MVC Implementations Playbook

Version 0.92

Table of Contents

Document History **3**

CrownPeak MVC Implementation Overview **4**

ASP.NET MVC Overview **5**

ASP.NET MVC Benefits **6**

Integrating MVC in CrownPeak **7**

Publishing Content as Separate Data Files **7**

MVC vs. Web Forms **8**

CrownPeak Integration **9**

Deploying ASP.NET MVC with the CrownPeak CMS **10**

Building a Basic ASP.NET MVC Project for use with CrownPeak 10

Implementing the ASP.NET MVC to CrownPeak CMS **11**

ASP.NET MVC Deployment to CrownPeak Hosting **12**

Notes **12**

Document History

Author/Editor	Date	Reason for Change	Version
Jason Yoo	01/02/2014	Draft	0.90
Fahd Shaaban	01/06/2014	Edits and new content proposing an approach to implement MVC within CrownPeak	0.91
Paul Taylor	06/23/2014	Edits and write-up based on MVC Reference Architecture	0.92

CrownPeak MVC Implementation Overview

The CrownPeak CMS is a publishing CMS that can generate and deploy content in any format and to any destination thanks to its decoupled architecture, flexible templating capability and API.

By decoupling the CMS from the hosting environment, a website and its underlying framework and technology is not dependent on the CMS for content retrieval at runtime. This means a customer website can enjoy any technology, whether it be Microsoft .Net, Java, PHP, Ruby on Rails, or even static HTML. The CrownPeak CMS API builds on this by allowing authors and editors to focus on the content, while the CMS Templates ensure that the fully tested and approved server-side code (C#, PHP, Java, Perl etc.) is generated and deployed as expected.

Given the CMS' decoupled architecture, the content managed in the CMS has to be included in the published pages, essentially "baked-in" the final output of each "web page". For example, this means that for a site with 100 articles, the CMS would essentially generate and deploy via sFTP 100 different pages containing the templated look and feel and any server-side code, with the unique content for each article in each published page. This is true even for "dynamic" pages powered by .Net, JSP, PHP, etc. The CMS will generate all the pages and deploy them to their final destination to be served to requesting web browsers. The benefits of this approach are numerous and will be discussed and elaborated upon in future playbooks.

More and more customers and/or their agencies are selecting MVC .Net architecture to power their sites.

Microsoft offers two flavors of .Net, the original "Web forms" technology based on ASPX pages with or without code behind, and "MVC", a model-view-controller approach.

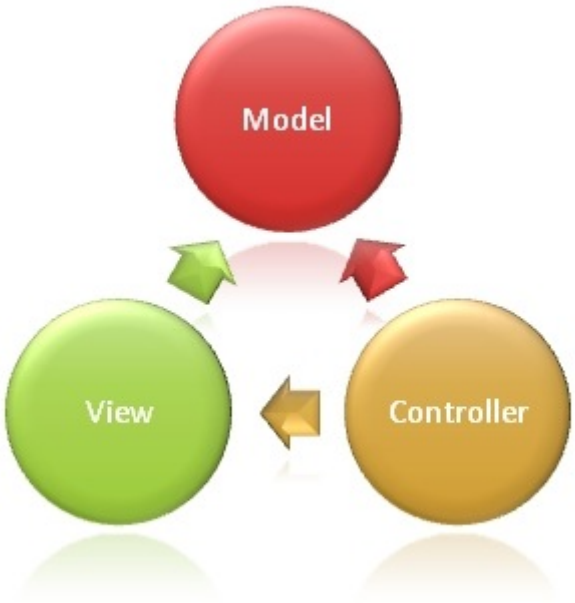
MVC, similarly to Web Forms, can be easily implemented in CrownPeak using templates to produce the necessary MVC components, and deployed with the content "baked-in" to the hosting environment.

This document illustrates how this can be accomplished.

ASP.NET MVC Overview

ASP.NET MVC is one of the methods of developing ASP.NET applications. ASP.NET MVC Framework is Microsoft's Web Application development framework, the other one being traditional web forms framework. MVC is a standard design pattern that many developers are familiar with. The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller.

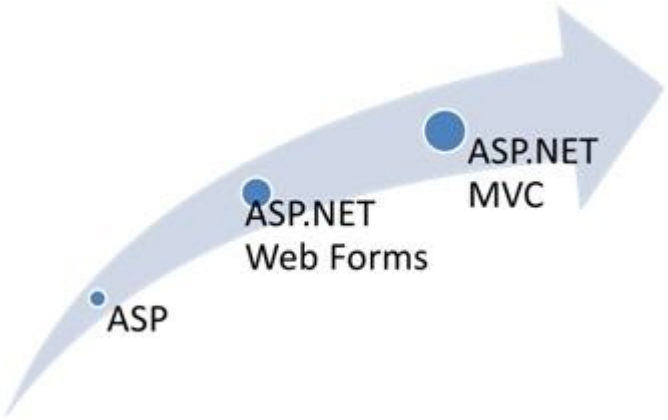
- **The Model** represents the application core. The Model is the part of the application that handles the logic for the application data. Often model objects retrieve data from a database.
- **The View** displays the data. The View is the parts of the application that handles the display of the data. Most often the views are created from the model data.
- **The Controllers** handles the input. The Controllers is the part of the application that handles user interaction. Typically controllers read data from a view, control user input, and send input data to the model.



CrownPeak subscribers that have any of the following needs, should consider developing in ASP.NET MVC:

- Separate data (Model), business logic (Controller), and presentation (View)?
- Is JavaScript going to be used extensively or integrate with other client side tools?
- Looking for good performance application and an Automatic Unit Testing (Find problems early, Simplifies integration etc.)?
- Planning to reuse the same input logic? Promotes a solid application architecture?

ASP.NET MVC Benefits



- **Project Architecture: Provides clean separation of concerns (SoC)** - It makes it easier to manage complexity by dividing an application into the model, the view, and the controller;
- **Enables the full control over the rendered HTML** – The MVC framework does less rendering and what is rendered is leaner. Integration of ASP.NET MVC application with third party client side tools becomes easy;
- **SEO (Search Engine Optimization) friendly URL and Routing**– URL's are friendlier to search engines. Rich routing features treats every URL as a resource supporting REST (Representational state transfer) full interfaces;
- **Enables Test Driven Development (TDD)** – The MVC controller is a separate class so automatic testing is possible featuring Test Driven Development;
- **Reusability** – Controllers are not bound to any specific view and so can be reused for multiple views;
- **Better Performance: No View State/Post Back** - It does not use view state or server-based forms. This makes the MVC framework ideal for developers who want full control over the behavior of an application. There will not be any automatic state management that reduces the page size and therefore impacts performance;
- **Support for Parallel Development** - It works well for Web applications that are supported by large teams of developers and Web designers who need a high degree of control over the application behavior;
- **Extensibility** – ASP.NET MVC supports multiple view engines like aspx, razor, and other customized engines if required;
- **Existing ASP.NET Features** - ASP.NET MVC framework is built on top of matured ASP.NET framework and thus provides developer to use many good features such as forms authentication, windows authentication, caching, session and profile state management etc.

Integrating MVC in CrownPeak

MVC is very different from web forms, but not when it comes to CrownPeak integration. In simple terms, even though MVC separates the core logic and data structure/retrieval (models) from the user interaction and input (Controller), which are combined in Web Forms as code behind, the CrownPeak integration is essentially identical.

The objective is to implement the “presentation” which may include server-side as well as “client-side” code, after they have been fully tested and approved, as templates. These templates expose the areas within the presentation that is defined as managed content as input form fields.

The CMS will then generate “static” versions of these files, replicating the presentation and backend logic as many times as there are “pages” and deploys them with the content embedded or “baked-in” to their final destination.

MVC specialists would likely reject this approach of publishing identical “views” with the only difference being the content, as they believe the content, being the only difference, should be retrieved and populated at run-time. However, MVC is not different from any server-side technology that fundamentally relies on Databases to provide their content. The CrownPeak CMS eliminates this need entirely. Thanks to its decoupled architecture and lack of availability of a public API, the content will need to be deployed to the server as well.

There has been implementations where the views are deployed to the web server using standard sFTP (or using the CMS to deploy non-templated files), with the content published from the CMS as JSON, XML, or HTML Fragments. This approach DOES work, however, it creates a double maintenance issue.

Publishing Content as Separate Data Files

The CMS, thanks to its multi-output support, can easily publish any content in any format, including JSON, XML, CSV, etc.

Typical approaches of publishing content as data files require the double effort and on-going maintenance of implementing the presentation in the CMS for the purpose of In Context Editing within the CMS. This enables authors and editors to edit the content in context of the full presentation.

With the presentation published as non-templated views directly to the web server, this means the views will need to be implemented as templates for the purpose of a complete authoring experience only. However, this results in a double maintenance effort to keep the externally managed views in sync with the implemented presentation. This approach is trouble!

CrownPeak recommends implementing the views and publishing them directly with the content baked in to the final destination.

MVC vs. Web Forms

Capability	Web Forms	MVC
Presentation	Web pages deployed as files with an .aspx extension	Deployed as Views with an .cshtml extension
Server-side Logic	Code behind developed as supporting files with the same name as the pages they support with a .cs extension (for C#), and are deployed in a complied DLL	Controllers and Models provide the server-side logic. The files are named as the View with "Controller" or "Model" appended to the name, with a .cs extension (c#), and are typically compiled and deployed as a DLL.
Shared Global Content (Header/Footer)	Masterpages can be deployed containing all global content (header, footer), and deployed with a .master extension. The Masterpages are referenced in the .aspx pages.	Layout files, which are Views, can be deployed with a .cshtml extension, and referenced by any view.
Includes	User controls provide include capabilities, and are files deployed as pages with a .ascx extension, and referenced by any .aspx page.	Any view can be included within another view as a "Partial View". As such, includes are deployed as standard .cshtml files.
In-line code	C# code can be inserted in-line in any aspx page	C# code can be inserted in a view.

CrownPeak Integration

Capability	Web Forms	MVC
Output Template	Web Pages containing html, css, js, and any server-side C# code are implemented in output templates	Views containing html, css, js, and any server-side C# code are implemented in output templates
Input Template	Content placeholders are exposed as editable input form fields.	Content placeholders are exposed as editable input form fields.
Deployment	Files are deployed with the content baked-in with an .aspx extension.	Files are deployed with the content baked-in with a .cshtml extension.
Shared Global Content (Header/Footer)	Navigation Wrappers are configured to publish as master pages with a .master extension.	Navigation Wrappers are configured to publish as layout files with a .cshtml extension.
Includes	Widgets and other assets can be configured to published as user controls with a .ascx extension, and referenced by other templates using name.aspx and url.aspx.	Widgets and other assets can be configured to published as user controls with a .cshtml extension, and referenced by other templates using name.aspx and url.aspx.
Server-side code	Compiled and deployed as dll	Compiled and deployed as dll

Deploying ASP.NET MVC with the CrownPeak CMS

Before starting the development, it is beneficial to solidify the list of the prerequisites.

- Microsoft Visual Studio or Visual Studio Express (Visual Studio Express is a set of freeware integrated development environments developed by Microsoft as a lightweight version of the Microsoft Visual Studio)
- Microsoft Framework .NET 3.5 or greater

Building a Basic ASP.NET MVC Project for use with CrownPeak

Although custom functionality may be wired into the ASP.NET MVC Project, a customer may wish to leverage the page creation and management abilities of the CrownPeak CMS. In this case, we need to create an ASP.NET MVC Controller, which will handle non-custom page rendering.

From the Visual Studio 'Empty MVC Project', within the /App_Start/RouteConfig.cs file (which controls all routing within the application), add a catchall route. This will be called if none of the custom routes above it are hit during a page request.

```
routes.MapRoute(
"CrownPeakPage",
"{*page}",
new
{
controller = "CrownPeakPage",
action = "Page"
        }
);
```

Create an ASP.NET MVC Controller to handle anything caught by this route. Create a controller at /Controllers/CrownPeak/CrownPeakPageController.cs, with the following:

```
public class CrownPeakPageController : Controller
{
        public ActionResult Page()
        {
                if (Request.Url == null) return null;
                var absolutePath = Request.Url.AbsolutePath.Replace(Request.ApplicationPath,
"");
                if (!absolutePath.StartsWith("/")) absolutePath = "/" + absolutePath;
                if (absolutePath == "/") absolutePath =
ConfigurationManager.AppSettings["CrownPeak:DefaultHome"];
                var viewFile = GetViewFileLocation(absolutePath);
                if (System.IO.File.Exists(Server.MapPath(viewFile))) return View(viewFile);
                Response.StatusCode = 404;
                Return
```

```
View(GetViewFileLocation(ConfigurationManager.AppSettings["CrownPeak:Default404"]));
        }


        private static string GetViewFileLocation(string viewFile)
        {
                return new
StringBuilder().Append(ConfigurationManager.AppSettings["CrownPeak:ViewsRoot"]).Append(viewFil
e).Append(ConfigurationManager.AppSettings["CrownPeak:ViewsFileExtension"]).ToString();
        }
}
```

Finally, set-up the necessary application settings within the Web.config:

```
<appSettings>
        <add key="CrownPeak:ViewsRoot" value="~/Views/CrownPeak" />
        <add key="CrownPeak:ViewsFileExtension" value=".cshtml" />
        <add key="CrownPeak:DefaultHome" value="/index" />
        <add key="CrownPeak:Default404" value="/404" />
        <add key="CrownPeak:Default500" value="/500" />
</appSettings>
```

This is a catchall route, that will handle any page request, which was not caught by a developer's custom routes. This functionality will examine the request URL, and look for a corresponding ASP.NET MVC 'View' file at /Views/CrownPeak/{request_url} and display it. If this is not found, it will redirect the user to /404, or if error occurs, to /500.

Implementing the ASP.NET MVC to CrownPeak CMS

Once the ASP.NET MVC Project is ready to compile, the ASP.NET MVC needs to be implemented in CrownPeak's CMS in the following way:

1. Compile the ASP.NET MVC project and publish from within Visual Studio;
2. Create a folder within the root of the CrownPeak CMS application, and import all compiled code into here. Set the folder's publishing properties to publish to the root of the website;
3. Templatize all the view files (*.cshtml) in CMS;
4. Create a folder in the root of the CrownPeak CMS application to hold the site content (this should be named logically, e.g. /EN for English language content). Set the folder's publishing properties to publish to /Views/CrownPeak, and set the filename.aspx template file to include the .cshtml extension on all published files.

ASP.NET MVC Deployment to CrownPeak Hosting

Before publishing the content, it is beneficial to check with CrownPeak IT Team if the hosting environments are ready for ASP.NET MVC.

Please send a help ticket to *support@crownpeak.com*.

- Microsoft Framework .NET 3.5 or greater is required
- .Net Framework Version for an Application pool needs to be set as 4.0 in IIS.

Notes

All new CrownPeak hosting servers are running on Windows Server 2012 with IIS 8.0 and Microsoft Framework .Net 4.0.

Courtesy domain names (Stage/Live) will be provided by CrownPeak IT Team;

An ASP.NET MVC Reference Architecture has been built, and should be examined and understood prior to beginning this type of integration.