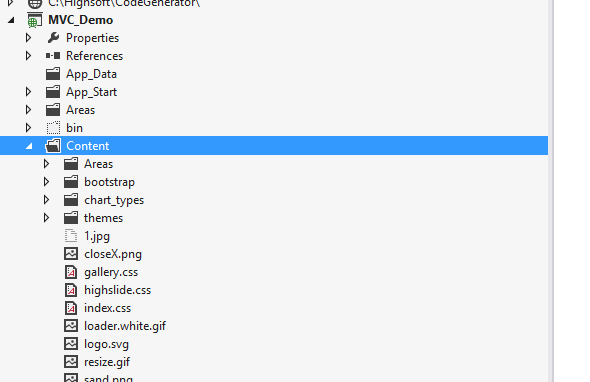
*, "chart")*

*)*

The View contains the chart options in C# format and gets the datasource from the Controller and displays the chart.

### Content

The content folder contains different resources needed by the demo, predominantly images. It also contains CSS files and C# demo files in .TXT format, since Amazon Web Services (ElasticBeansTalk) does not automatically deploy .cs files and we need them in .txt format. The .txt formatted .cs files are located in Content/Areas. Here is the current structure of the Content folder:

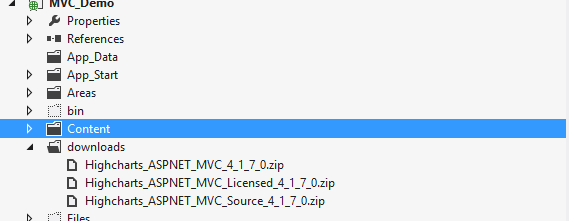


### Downloads

The downloads folder contains the actual deliverables that will be shipped to the customers. They are

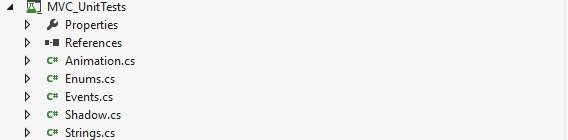
1. Highcharts\_ASPNET\_MVC\_4\_1\_7\_0.zip - the 30-day trial version of the Highcharts.NET product. This is what anyone interested can simply download from our site, test and decide on purchasing decision
2. Highcharts\_ASPNET\_MVC\_Licensed\_4\_1\_7\_0.zip - this is what paying customers get (the same as above, however it never expires)
3. Highcharts\_ASPNET\_MVC\_Source\_4\_1\_7\_0.zip - we may or may not introduce a more expensive “source code” license where customers also get the source code of the product. This is the download for that, also for paying customers.

The download structure currently looks as this:



# MVC\_UnitTests

Contains the unit tests for the Highcharts and Highstock libraries (MVC\_Highcharts and MVC\_Highcharts). These are unit tests written directly against the code generated to verify that the code generated and the HTML / Javascript produced comply to what is expected by the product. The current project structure is:



A typical unit tests that tests the JSON output of the MVC\_Highcharts library is:

*[TestMethod]*

*public void FormatterShouldRenderWithoutQuotes()*

*{*

*\_chart.Chart.Type = ChartType.Area;*

*\_chart.XAxis.Labels.Formatter = "formatXLabels";*

*string json = \_renderer.RenderHtml();*

*Assert.IsTrue(json.Contains("\"formatter\":formatXLabels"));*

*}*

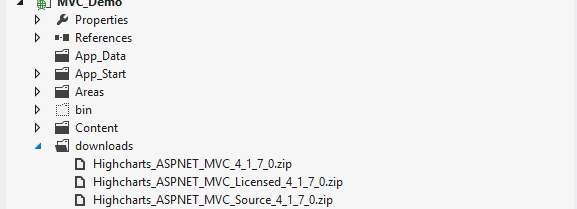
# UtilDateReplacer

The UtilDateReplacer project is a utility project that takes part in the build process. Its purpose is to replace the CompiledOn.cs file in the MVC\_Highcharts and MVC\_Highstock projects with a new CompileOn.cs file with the current date (so that tthe 30-day trial logic kicks in).

The resulting executable is used by the build process Nant file when producing the deliverables to the customers (.zip files)

# Automated Build Process

The purpose of the automated build process is to produce the deliverables - .zip files, which trial and licensed customers can download from the site. There are currently three deliverables in the /Download directory of the MVC\_Demo project



1. Highcharts\_ASPNET\_MVC\_4\_1\_7\_0.zip - the 30-day trial version of the Highcharts.NET product. This is what anyone interested can simply download from our site, test and decide on purchasing decision
2. Highcharts\_ASPNET\_MVC\_Licensed\_4\_1\_7\_0.zip - this is what paying customers get (the same as above, however it never expires)
3. Highcharts\_ASPNET\_MVC\_Source\_4\_1\_7\_0.zip - we may or may not introduce a more expensive “source code” license where customers also get the source code of the product. This is the download for that, also for paying customers.

All build related files are located in the /Build folder in the github repository. In the /Build folder there is a subfolder named /nant which contains the Nant executables. Nant is a .NET port of the popular Java ANT automated build compiler. There are only two other files in the /Build folder

1. build.xml - the ANT xml definitions that constitute the build process itself
2. build.bat - a simple .bat file that executes that Nant executables with build.xml as a parameter and produces the .zip deliverables

All the logic of the build process is contained in the build.xml file. It is a ANT syntax of tasks that performat the following:

<target name="run">

<call target="clean"/>

<call target="generateCompiledOnStampMvc"/>

<call target="buildTrialMvc"/>

<call target="buildLicensedMvc"/>

<call target="copySampleSiteMvc"/>

<call target="updateBinariesMvc"/>

<call target="generateCompiledOnStampMvcSource"/>

<call target="prepareSourceMvc" />

<call target="zipDeliverables"/>

</target>

Theres are the high-level taks and their sequence, the tasks themselves are also in XML format and can be seen in the XML file.