22

Building and distributing components using portable areas

This chapter covers

* Implementing  Componentize AccountController & Membership Provider from default template
* Distribute portable area
* Install in ASP.NET MVC project   - Show sample of receiving a Bus Message and writeout user login
* Install in WebForms project
* Distribute RSS Widget (Html.Rss(string url)) with portable area - where Html.Rss is a wrapper of an Html.Action( ) call.

ASP.NET MVC 2's areas allow us to structure the controllers and view within our application, organizing hierchically our projects into folders and namespaces. Portable areas, a feature in MvcContrib, allow us to take that concept even further. Portable areas are like regular areas in that they are a collection of controllers and views - segmented from other areas. But they are also portable: the entire area is a seperate assembly - typically deployed as a DLL file - and can be shared among several ASP.NET MVC 2 projects. In other words, areas allow us to segment our application, but portable areas allow us to compose several applications together in one project.

Imagine a common set of pages and logic that a company would want to share among all its projects. Take, for instance, the common AccountController that's generated in the default ASP.NET MVC 2 project template. AccountController provides basic authentication support - registering users and logging in and the traditional things you'd need to start accepting users. That template could be used as a starter kit for many projects, and they'd all work the same. But as it stands, the AccountController and its supporting players would be duplicated in all of them. We can instead move that stuff into a portable area which all our projects could use. We can eliminate that boilerplate code from our projects and share the new assembly instead of code files. We'll use this example to demonstrate how to use MvcContrib to create a simple portable area, gaining all the benefits of non-duplicated code.

22.1 A simple portable area

A portable area is a class library project with controllers and views. It has all the trappings of an ASP.NET MVC 2 project: controllers, folders for views and the views themselves. To extract the AccountController we'll simply move those related files from the default template to a new class library project. The overall structure of the project is the same, but it's not a web project.

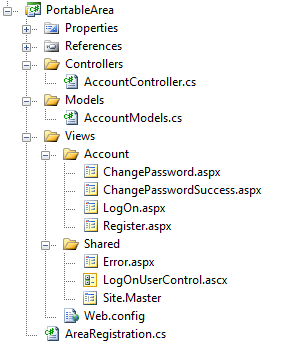


Figure 22.1 A portable area class library project

Developers familiar with the ASP.NET MVC 2 default template will recognize most of the files in the portable area shown in figure 22.1. For the most part, it's exactly the same and in the same structure. The views, however, are not content files like in ASP.NET MVC 2 projects; they are embedded resources. To make a view an embedded resource, highlight it in Solution Explorer and press the F4 key, or right-click it and select Properties from the context menu. The properties window (shown in figure 22.2) will appear.

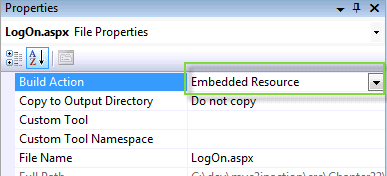


Figure 22.2 Visual Studio's file properties window

Select "Embedded Resource" to instruct Visual Studio to include the file as an embedded resource of the project.

Embedded resources

Embedded resources are project artifacts that are compiled into the assembly and they can be programmatically retrieved. Normally, views are set with a build action of "Content" which means they will be stored and accessed like regular files in the file system. Class files have a build action of "Compile", which compiles them into the assembly regularly. For more information on embedded resources, visit the MSDN reference page: http://msdn.microsoft.com/en-us/library/ht9h2dk8.aspx

Like a regular area, portable areas must be registered. Here we use a base class provided by MvcContib, PortableAreaRegistration.

Listing 22.1 Registering our portable area by deriving from PortableAreaRegistration

public class AreaRegistration : PortableAreaRegistration #1

{

public override string AreaName #2

{

get { return "login"; }

}

public override void RegisterArea

(AreaRegistrationContext context, IApplicationBus bus) #3

{

context.MapRoute(

"login",

"login/{controller}/{action}",

new { controller = "Account", action = "index" });

base.RegisterTheViewsInTheEmbeddedViewEngine(GetType()); #4

}

}

#1 Deriving from PortableAreaRegistration

#2 We still provdide AreaName

#3 RegisterArea is familiar...

#4 but we call a special method

In listing 22.1 we register our portable area. It's very similar to the regular AreaRegistration classes we wrote in chapter 21, with one additional, required step: we must call base.RegisterTheViewsInTheEmbeddedViewEngine(GetType()); (4).

That call allows us to use a special view engine (also included in MvcContrib) that makes our embedded views available to the consuming project. The embedded views are the trick behind portable areas. When our consuming project needs a view, the special embedded view engine can find them. If we didn't use this view engine, we'd have to automate our deployments so that each portable area's views were in the correct spot in our projects file system. Even though this can be automated, using embedded views allows us to skip this tedious and error prone step. In the next section we'll actually use the portable area in our consuming application.

22.2 Consuming portable areas

Once we have our portable area class library project with its controllers and embedded views, we must configure our consuming application so that it can use them. MvcContrib makes this easy. We only need one additional call in the bootstrapping code in Global.asax.cs. This is shown in listing 22.2.

Listing 22.2 Consuming a portable area in a regular ASP.NET MVC 2 project

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas(); #1

RegisterRoutes(RouteTable.Routes);

MvcContrib.UI.InputBuilder.InputBuilder.BootStrap(); #2

}

#1 Register areas normally

#2 Required for portable areas

The routing to register all areas (1) will look for any assemblies in the bin folder - if our portable area project is referenced by the consuming application it goes there automatically. If our consuming application does not reference the portable area assembly, we need to put it in the bin folder. That can be done automatically using a post-build step, configued in the build tab of the project properties.

Our consuming application must also tell MvcContrib to prepare the portable area (2). This is all that's needed to begin using the shared functionality of our portable area. In our consuming project we can link to an otherwise use portable area controllers as if they were included in our project.