The .NET Framework has remained very vibrant since the introduction of .NET 1.0 in early 2002. Over the years, along with the major releases of .NET, Microsoft has added new languages, features, and technology sets. It is very clear that a lot of planning and consideration has gone into the .NET Framework. The same thoughtfulness is apparent in the evolution of the C# language. With each release, the C# language has added new and powerful constructs that have improved the language and the way developers write code in C#.

With C# 2.0, the language designers introduced new features, such as

- Generic types and methods
- Nullable types
- Simplified delegates and anonymous methods
- Iterators and the yield statement
- Partial classes

With C# 3.0, the following new features were added:

- Implicitly typed local variables
- Anonymous types
- Extension methods
- Lambda expressions
- Query syntax and LINQ

With C# 4.0, the language brought more features:

- Optional parameters and named arguments
- Dynamic binding and the Dynamic Language Runtime (DLR)
- Covariance and contravariance

This chapter will review some of the powerful C# constructs, as they relate to following best practices. The most important concept is that effective C# development is a key part of following .NET practices. Take the time to dive into the C# language to learn how to take full advantage of the language. Table 7-1 lists the ruthlessly helpful practices that take advantage of some of the powerful C# constructs.
Table 7-1. Powerful C# Constructs: Ruthlessly Helpful Practices

<table>
<thead>
<tr>
<th>Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-1</td>
</tr>
<tr>
<td>7-2</td>
</tr>
<tr>
<td>7-3</td>
</tr>
<tr>
<td>7-4</td>
</tr>
<tr>
<td>7-5</td>
</tr>
<tr>
<td>7-6</td>
</tr>
<tr>
<td>7-7</td>
</tr>
<tr>
<td>7-8</td>
</tr>
<tr>
<td>7-9</td>
</tr>
<tr>
<td>7-10</td>
</tr>
</tbody>
</table>

Extension Methods

Extension methods are a special kind of static method. They allow you to write methods as if they were instance methods on an existing type. This approach does not involve creating a derived type, nor does it involve recompiling or modifying the original type. The extension method is called as if it is a method on the type it extends and appears (even through Visual Studio IntelliSense) as if it is actually defined as a method of the type.

Let’s take a look at the power of extension methods by way of an example. In the example system there is different logic that is driven by the current day of the week. There is a frequent need to know if today’s date is a weekday or a weekend. By using the DayOfWeek property of System.DateTime it is simple to properly branch the logic. Listing 7-1 shows a typical section of code with logic that repeats throughout the codebase.

Listing 7-1. Repeating Code That Acts on a DateTime

```csharp
datetime importantDate = new DateTime(2011, 5, 7);
switch (importantDate.DayOfWeek)
{
    case DayOfWeek.Saturday:
    case DayOfWeek.Sunday:
        WeekendProcessing();
        break;
    default:
```