

Saltlake Infosolutions Pvt. Ltd.

<http://www.saltlakesoft.com/>



ASP.NET MVC

A Presentation

by

Tapamay Ghosh
email: tapamay@saltlakesoft.com

What is ASP.NET MVC ?



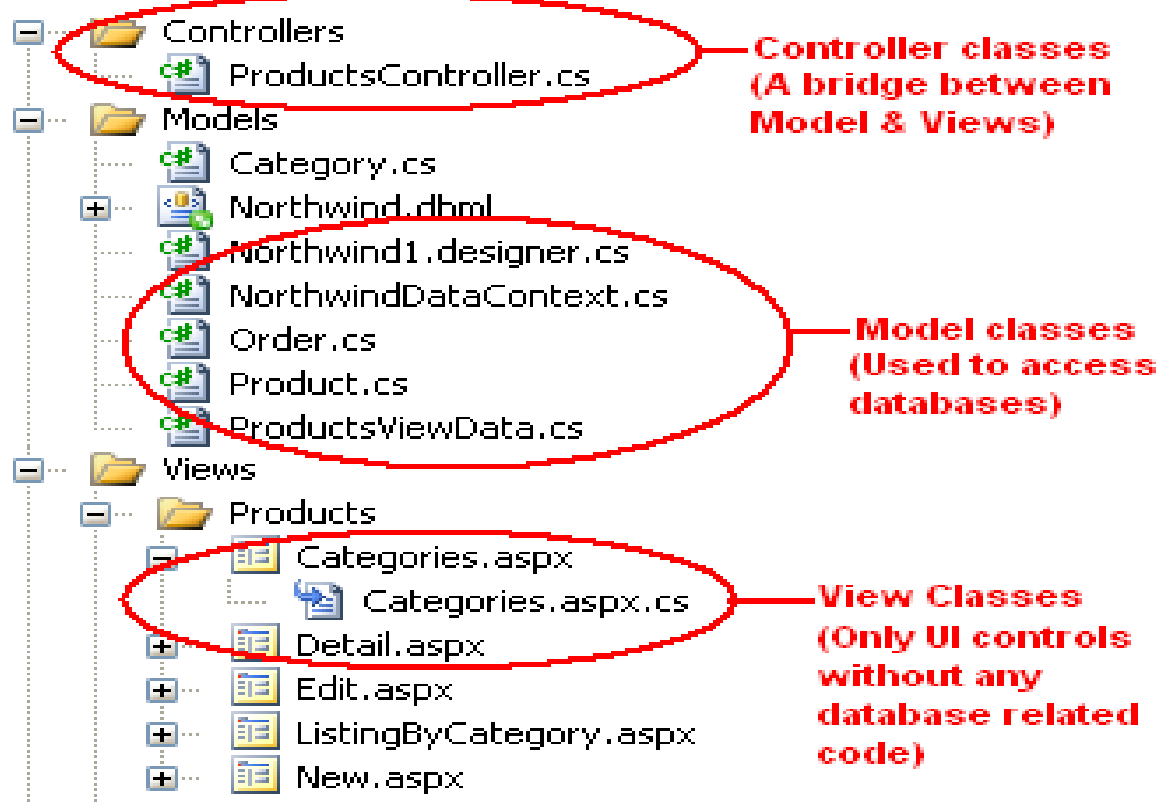
MVC is a framework methodology that divides an application's implementation into three component roles: models, views, and controllers.

* "Models" in a MVC based application are the components of the application that are responsible for maintaining state. Often this state is persisted inside a database (for example: we might have a Product class that is used to represent order data from the Products table inside SQL).

* "Views" in a MVC based application are the components responsible for displaying the application's user interface. Typically this UI is created off of the model data (for example: we might create an Product "Edit" view that surfaces textboxes, dropdowns and checkboxes based on the current state of a Product object).

* "Controllers" in a MVC based application are the components responsible for handling end user interaction, manipulating the model, and ultimately choosing a view to render to display UI. In a MVC application the view is only about displaying information - it is the controller that handles and responds to user input and interaction.

ASP.NET MVC Folder structure?



Step -1: The Route Table is Created

- An ASP.NET MVC application has something called a Route Table. The Route Table maps particular URLs to particular controllers.
- An application has one and only one Route Table. This Route Table is set-up in the Global.asax file
- An application's Route Table is represented by the static RouteTable.Routes property
- This property represents a collection of Route objects.
- A Route object is responsible for mapping URLs to handlers.
- The Global.asax file maps URLs to the MvcRouteHandler.
- The routing infrastructure described in this section is contained in a distinct assembly named System.Web.Routing.dll.

***NOTE:** This new routing infrastructure can be used independently of an ASP.NET MVC application.

Step -1: Code Section

Global.asax

```
public class GlobalApplication : System.Web.HttpApplication
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        // Note: Change the URL to "{controller}.mvc/{action}/{id}" to enable
        //       automatic support on IIS6 and IIS7 classic mode

        //This Route maps any URL that follows the pattern {controller}/{action}/{id} to the MvcRouteHandler.
        routes.Add(new Route("{controller}/{action}/{id}", new MvcRouteHandler())
        {
            Defaults = new RouteValueDictionary(new { action = "Index", id = "" }),
        });

        //This Route maps the particular URL Default.aspx to the MvcRouteHandler.
        routes.Add(new Route("Default.aspx", new MvcRouteHandler())
        {
            Defaults = new RouteValueDictionary(new { controller = "Products", action = "Index", id = "" }),
        });
    }

    protected void Application_Start(object sender, EventArgs e)
    {
        //An application's Route Table is represented by the static RouteTable.Routes property
        RegisterRoutes(RouteTable.Routes);
    }
}
```

Route Object1

A dictionary contains all default values if parameters are null.

Route Object2

Step-2: The UrlRoutingModule Intercepts the Request



- Whenever you make a request against an ASP.NET MVC application, the request is intercepted by the UrlRoutingModule HTTP Module.
- When the UrlRoutingModule intercepts a request, the first thing the module does is to wrap up the current HttpContext in an HttpContextWrapper2 object.
- Next, the module passes the wrapped HttpContext to the RouteTable that was setup in the previous step.
- If a match can be made between the current request and one of the Route objects in the Route Table, then a RouteData object is returned.
- If the UrlRoutingModule successfully retrieves a RouteData object then the module next creates a RouteContext object that represents the current HttpContext and RouteData.
- The module then instantiates a new HttpHandler based on the RouteTable and passes the RouteContext to the new handler's constructor.
- In the case of an ASP.NET MVC application, the handler returned from the RouteTable will always be an MvcHandler (The MvcRouteHandler returns an MvcHandler).
- The last step that the module performs is setting the MvcHandler as the current HTTP Handler. An ASP.NET application calls the ProcessRequest() method automatically on the current HTTP Handler which leads us to the next step.

Step-2: Code Section

Web.config

```
<httpHandlers>
  <remove verb="*" path="*.asmx"/>
  <add verb="*" path="*.asmx" validate="false" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions.dll" />
  <add verb="*" path="*_AppService.axd" validate="false" type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions.dll" />
  <add verb="GET,HEAD" path="ScriptResource.axd" type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions.dll" />
</httpHandlers>
<httpModules>
  <add name="ScriptModule" type="System.Web.Handlers.ScriptModule, System.Web.Extensions.dll" />
  <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule, System.Web.Routing.dll" />
</httpModules>
```

Step-4: The MvcHandler & Controller Executes

- In the previous step, an MvcHandler that represents a particular RequestContext was set as the current HTTP Handler.
- An ASP.NET application always fires off a certain series of events including Start, BeginRequest, PostResolveRequestCache, PostMapRequestHandler, PreRequestHandlerExecute, and EndRequest events etc.
(there are a lot of application events – for a complete list, lookup the [HttpApplication class in the Microsoft Visual Studio 2008 Documentation](#)).
- Everything described in the previous section happens during the PostResolveRequestCache and PostMapRequestHandler events.
- When ProcessRequest() is called on the MvcHandler object created in the previous section, a new controller is created.
- Finally, the Execute() method is called on the controller class.
- The controller determines which controller method to execute, builds a list of parameters, and executes the method.

Step-5: The RenderView Method is Called

- Normally, your controller methods end with a call to either the `RenderView()` or `RedirectToAction()` method.
- The `RenderView()` method is responsible for rendering a view (a page) to the browser.
- When you call a controller's `RenderView()` method, the call is delegated to the current `ViewEngine`'s `RenderView` method.
- The `WebFormViewEngine.RenderView()` method uses a class named the `ViewLocator` class to find the view.
- Next, it uses a `BuildManager` to create an instance of a `ViewPage` class from its path.
- Next, if the page has a master page, the location of the master page is set (again, using the `ViewLocator` class).
- If the page has `ViewData`, the `ViewData` is set.
- Finally, the `RenderView()` method is called on the `ViewPage`.
- The `ViewPage` class derives from the base `System.Web.UI.Page` class.

Step-5: Code Section

ProductController.cs(See the folder structure)

```
public void About ()
{
    RenderView("About");
}

public void Categories() ControllerAction Method
{
    RenderView("Categories", northwind.Categories.ToList());
}

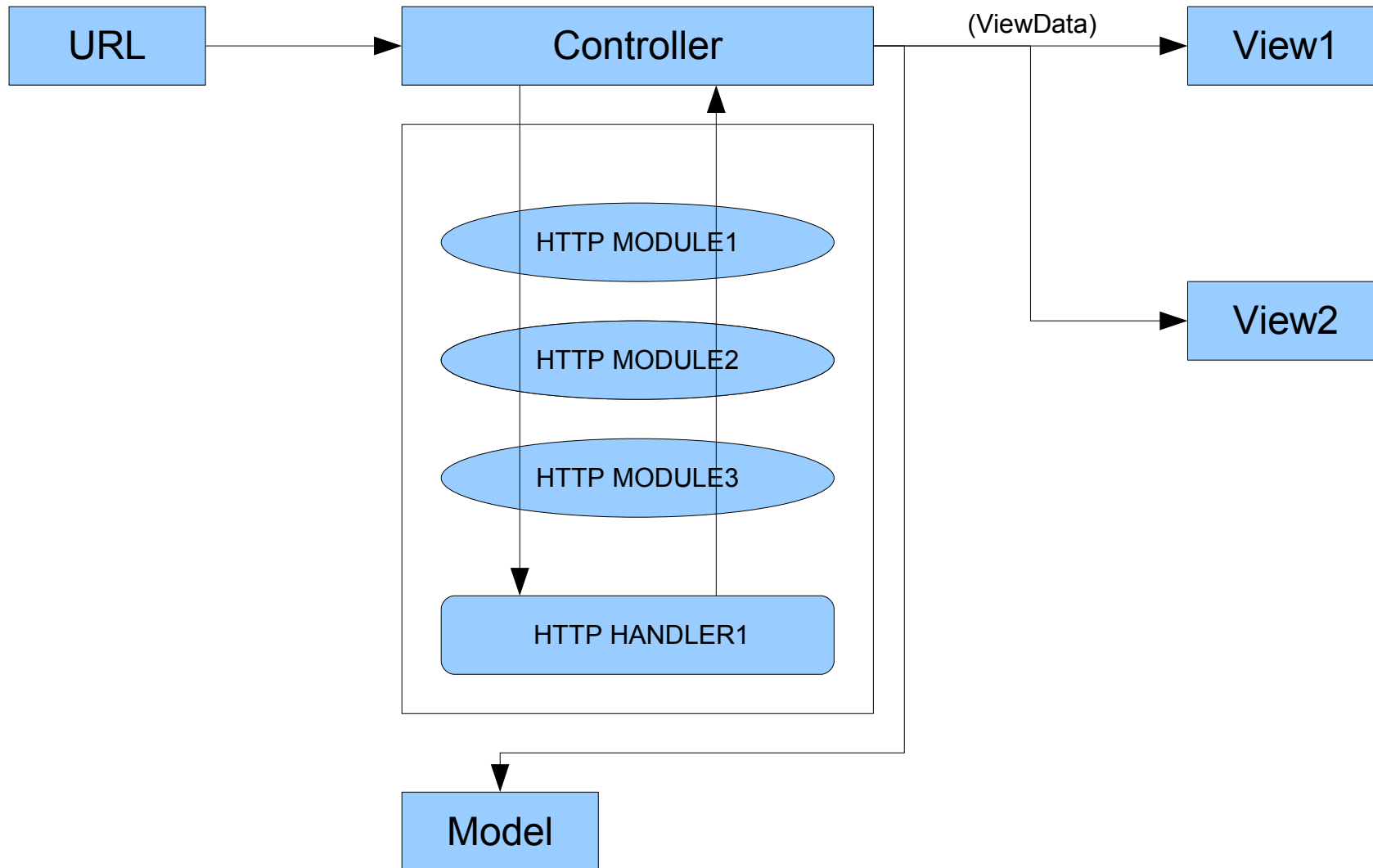
public void Detail(int id) if the request url has to contain "id" in the url pattern
{
    Product product = northwind.Products.SingleOrDefault(p => p.ProductID == id);
    RenderView("Detail", product); RenderView called
    Data from Database
    Name of View (i.e .aspx page)
}

public void List(string id, int? page)
{
    var category = northwind.Categories.SingleOrDefault(c => c.CategoryName == id);

    var products = from p in northwind.Products
                   where p.CategoryID == category.CategoryID
                   select p;

    RenderView("ListingByCategory", products.ToList());
}
```

How ASP.NET MVC does work?



Finish...



Thank You...

References:-

- 1) <http://weblogs.asp.net/stephenwalther/archive/2008/03/17/asp-net-mvc-in-depth-the-life-of-an-asp-net-mvc-request.aspx>
- 2) <http://weblogs.asp.net/scottgu/archive/2007/10/14/asp-net-mvc-framework.aspx>
- 3) <http://weblogs.asp.net/scottgu/archive/2007/12/06/asp-net-mvc-framework-part-3-passing-viewdata-from-controllers-to-views.aspx>