ASP.NET MVC is not designed to stand alone. As a web development framework, it inherits much of its power from the underlying ASP.NET platform, and that in turn from the .NET Framework itself (Figure 17–1).

Figure 17–1. ASP.NET MVC builds on more general infrastructure.

Even though ASP.NET MVC’s notions of controllers, views, and filters are flexible enough to implement almost any piece of infrastructure you’ll need, to stop there would be missing the point. A good percentage of your work is already done out of the box if only you know how to leverage ASP.NET’s built-in raft of time-saving facilities. There are just two problems:

- **Knowing what's there**: We’ve all done it—you struggle for days or weeks to invent the perfect authentication or globalization infrastructure, and then some well-meaning colleague points out that ASP.NET already has the feature; you just need to enable it in `Web.config`. Curses!

- **This ain't Web Forms**: Much of ASP.NET’s older infrastructure was designed with Web Forms in mind, and not all of it translates cleanly into the MVC world. While most platform features work flawlessly, others need the odd tweak or workaround, and some just don’t work or aren’t applicable.

The goal of this chapter is to address both of those problems. You’ll learn about the most commonly used ASP.NET platform features that are relevant in an MVC application, as well as the tips and tricks needed to overcome compatibility problems. Even if you’re an ASP.NET veteran, there’s a good chance you’ll find something you haven’t used yet. This chapter will cover the following:

- Authentication—both Windows Authentication and Forms Authentication mechanisms
- The Membership, Roles, and Profiles facilities
Authorization
• Configuration
• Data caching
• Site maps (for navigation)
• Internationalization
• Features for monitoring and improving performance

Just one thing before we get started—this chapter doesn’t attempt to document all of these features in full detail—that would take hundreds of pages. Here, you’ll see the basic usage of each feature in an MVC context, with discussion of any MVC-specific issues. It should be just enough for you to decide whether the feature is right for you. When you decide to pursue a particular feature, you may wish to consult a dedicated ASP.NET platform reference. I would recommend *Pro ASP.NET 4 in C# 2010*, by Matthew MacDonald (Apress, 2010).

**Windows Authentication**

In software terms, *authentication* means determining who somebody is. This is completely separate from *authorization*, which means determining whether a certain person is allowed to do a certain thing. Authorization usually happens after authentication. Appropriately, ASP.NET’s authentication facility is concerned only with securely identifying visitors to your site, setting up a security context in which you can decide what that particular visitor is allowed to do.

The simplest way to do authentication is to delegate the task to IIS (but as I’ll explain shortly, this is usually only suitable for intranet applications). Do this by specifying Windows Authentication in your Web.config file, as follows:

```xml
<configuration>
  <system.web>
    <authentication mode="Windows" />
  </system.web>
</configuration>
```

ASP.NET will then rely on IIS to establish a security context for incoming requests. IIS can authenticate incoming requests against the list of users known in your Windows domain or among the server’s existing local user accounts, using one of the following supported mechanisms:

*Anonymous*: The visitor need not supply any credentials. Unauthenticated requests are mapped to a special anonymous user account.

*Basic*: The server uses RFC 2617’s HTTP Basic authentication protocol, which causes the browser to pop up an Authentication Required prompt into which the visitor enters a name and password. These are sent in plain text with the request, so you should only use HTTP Basic authentication over an SSL connection.

*Digest*: Again, the server causes the browser to pop up an Authentication Required prompt, but this time the credentials are sent as a cryptographically secure hash, which is handy if you can’t use SSL. Unfortunately, this mechanism only works for web servers that are also domain controllers, and even then it only works with Internet Explorer.

*Integrated*: The server uses either Kerberos version 5 or NTLM authentication to establish identity transparently, without the visitor having to enter any credentials at all. This only works transparently when both the client and server machines are on the same Windows domain (or