Add Umbraco to an existing MVC site

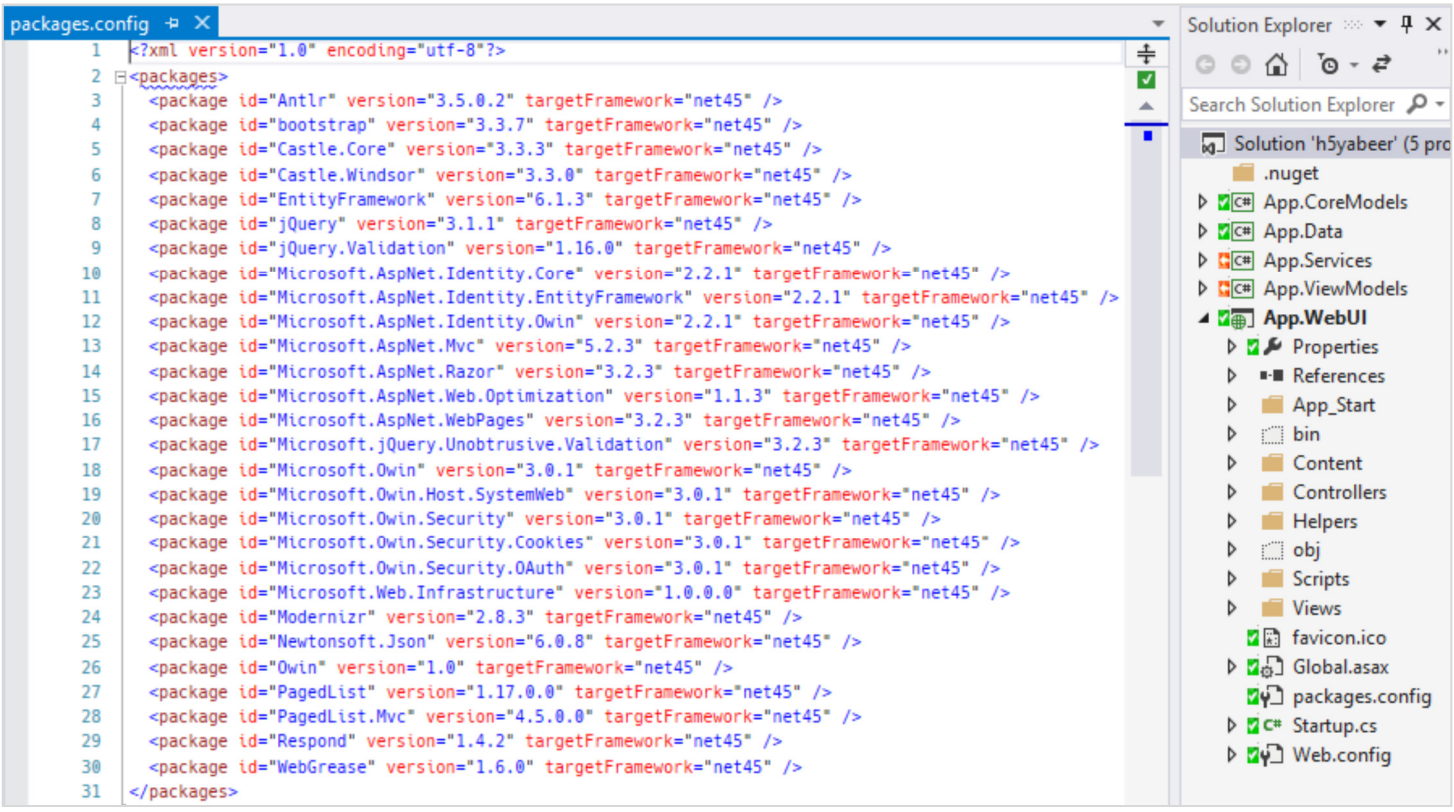
This document is a detailed walkthrough explaining how to add Umbraco into an existing MVC site, to accompany a '24 days in Umbraco' article written by Lotte Pitcher (@lottepitcher).

The example for this walkthrough is a fully working site using:

- MVC (v5.2.3)
- Entity Framework Code First (v6.1.3)
- ASP.Net Identity (v2.2.1) and Owin
- Dependency Injection using Castle Windsor (v3.3.0)

The Visual Studio solution has five projects:

- App.CoreModels – domain models that Entity Framework Code First uses for db creation
- App.Data – data context and migration
- App.Services – services and interfaces
- App.ViewModels – any 'non-domain' view models needed by the website
- App.WebUI – the current working MVC site



The Visual Studio solution and packages.config

Our objective is for content editors to edit the home page using Umbraco without breaking our working application and whilst causing the least amount of 'disruption' to our code as possible.

I assume I don't need to say "make sure you have a full backup" before starting...

Step 1: Check your existing application

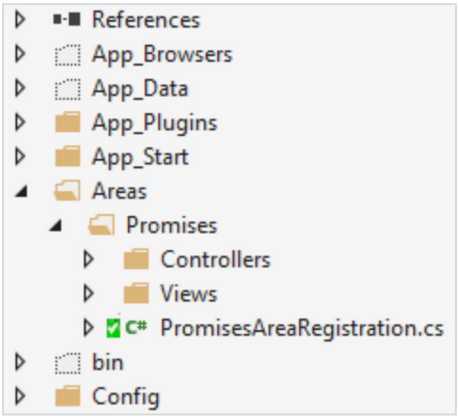
Hopefully your application is already in separate projects along the lines of mine. If not, I suggest you should refactor so that it is. As you can see in the screenshot on the previous page, my website project has only what needs to be there: assets, controllers, views and start-up code. If all your models, services etc. are in separate projects then none of that code will need to change.

You should also check that any nuget packages in common with Umbraco are at the same version. If not, upgrade and test yours.

Step 2: Move to a new MVC area

Arguably this is the most disruptive and fiddly part. To keep your existing application 'safely' away from Umbraco, and to avoid naming conflicts with the content your editors create, I really think moving your application to run in its own MVC area is the way to go.

- Add a new Area to your web project (right click Add > Area) and give it a name. The name is important as all our urls will now be prefixed with this. I'm calling my area 'Promises'.
- Refactor/move all controllers to Areas/{yourarea}/Controllers
- Refactor/move all views folders apart from Shared to Areas/{yourarea}/Views. Also move the _ViewStart.cshtml, assuming you have one.
- Our login url has changed as well of course. Where that is set depends on what you are using to handle user access. I need to update the login path in my Owin startup code.
- Finally you should check the default routing that MVC has created for your area ({yourarea}AreaRegistration.cs) is as required (I had to add Home as the default controller).



Now build and test the site in your browser. Obviously your home page won't load as we've moved all the controllers, so add the name of your area to the url. Hopefully your site should be running fine there, mine is! All the navigation items and page links stay within our 'promises' area. Logging in and out still works, as do all the data entry pages.

If your links don't work then I suggest that, unfortunately, you haven't followed recommended guidelines and have hard-coded some links instead of using the Html.ActionLink and Url.Action helpers. That's a shame if so as you've got some work to do to fix all those before going any further.

Step 3: Change the database connection string name

I believe that Umbraco has to use a connection string with the name of 'umbracoDbDSN'. I can't imagine you used that name, so change it now (assuming you want everything in the one database).

If you are using Entity Framework you also need to change your DbContext class, along the lines of:

```
public AppDbContext() : base("umbracoDbDSN", throwIfV1Schema: false)
{
     Database.SetInitializer<AppDbContext>(null);
}
```

Step 4: Install Umbraco via nuget

Right, now we're ready to do this!

1. Install Umbraco into your web project via nuget, saying [Y]es to overwrite global.asax
2. As the readme reminds us, do a build!
3. Put back the missing elements from your backup web.config (leaving db connection string empty) as Umbraco will have overwritten ours
4. Load the website in the browser and do a custom install using the correct db connection string (assuming you want Umbraco tables and your application tables in the one db).

Step 5: Build the home page in Umbraco

If you get to this point and you don't know how to do things in Umbraco then go on the official training as soon as you can. I'll just continue assuming you do know what you're doing!

- Add the master template (I called mine Site.cshtml) via the back office:
  - Copy in the contents of your existing master layout
  - Set the first line of the view to be: `@inherits UmbracoViewPage<dynamic>`
  - You can now delete the previous master layout
- Create the home page:
  - Create a new document type
  - Set the template to use the master
  - Add an item in the content tree and publish
- Reload the root of the website in the browser

You should see your master layout content. If the CSS doesn't render don't panic. Assuming you're using bundling we'll get that working soon.

Step 6 – MVC area to use master template

Update the MVC area ViewStart file - Areas/{yourarea}/Views/_ViewStart.cshtml - to use to our master Umbraco template:

```
@{
    Layout = "~/Views/Site.cshtml";
}
```

You should also update web.config: add the path in the umbracoReservedPaths appSetting (setting it as '~/yourarea')

Step 7 – Rewire the start-up code

7a. Delete default MVC routing

I imagine you have a RegisterRoutes method in your start-up code which sets a default route. Well delete this route registration as it breaks Umbraco routing!

7b. Move global.asax.cs code

My start-up code in global.asax.cs is not being run because Umbraco changed global.asax to inherit from Umbraco.Web.UmbracoApplication. To run start-up code you have to create a class that implements the IApplicationEventHandler interface.

```
public class ApplicationEventHandler : IApplicationEventHandler
{
    public void OnApplicationInitialized(UmbracoApplicationBase umbracoApplication,
              ApplicationContext applicationContext)
    {
    }

    public void OnApplicationStarting(UmbracoApplicationBase umbracoApplication,
              ApplicationContext applicationContext)
    {
    }

    public void OnApplicationStarted(UmbracoApplicationBase umbracoApplication,
              ApplicationContext applicationContext)
    {
        AreaRegistration.RegisterAllAreas();
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}
```

You must have the AreaRegistration.RegisterAllAreas() method specified so that the route to our MVC area is added to the routes table.

I suggest you delete the now redundant global.asax.cs to avoid confusion in the future.

7c. Using Dependency Injection?

If you are using dependency injection, you will have some start up code that looks something like the following (I'm using Castle Windsor):

```
IoC.Container.Install(FromAssembly.This());
ControllerBuilder.Current.SetControllerFactory(new WindsorControllerFactory());
```

We can't use the SetControllerFactory method any more as this entirely replaces all controller factories, so will break the Umbraco controllers! Instead we need to insert our controller factory into the collection of factories to use (this needs to be in the OnApplicationStarting event of our IApplicationEventHandler class):

```
public void OnApplicationStarting(UmbracoApplicationBase umbracoApplication,
                ApplicationContext applicationContext)
{
    IoC.Container.Install(FromAssembly.This());
    FilteredControllerFactoriesResolver.Current.InsertType<WindsorControllerFactory>(0);
}
```

For this to compile, our controller factory has to implement the IFilteredControllerFactory interface. This requires a 'CanHandle' method to be declared. The highlights indicate what I had to add:

```
public class WindsorControllerFactory : DefaultControllerFactory, IFilteredControllerFactory
{
    readonly IWindsorContainer _container;

    public bool CanHandle(RequestContext requestContext)
    {
        var controllerType = GetControllerType(requestContext,
                  requestContext.RouteData.Values["controller"].ToString());
        return _container.Kernel.HasComponent(controllerType);
    }

    public WindsorControllerFactory(IWindsorContainer container)
    {
        this._container = container;
    }

    public override void ReleaseController(IController controller)
    {
        _container.Release(controller);
    }

    protected override IController GetControllerInstance(RequestContext requestContext,
                Type controllerType)
    {
        if (controllerType == null) return null;
        return (IController)_container.Resolve(controllerType);
    }
}
```

7d. Using ASP.Net Identity and Owin?

Well first of all, be grateful that you're working with Umbraco 7.3+, I think that's right the version for when this all got a lot easier!

In our Owin start-up we don't need the attribute declaring that it is the automatic start-up class. Instead we should inherit from UmbracoDefaultOwinStartup.

The highlights indicate what I had to add:

```
//[assembly: OwinStartup(typeof(Startup))]
namespace App.WebUmb
{
    public partial class Startup : Umbraco.Web.UmbracoDefaultOwinStartup
    {
        public override void Configuration(IAppBuilder app)
        {
            // Umbraco's Owin configuration must be done first
            base.Configuration(app);

            // Then do our Owin configuration
            ConfigureAuth(app);
        }
    }
}
```

Then there are some web.config changes to do:

- In appSettings change the Owin start-up to your class:
- `<add key="owin:appStartup" value="App.WebUmb.Startup" />`
- In system.web: comment out or remove:
  `<authentication> ... <authentication>`
- In system.webServer > modules: comment out or remove
  `<add name="FormsAuthentication" type="System.Web.Security.FormsAuthenticationModule" />`

After you change your authentication settings the safest thing is to change the name of your cookie, or at least that seemed to fix my broken back-office login!

Step 8: Try it!

So hopefully after you've worked through these steps your site is working properly:

- The home page is rendered by Umbraco
- Your MVC area is all working as it did before

Return to the '24 days in Umbraco' article to read about what to do next, and for the github link for the sample code.