ASP.NET MVC 2 IN ACTION

Jeffrey Palermo
Ben Scheirman
Jimmy Bogard
Eric Hexter
Matthew Hinze

FOREWORDS BY ROD PADDOCK AND PHIL HAACK

New Text

MANNING

ASP.NET MVC 2
in Action

by Jeffrey Palermo,
Ben Scheirman,
and Jimmy Bogard

Chapter 22

brief contents

PART 1 HIGH-SPEED FUNDAMENTALS 1

1 ■ High-speed beginner ramp-up 3
2 ■ Presentation model 22
3 ■ View fundamentals 31
4 ■ Controller basics 50
5 ■ Consuming third-party components 66
6 ■ Hosting ASP.NET MVC applications 78
7 ■ Leveraging existing ASP.NET features 95

PART 2 JOURNEYMAN TECHNIQUES.......................................117

8 ■ Domain model 119
9 ■ Extending the controller 127
10 ■ Advanced view techniques 136
11 ■ Security 152
12 ■ Ajax in ASP.NET MVC 167
13 ■ Controller factories 190
14 ■ Model binders and value providers 203
15 ■ Validation 215

PART 3 MASTERING ASP.NET MVC 225

16 ■ Routing 227
17 ■ Deployment techniques 251
18 ■ Mapping with AutoMapper 258
19 ■ Lightweight controllers 268
20 ■ Full system testing 283
21 ■ Organization with areas 301
22 ■ Portable areas 312
23 ■ Data access with NHibernate 322

PART 4 CROSS-CUTTING ADVANCED TOPICS 347

24 ■ Debugging routes 349
25 ■ Customizing Visual Studio for ASP.NET MVC 356
26 ■ Testing practices 364
27 ■ Recipe: creating an autocomplete text box 380

22 Portable areas

This chapter covers

- Building a portable area
- Embedding views
- Distributing a portable area
- Creating an `RssWidget` portable area
- Integrating with a host using the bus

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

Imagine a common set of pages and logic that a company wanted 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, logging in, and the other 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 way. But as it stands, the `AccountController` and its supporting players would be duplicated in all of them. We could instead move this into a portable area that 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 nonduplicated code.

22.1 Understanding the portable area

The portable area is a concept that comes from the MvcContrib project. As the name suggests, it's a native MVC 2 area packaged up in a way that's easier to distribute and consume than an area built with the out-of-the-box MVC 2 support. That's a pretty broad statement, so let's first look at what's in an area and then cover which pieces may need to be made portable.

Areas are a subset of an MVC application that are separated in a way that gives them some physical distance from other groups of functionality in the application. This means that an area will have one or more routes, controllers, actions, views, partial views, master pages, and content files, such as CSS, JavaScript, and image files. These are all the pieces that may be used in an area.

Of those individual elements, many aren't part of the binary distribution of an MVC application. Only the routes, controllers, and actions get compiled into an assembly. The rest of the elements are individual files that need to be copied and managed with the other assets that are part of the application. This is reasonably trivial to manage if we build an area for our application and just use it as a way of managing smaller modules of the application. But if we want to use an area as a way of packaging up and sharing or distributing a piece of multipage UI functionality, managing all of the individual files make this option a bad choice when integrating someone else's component with our application.

This is where the MvcContrib project developed the idea of portable areas. By building on top of the existing area functionality, it only takes some minor changes to an area project to make it portable. A portable area is simply an area that can be deployed as a single DLL.

The process of making an area portable is trivial. As area developers, instead of leaving the file assets as content items in your project, we make them embedded resources. An *embedded resource* is a content file that's compiled into the assembly of a project. The file still exists, and it can be programmatically extracted from the assembly at runtime. This means that a portable area only contains a single file, the assembly of the project, rather than all the individual content files.

22.2 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 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, as shown in figure 22.1.

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, the content is exactly the same, and it's in the same structure. But the views aren't content files like in ASP.NET MVC 2 projects; they're embedded resources.

To make a view an embedded resource, select 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.

For the Build Action, select Embedded Resource to instruct Visual Studio to include the file as an embedded resource of the project.

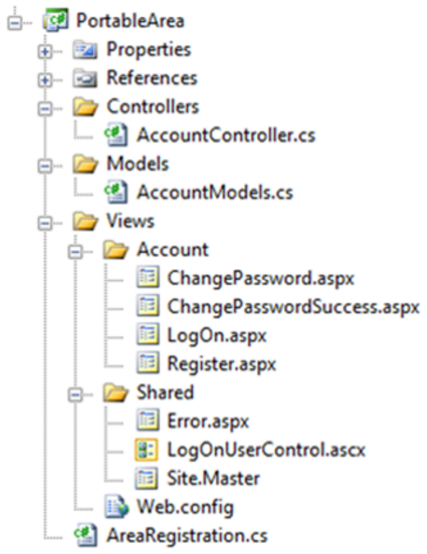
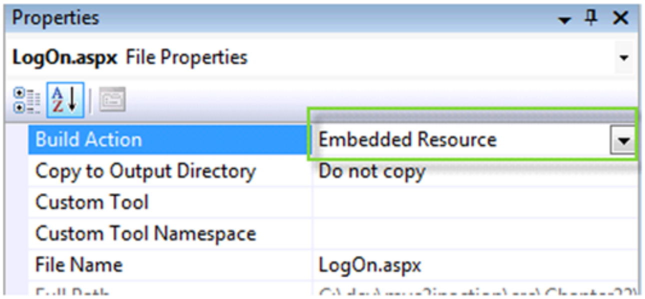

```
PortableArea
  Properties
  References
  Controllers
    AccountController.cs
  Models
    AccountModels.cs
  Views
    Account
      ChangePassword.aspx
      ChangePasswordSuccess.aspx
      LogOn.aspx
      Register.aspx
    Shared
      Error.aspx
      LogOnUserControl.ascx
      Site.Master
    Web.config
  AreaRegistration.cs
```

Figure 22.1 A portable area class library 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'll be stored and accessed like regular files in the filesystem. 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://mng.bz/Uz67.

Like regular areas, portable areas must be registered. This is done by inheriting from a base class provided by MvcContrib, `PortableAreaRegistration`, as shown in listing 22.1.


Properties

LogOn.aspx File Properties

Build Action	Embedded Resource
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	
File Name	LogOn.aspx

Figure 22.2 Visual Studio's Properties window

Listing 22.1 Registering a portable area from `PortableAreaRegistration`

```
public class AreaRegistration : PortableAreaRegistration
{
    public override string AreaName
    {
        get { return "login"; }
    }

    public override void RegisterArea
      (AreaRegistrationContext context, IApplicationBus bus)
    {
        context.MapRoute(                                    Registers ❶
            "login",                                      embedded views
            "login/{controller}/{action}",
         new { controller = "Account", action = "index" });

        base.RegisterTheViewsInTheEmbeddedViewEngine(GetType());  <-
    }
}
```

In listing 22.1 we register our portable area. It's similar to the regular `AreaRegistration` classes we wrote in chapter 21, with one additional required step: we must call `base.RegisterTheViewsInTheEmbeddedViewEngine(GetType())` ❶. 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 project's filesystem. 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 use the portable area in our consuming application.

22.3 Consuming portable areas

When 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. As well as registering the area, we also need to call `InputBuilder.BootStrap` in Global.asax.cs, as 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();

   RegisterRoutes(RouteTable.Routes);

   MvcContrib.UI.InputBuilder.InputBuilder.BootStrap();
}
```

The call to `AreaRegistration.RegisterAllAreas` 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 doesn't reference the portable area assembly, we need to put it in the bin folder. That can be done automatically using a postbuild step configured on the Build tab of the project's Properties dialog box.

In addition to registering the area, the call to `InputBuilder.BootStrap` initializes a custom view engine that can be used to render views that are configured as embedded resources within the portable area.

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

22.4 Creating an RSS widget with a portable area

A portable area can and should include additional helpers to make the use of consuming a portable area frictionless for developers.

Consider a portable area that would provide a web page widget for rendering an RSS feed as an unordered list. We'll walk through an example and look at how we can add a helper to make the portable area easier to use. Figure 22.3 shows the Visual Studio structure for the `RssWidget` portable area.

The `RssWidget` project shown in listing 22.3 contains all the files that are part of this portable area. The interesting difference between this `RssWidget` example and the previous example is the addition of the `SyndicationService` and the `HtmlHelperExtensions` classes. This example demonstrates that you can include a complete feature in a portable area. We've found that by including custom HTML helpers in the projects, the ease of use for the area increases significantly. Let's walk through the code.

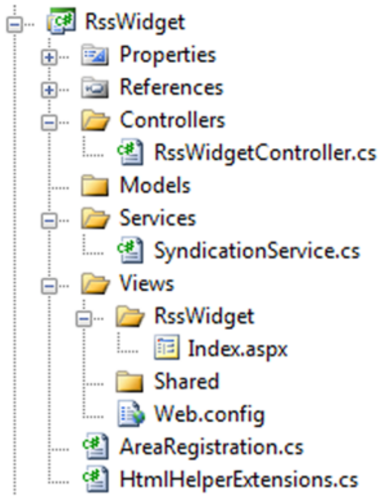

RssWidget
Properties
References
Controllers
RssWidgetController.cs
Models
Services
SyndicationService.cs
Views
RssWidget
Index.aspx
Shared
Web.config
AreaRegistration.cs
HtmlHelperExtensions.cs

Figure 22.3 Layout of the `RssWidget` portable area

Listing 22.3 `RssWidget` registration

```
using System.Web.Mvc;
using MvcContrib.PortableAreas;

namespace RssWidgetPortableArea
{
    public class RssWidgetAreaRegistration : PortableAreaRegistration
    {
        public override string AreaName
        {
            get { return "RssWidget"; }
        }

        public override void RegisterArea(AreaRegistrationContext context,
                     IApplicationBus bus)
        {
            context.MapRoute(                          <-- 1 Maps routes for area
                "RssWidget_default",
                "RssWidget/{controller}/{action}/{id}",
                new {action = "Index", id = ""});

            RegisterTheViewsInTheEmbeddedViewEngine(   <-- 2 Registers embedded views
                GetType());
        }
    }
}
```

The registration code for the area, in listing 22.3, is boilerplate code. The standard calls to `MapRoute` 1 and `RegisterTheViewsInTheEmbeddedViewEngine` 2 are included. No special registration code is needed for this example.

Only one action is included in this portable area—the `RssWidgetController.Index` method. This method is basic. Its only purpose is to tie together the `RssUrl` and the `SyndicationService` dependency. See listing 22.4 for the details of the `Index` method.

The `SyndicationService` provides the logic to retrieve an RSS feed from a URL and return the model of the feed. The controller then sends that model to the view for formatting, as shown in listing 22.4.

Listing 22.4 Passing the contents of the feed to the view

```
using System.Web.Mvc;

namespace RssWidgetPortableArea.Controllers
{
    public class RssWidgetController : Controller
    {
        public ActionResult Index(string RssUrl)
        {
            var service = new SyndicationService();
            var feed = service.GetFeed(RssUrl, 10)     <-- Gets feed based on RssUrl
            return View(feed);
        }
    }
}
```

The feed is rendered by a simple view—shown in listing 22.5—that will create an unordered list of the items in the RSS feed. The code is pretty simple in this view. It loops over a collection of `System.ServiceModel.Syndication.SyndicationFeed` objects and displays the `Title` and `Author` for each item.

If a developer needs to control the HTML for this widget, the great thing about a portable area is that we can override this view and still take advantage of the controller and `SyndicationService` provided by the component. Using the portable area isn't an all-or-nothing decision. Because the portable area is built on top of the MVC 2 areas implementation, it's easy to start taking control back from the component and providing our own implementation code. This can be considered incremental customization.

The view for displaying the RSS feed is shown in listing 22.5.

Listing 22.5 View for the `RssWidget.Index` action

```
<%@ Page Title="" Language="C#"
Inherits="System.Web.Mvc.ViewPage<
     System.ServiceModel.Syndication.SyndicationFeed>" %>
<ul>
    <%foreach(var item in Model.Items) {%>
        <li>
          <%=item.Title.Text %> -
          <%=item.Authors[0].Name %>
       </li>
    <%} %>
</ul>
```

The view in listing 22.5 iterates over each item in the feed and displays the title as well as the author inside an unordered list.

The developer's experience using this `RssWidget` portable area is where this type of component model shines. Using this widget in an application consists of referencing the HTML helper extensions from our view and then calling the `RssWidget` method, as shown in listing 22.6.

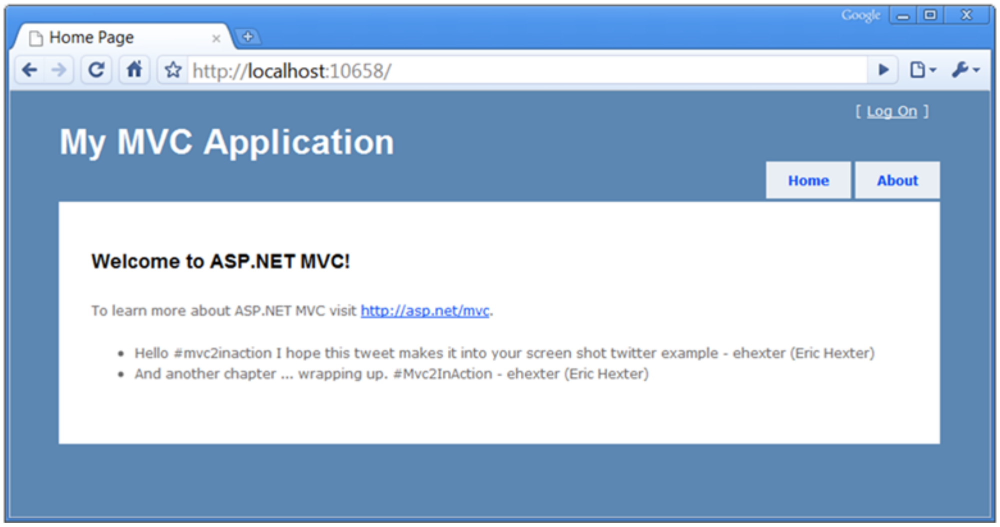
Listing 22.6 Calling an `RssWidget HtmlHelper` extension

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage" %>
<%@ Import Namespace="RssWidgetPortableArea"%>            <-- Imports helper namespace

<asp:Content ID="indexTitle"
    ContentPlaceHolderID="TitleContent" runat="server">
    Home Page
</asp:Content>

<asp:Content ID="indexContent"
              ContentPlaceHolderID="MainContent"
              runat="server">                    Invokes RssWidget helper ❶
<%
Html.RssWidget(
    "http://search.twitter.com/search.atom?q=%23mvc2inaction");
%>
</asp:Content>
```

The only line of code in the application that calls the portable area is the call to the `RssWidget` method ❶. After calling that method and running a simple view, the resulting web page is displayed in figure 22.4. The view merely references an RSS feed for Twitter messages containing "MVC2InAction." The title and user will show up on the screen.

The `RssWidget` HTML helper method that's used in the view is the syntactic sugar that makes consuming this portable area simple. If this method weren't made available, developers using the portable area would have to know some of the internals of how the area was constructed.

For example, the `RssWidget` was intended to be used with the `RenderAction` method calling the `RssWidgetController`'s `Index` method. To make that call, the area name registered in the area's registration is required, and in this case the area name is `RssWidget`. The implementation of the `RssWidget` helper is shown in listing 22.7.



My MVC Application

Welcome to ASP.NET MVC!

To learn more about ASP.NET MVC visit http://asp.net/mvc.

- Hello #mvc2inaction I hope this tweet makes it into your screen shot twitter example - ehexter (Eric Hexter)
- And another chapter ... wrapping up. #Mvc2InAction - ehexter (Eric Hexter)

Figure 22.4 The view that uses the `RssWidget` portable area

Listing 22.7 Hiding complexity in an `HtmlHelper` extension method

```
using System.Web.Mvc;
using System.Web.Mvc.Html;

namespace RssWidgetPortableArea
{
    public static class HtmlHelperExtensions
    {
        public static void RssWidget(this HtmlHelper helper, string RssUrl)
        {
            helper.RenderAction("Index", "RssWidget",
                new {RssUrl, Area = "RssWidget"});
        }
    }
}
```

The `HtmlHelper` extension method, displayed in listing 22.7, shows a call to `RenderAction` that could easily be put into the view directly in order to call the appropriate action in the portable area, but this call requires knowledge about the internals of the area.

By moving this code into an HTML helper extension method, all code specific to the portable area can be pushed into the portable area. As a result, the developer using the area just needs to worry about where the widget should be displayed in the application and what RSS URL needs to be displayed. Creating this separation of concerns allows us the flexibility to make internal changes to the implementation while leaving the public-facing interface nice and simple.

22.5 Distributing the RssWidget

We've covered how to create the widget and how to use it from an MVC application. The one missing piece is distributing the `RssWidget` portable area.

This entire component was written in a way that allows it to be compiled down to one file. To use this portable area from an MVC application, the application needs the portable area in its bin directory, so distributing the portable area consists of distributing the DLL. We recommend distributing portable areas in a zip file, and that package should include:

- The assembly
- A readme file that explains what the portable area is intended to do
- A sample application that shows how to use the portable area

Developers should also consider including a license, which makes it clear to anyone using the portable area how it's intended to be distributed and used.

We don't see portable areas being a tool that's tied to just open source or component vendors exclusively. The concept demonstrates the technical solution to easily sharing functionality. We see this as being interesting to both open source and closed source developers and companies.

22.6 Interacting with the portable area bus

The samples that we've covered so far have solved some pretty specific problems. These examples have been able to take little input from the hosting application and provide some useful benefits. In most cases, a portable area will need to programmatically interact with the hosting application, and rather than leaving the method of interacting up to each portable area developer, the MvcContrib project laid out a simple but effective mechanism: a message bus. The bus was created to allow synchronous communication to send and receive messages that the portable area defines.

As an example, let's take the login portable area from section 22.2. If this area simply provided a user interface for logging in but didn't provide any mechanism for looking up usernames and passwords, it could send a message on the bus. The hosting application could then look up a username in its custom user data store, compare the password, and then return a message, letting the portable area know whether the user's credentials are valid.

Let's look at how a message is sent from a portable area. Here's a call to send a message down the bus:

```
MvcContrib.Bus.Send(new RssWidgetRenderedMessage{Url = RssUrl});
```

This example shows a one-way message being sent to an application, say for logging purposes.

In order for a message to be received, the host application needs to register a handler, like this:

```
MvcContrib.Bus.AddMessageHandler(typeof(RssMessageHandler));
```

Registering a message handler is a one-line call that should only happen once in an application at application startup. The bus will keep track of the handlers and messages and make sure the handlers are called when needed.

The code that's more interesting is the `RssMessageHandler` class. Each message handler needs to be implemented in the host application. Handlers should be considered integration code that stitches together a portable area with the host application. This means that the handler code should be minimized, and that it relies on application service classes rather than on implementing logic inside of a handler class.

Listing 22.8 demonstrates the boilerplate code required to implement a message handler for a message using the bus.

Listing 22.8 A message handler class

```
using MvcContrib.PortableAreas;
using RssWidgetPortableArea.Controllers;

namespace RssWidgetPortableArea
{
    public class RssMessageHandler :
             MessageHandler<RssWidgetRenderedMessage>
    {
        public override void Handle(
             RssWidgetRenderedMessage message)
        {
            //log the message to the application's log.
        }
    }
}
```

Inside the `Handle` method, you can implement calls to your application services and data storage.

22.7 Summary

The biggest benefit that a portable area can provide over a standard area is the ability to distribute the portable area as a single assembly. This chapter showed how to create a portable area.

We learned how using this mechanism can allow us to build reusable components easily. We also saw how easy it is to distribute portable areas and that rich functionality can be integrated using the portable area bus.

Portable areas are just one tool that allows developers to build functionality more quickly, and we'll show how using object-relational mapping tools like NHibernate can increase your team's productivity. The next chapter covers using NHibernate to streamline your application's data access.

ASP.NET / WEB DEVELOPMENT

ASP.NET MVC 2 IN ACTION

Palermo • Scheirman • Bogard • Hexter • Hinze
Forewords by Rod Paddock and Phil Haack • Technical Editor Jeremy Skinner

The future of high-end web development on the Microsoft platform, ASP.NET MVC 2 provides clear separation of data, interface, and logic and radically simplifies tedious page and event lifecycle management. And since it's an evolution of ASP.NET, you can mix MVC and Web Forms in the same application, building on your existing work.

ASP.NET MVC 2 in Action is a fast-paced tutorial designed to introduce the MVC model to ASP.NET developers and show how to apply it effectively. After a high-speed ramp up, the book presents over 25 concise chapters exploring key topics like validation, routing, and data access. Each topic is illustrated with its own example so it's easy to dip into the book without reading in sequence. This book covers some high-value, high-end techniques you won't find anywhere else!

What's Inside

- Dozens of self-contained examples
- Real-world use cases
- Full-system testing for ASP.NET applications

All authors are Microsoft MVPs and ASPInsiders. **Jeffrey Palermo** is cofounder of MvcContrib and CIO of Headspring Systems. **Ben Scheirman, Jimmy Bogard, Eric Hexter** (the other cofounder of MvcContrib), and **Matthew Hinze** are architects and .NET community leaders.

For online access to the authors and a free ebook for owners of this book, go to manning.com/ASP.NETMVC2inAction

Free ebook
SEE INSERT

"...learn from expert users of the ASP.NET MVC framework."
—From the Foreword by Rod Paddock

"An authoritative source on ASP.NET MVC 2. Pick up this book!"
—Alessandro Gallo
Microsoft MVP

"Learn MVC 2 from the people who helped shape it.
—Alex Thissen
Killer-Apps

"Hands-down the best MVC resource available!"
—Andrew Siemer
Lamps Plus

ISBN 13: 978-1-935182-79-5
ISBN 10: 1-935182-79-X
54999
9 781935 182795

MANNING $49.99 / Can $62.99 [INCLUDING eBOOK]