6

ASP.NET MVC: Putting you in control

* Know about filters
* Know about major ASP.NET components
* Know about the MVC pattern

When I started developing on the .NET platform I came from ASP with Javascript and basically went from a stateless model of programming to a model that pretended to be stateful, ASP.NET WebForms. Over the years I’ve seen some pretty horrible consequences from that model like a view state that contains 25MB worth of data for example, which get sent back and forth over the wire. So I welcomed a more web-friendly way of building .NET web applications with open arms.

I had been in anticipation of the first release of ASP.NET MVC when I first heard about it and also contributed the xslt view engine to the ASP.NET MVC framework about 2 days after the first community technology preview. This view engine now exists as part of the mvc contrib. project and can be found on github. Although many people complained that there weren’t many features in the ASP.NET framework that could be found in other ones, I actually liked it because it didn’t force me down a path that I did or did not want to take.

This chapter of the book will present a sample application, this sample application is nowhere neer production ready code and functions as a starting point on how you might go about integrating IronRuby in your own projects. Many of the topics we’re going to cover will come back when we get to the chapter about IronRuby on Rails. The most prevalent topics when talking about any MVC implementation for the web are: routing, filters, models, views and controllers. We’ll discuss all of these shortly. Because the ASP.NET MVC framework is built on top of the ASP.NET technology we’ll start by looking at the most important and relevant components of the ASP.NET technology.

6.1 Anatomy of the ASP.NET technology stack

The .NET framework is huge, heck even ASP.NET in itself is a fairly large framework with facilities to do many of the most common tasks you need to perform when developing web applications. I won’t explain the low-level plumbing like what happens when a request arrives at IIS but we will look at some of the basic components that are key components of every web application.

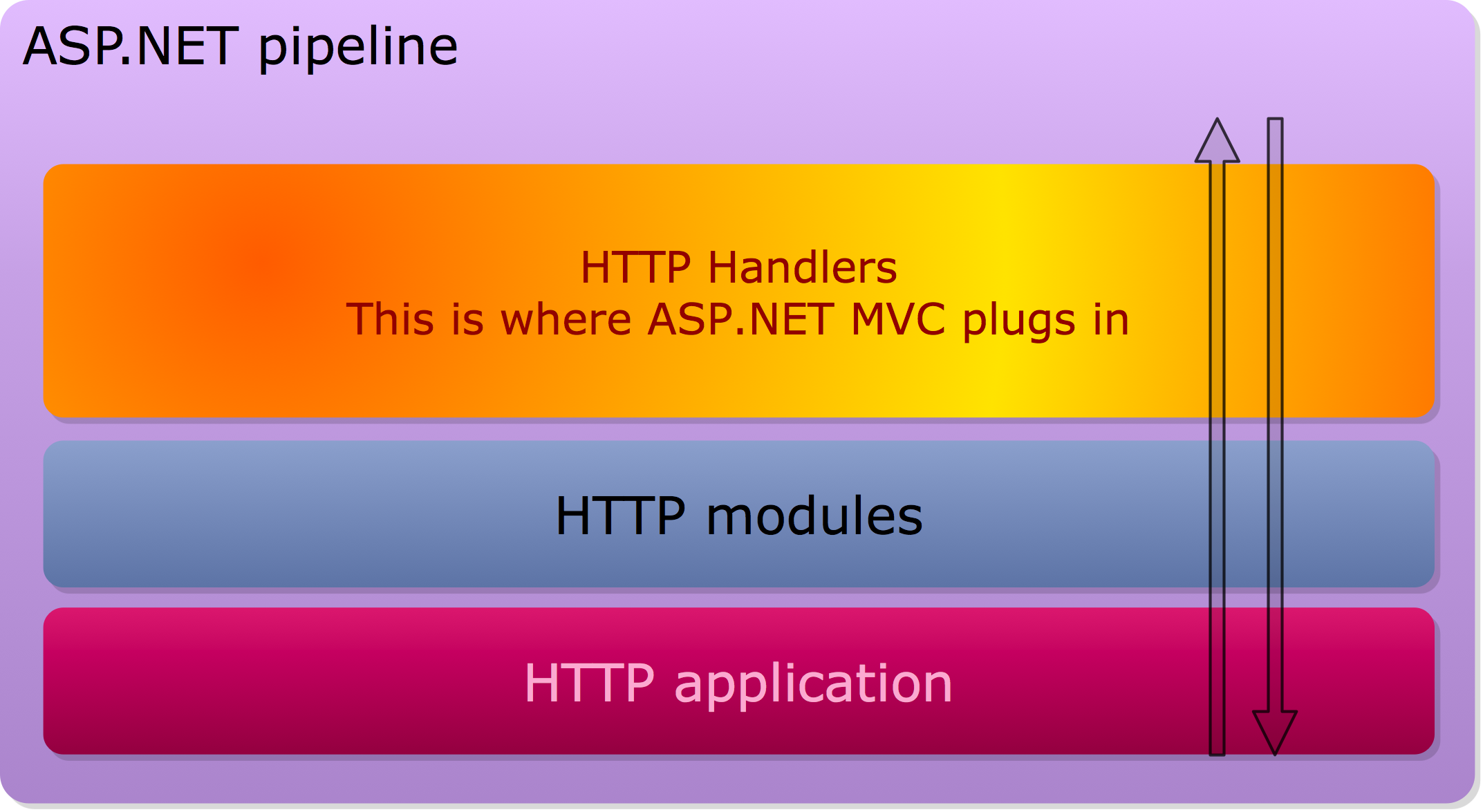
6.1.1 Key extension points in an ASP.NET application life cycle

Every ASP.NET application is built on the same pipeline. That is a request always follows the same path giving you common points to extend an application. Figure 6.1 illustrates those extension points. It’s at these point you can subscribe to certain events in the lifetime of a request to add your own logic to it.

I’d like to use this illustration to explain where ASP.NET MVC is situated in that lifecycle. When a request arrives at the web server and ASP.NET takes over everything is the same as with WebForms except that a different handler is used. Webforms typically used an IHttpHandler implementation called PageHandler, which knows how to operate on a certain base class (Page) and process the logic contained in a classic WebForm. ASP.NET MVC, like WebForms, hooks into ASP.NET by using a Handler. And from then on the ASP.NET MVC framework starts executing.

So before a handler executes the request passes through a series of modules, one such module is responsible for providing the FormsAuthentication infrastructure. Another one is responsible for managing Session state. The request then reaches the ASP.NET MVC handler, a controller gets instantiated and it starts processing the request. When the controller is finished a rendered view is pushed down the pipeline and passes through a set of modules before being sent to the browser.

Figure 6.1: Key extension points for an ASP.NET application

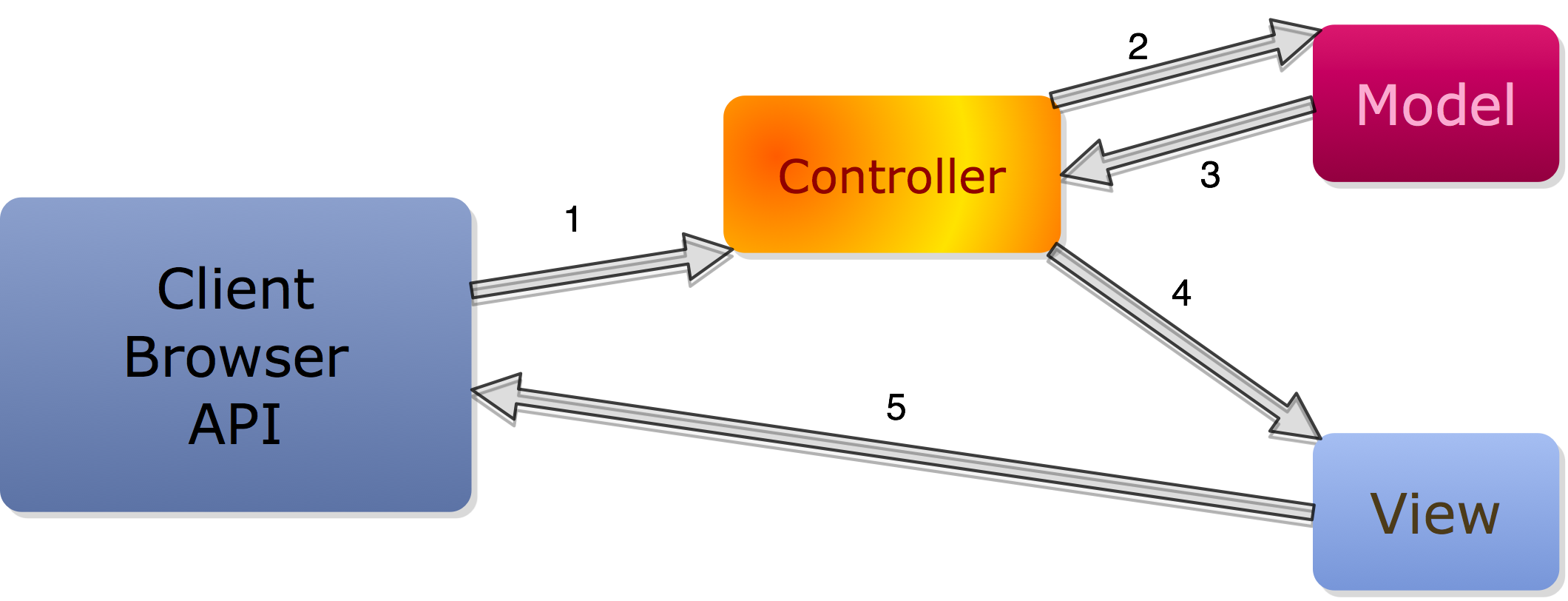


Now that the request pipeline and how ASP.NET MVC hooks into the ASP.NET infrastructure is clear to us, we might have a look at some of the key components in an ASP.NET MVC application. The controller, model and view and how they fit together.

6.1.2: Key components in and ASP.NET MVC application

At this point it might be worth taking a look at the major components that are being used by the handler and that make up the letters M, V and C in the pattern name. Let’s first introduce all the components and then later look at how they fit together in the context of a request. Figure 6.2 illustrates the components and how the control flows between each of them. You will see that controller is a crucial pivotal point in your application.

Figure 6.2: Key components of an ASP.NET MVC application



Routing

When a request arrives at the server, the URL gets mapped to a controller and it’s appropriate action. This process is called routing and the default route that is implemented maps the first part of the path to a controller name and the second part to an action name. So an URL <http://somecoolsite.com/products/list> would map to the controller ProductsController and the action on that controller would be the List method implemented on that controller. After routing the request passes into the controller

Controller

The controller is an orchestrator and takes the input from the client, this can be in a query string, request headers, form variables, … and calls various models to process that input. When the controller has gathered the output it needs it directs its attention to the view and passes the gathered output to the view to render it.

All logic that is responsible for calling out and getting output back for the view belongs in the controller. A controller takes orders, delegates them and delivers them back to you, much like a waiter in a restaurant.

Model

The model is really an object that contains some form of state for your application. This doesn’t mean that the state has to be persisted to the database but it does mean that it does something with data and generally returns a result for one or more properties. This is where the processing belongs nowhere else. Any logic that isn’t directed at orchestrating models to get output for the view belongs in the model, but a little bit more about that later.

View

The view is actually just a means to display a given set of data. This component typically serves to generate html or JavaScript/JSON. So a view is generally built using some kind of templating engine that allows you to create html or JavaScript.

As a general rule of thumb I use: if the logic can’t be expressed by simple if/else statements and for loops it doesn’t belong in the view and as such should be moved to a model preparing it to send to the view.

Control flow

Ok now that we have a grip on what makes up the different components of an MVC application, it’s probably a good time to explain the numbers on figure 6.2.

1. The client sends a request to the web server where it arrives in the handler. The handler functions as a front controller and handles the routing to collect the necessary information for which controller should be instantiated by the controller factory. The handler then invokes the appropriate action, which it also got from the routing handler.
2. The controller orchestrates one or more models and passes input to these models
3. The models return their values to the controller who prepares them to be sent to the view.
4. The controller is finished preparing and aggregating values and is now ready to decide which view it wants to render and in which form. After having done so it passes the necessary data to the view and renders the template.
5. The controller is finished rendering, perhaps does some post-processing and passes the output back to the client.

MVC in real life

As it so happens you probably deal almost daily with some kind of MVC pattern in real life. To get to the full explanation the restaurant would probably be fairly fancy.

You arrive at the restaurant and the maître d’ will show you to your seat. In this case the maître d’ is the front controller, which takes care of the routing. Hopefully he will also send a waiter to your table (the controller). This waiter takes your order (input) and goes off to deliver your order to the kitchen. The kitchen is full of models aka. cooks, they prepare your platters and put them back on the counter. The waiter keeps an eye on that counter and when he sees one of his orders there he goes to pick it up, perhaps takes a pepper crusher (post-processing) with him and delivers the food (view) to your table.

The last topic of discussion in this section is about some of the key objects available to you in the ASP.NET MVC application. Most of them are available in the controller and some of them are reachable in the view too.

6.1.3: Some of the key objects in an ASP.NET MVC application

Just about every web framework you’ll work with will wrap the environment variables provided by the request and web server in some kind of more friendly objects. ASP.NET is no different in that regard. We’ll look at the stock ASP.NET objects and will touch on how ASP.NET MVC uses them in the next section.

HttpApplication

This class is the entry point for your web application and as such encapsulates some application level variables. This class is implemented as a Singleton and calls a magic method Application\_Start once at the start of an application. This class also provides a bunch of methods you can override to hook into the application life cycle.

HttpServerUtility

This utility provides a set of helper methods that make it easier to work with the web server. One of those methods is mapping an absolute URL path to a path on the file system of the server.

HttpContext

This class encapsulates the entire context of the request and response, it also contains the server utility object as well as the session state object. If you want to know something about the request this, its variables,… this would probably be a good starting point to find it.

HttpRequest

I guess you have pretty good idea of what is going to be in this object, but just in case you don’t; this class encapsulates the environment variables that make up a request. Among its properties one finds the query string, form post data, the request URI and such. The controller and the handler use this object most frequently.

HttpResponse

And the last one I want to discuss at this point is the HttpResponse object. This object provides methods to write to the response stream that will be sent to the client. It also has facilities for you to set headers for the response. The view engine uses this object most often.

These components are essential to just about any web application and even if you don’t use them directly it pays off to understand that the internals of your particular framework probably use them to some degree. Now that we know some of the major components of ASP.NET you’re probably asking yourself why would I want to use ASP.NET MVC?

6.2 Why, oh why ASP.NET MVC?

Many developers in the .NET sphere that I talked with before ASP.NET MVC 1.0 had been released didn’t know why they would want to use ASP.NET MVC over standard ASP.NET WebForms. The second problem they face is that most of the MSDN guidance to date had been written from the point of view of an n-tiered application where you have ie.: a data and a business logic tier as well as a presentation tier. When we talk about the MVC architecture the tiers still apply but are distributed differently, this is not better or worse it’s just different. For example your data logic will most definitely be contained in the models as well as most of your business logic. Some of that business logic may spill into the controller but you should be careful with that.

While ASP.NET WebForms have been used to create some pretty large web applications successfully it’s not without its pain points. We’ll look at the areas where ASP.NET alleviates some of these pain points and how they are solved. The aim of this section is not to say how webforms are bad that discussion has been held on the Internet and is for you to decide.

6.2.1 Unit testing

ASP.NET MVC wraps most of the basic objects of a web application in classes so they can be stubbed or mocked out easily. This has a big impact on how you can develop your application in a test-driven way. You can now test the application from unit tests without requiring a webserver to provide you with the necessary objects. We’ve highlighted the importance of unit testing at the beginning of this book. Whatever makes it easier for me to feel warm and fuzzy I’ll take!

Unit testing is also a lot easier because of the clear separation of concerns enforced by the MVC pattern.

6.2.2 UI driven development

There is a trend in web development that uses a UI driven approach to the development process. They start by developing the interface with mocked data and once everybody is happy with the UI they start writing the code that is necessary to make the interface work.

This is probably the more efficient way of building an application, because during the design phase the only thing that needs to be changed is html and perhaps some mock data. These things are much easier to change than say a database schema or the way billing works. This works particularly well in MVC because the mock data can be created in the controller and the html can be done in the view templates and reused later.

6.2.3 Accommodation of AJAX web applications

ASP.NET MVC makes it very easy to respond with different output formats (JSON, HTML, XML, …) to a request depending on certain variables etc. We’ll look at an example of this later in this chapter. This means that the degree of potential code reuse is pretty high because you can use the exact same code and data but give it any number of different representations.

6.2.4 Flexibility

An overall benefit of using MVC is that you get a lot more control over how your application is built and how it does certain things rather than using a bunch of black box components that are outside of your control. An MVC application makes it easier to create loosely coupled applications and create flexible designs. Then there are some other benefits that we should look at.

ASP.NET MVC in particular lets you change the view engine. It allows you to replace the controller factory. It basically allows you to replace any one of its components by a custom implementation, making this a platform you can truly shape to be your own.

These are some of the advantages that ASP.NET MVC has to offer. At the time of this writing there is no WebForms story for IronRuby but there will be one short after IronRuby gets released. So I wanted to inform you about the choices ahead without the ability to provide samples for the WebForms version. Of course there are some downsides to ASP.NET MVC as well which is what we’ll look at next.

6.3 All roses? Any of them thorny?

The previous section highlighted some of the benefits of using ASP.NET MVC but to give a balanced explanation I ought it important to also look at some of the downsides. Depending on your POV or your objective some of the advantages of ASP.NET MVC are in fact disadvantages. This part aims at putting some of these items forward.

6.3.1 Controls or lack thereof

Most of the Microsoft/.NET technologies involve event-driven programming. Almost everything in the UI triggers an event that gets handled by the code-behind. This is a good solution if you’re working in a stateful environment. To ease the transition from desktop to web development you may opt to choose for a solution that also supports this pattern of event-driven programming.

Something that is also common in most GUI technologies provided by the .NET stack is that they work with widgets in the form of controls that expose certain events to so you can program against them. These controls are absent in the ASP.NET MVC framework, a lot of the lack of controls gets made up by the possibility to write helper methods to generate html for u in a reusable way and across projects.

6.3.2 RAD story or code volume

One of such advantages is the fact that you have more control. More control means that you probably have to do some plumbing. Depending on your goal it might be a good idea to go for something that requires virtually no coding. Maybe you need something really quick that you can throw away later. Perhaps you’re application doesn’t do anything except show data from a database. Or maybe you just don’t want to write any code. In any of the aforementioned cases the WebForms alternative or dynamic data might be a good solution for you. We’ll look at another case next, the case in which you have an existing codebase written in ASP.NET WebForms.

6.3.2 Existing codebases

You or your company may already have made a serious investment in the ASP.NET WebForms platform in the last few years. Perhaps you want to migrate but can’t justify an entire rewrite of the application. New functionality and pages can be added to the ASP.NET WebForms application but using ASP.NET MVC controllers and view because you can use both side by side. You may even agree to move existing pages to the MVC model gradually by moving them over whenever they need to be touched. There is a special case that we haven’t discussed yet and that is the case in which you already have existing ASP.NET MVC components but they are written in C#.

6.3.3 Existing CLR ASP.NET MVC codebases

The way IronRuby MVC is implemented makes it compatible with existing components created in another language like C#. That means you can reuse all of your existing logic as is but you can add new logic using the wonderful Ruby language. This has proven to be an advantage when working with a CLR OR/M tool like we’ll be doing in this chapter.

Phew this again concludes the theoretic base for this chapter and we can now move on to a more practical exploration of the ASP.NET MVC framework with IronRuby. During the remainder of this chapter we’ll build a very simple chat application that updates the conversation using AJAX.

6.4 A practical exploration with MockChat

So far this chapter has been primarily theoretical, it would also be presumptuous to say that this is all you’re going to need to become an expert at ASP.NET MVC. There are many excellent books about ASP.NET MVC whose content is also valid for IronRuby (but the samples will most likely be in C#).

In this exploration we’re not going to follow the interface driven approach but instead will follow this order Model – Controller - View when talking about the MVC implementation. Why? Otherwise we’d have to jump around in our code too much, which would make this exploration pretty hard to follow. So instead we’re going to take a much more sequential approach to writing this application. Let’s start by getting things set up.

6.4.1 The setup

Most MVC applications use a database to persist the state of most of their models. Our application is no different in that regard. We also use a database to persist our state. In this case we’re going to use a sqlite database. Sqlite is a very lightweight database system that holds up well when you don’t have too many users or data to manage (in the regions of 2GB). This makes it almost perfect to use during development as there is no setup involved all you need is the library to talk with the database file.

Once you have a database you probably are going to want to map the database tables to your model objects. Typically one would use an OR/M to handle that mapping for you. There are many good reasons for using an OR/M and not rolling your own mapping layer but that is out of the scope of this book. The OR/M tool we’re going to use is LightSpeed, there is a free version of their tool available which is more than sufficient for what we’re going to need from it in this chapter.

One of the reasons I picked LightSpeed for this chapter was the fact that with little or no coding I get validation, it has a good visual designer and it is really simple to use. In addition LightSpeed allows me to either create my models from the database or use their designer to create the models and then have it generate the tables for me in the database. The last thing that influenced my choice was that LightSpeed has built in support for full text searching through Lucene. These features make it a pleasure to work with for me. I should probably also disclose that the guys that wrote this software are personal friends of mine, so there might be some bias involved too.

These tools cover the models end. On the views end we’re going to make use of JQuery as our javascript library of choice. Again we won’t cover any ground on how to work with jquery because there are some other books that will do a far better job at that. In addition to the base library from jquery we’ll also make use of 2 jquery plugins to make our life a little easier with regards to making some ajax calls.

The tools you’re going to need for this setup are scattered over the net. Let’s compile a list of where to get, install and configure them.

Getting all the tools you’ll need

The first tool on our list is of course ASP.NET MVC itself. We will use ASP.NET MVC 1.0 in this sample as that is the official release at the time of this writing. You can get the download package in the Microsoft download center (<http://www.microsoft.com/downloads/details.aspx?FamilyID=53289097-73ce-43bf-b6a6-35e00103cb4b&displaylang=en>). Make sure your visual studio is closed and run the setup.

The second tool on the list is the visual studio integration for sqlite

Next up lightspeed

And at last JQuery and plugins.

7.4.2 Creating the solution

7.4.3 Routing in a nutshell

7.4.4 The first controller

7.4.4 Configuring our models

7.4.5 A controller duet

7.4.6 The chat view

7.5 Common gotcha’s

7.6 Summary