Disadvantages:

- Highly-coupled GUI and data access code.
- Only good for small applications with no business logic.
- Scalability issues—if the project grows big, then performance will be affected.
- Not efficient when dealing with complex hierarchical result-sets.
- Because they abstract data access operations, they are not very flexible and may present problems when dealing with customized data.

A simple guestbook application for a personal website is one such case warranting the use of data controls. Such applications have quick turn-around time, and one can build a working application in a matter of few days. If we do not use Data Source Controls, then we can directly write custom data access code in the code-behind file. Although we will be putting non-UI related code in the UI layer even then, we will have more flexibility as we can write custom SQL statements and modify data at runtime, which is quite difficult to do using Data Source Controls.

To overcome the disadvantages of data controls, Microsoft introduced Object Data Source control, to support business objects, and business logic code and the N-tier architecture. We will learn more about object Data Source Controls in the next chapter.

Summary

In this chapter, we examined how web applications inherently follow a 3-tier client-server model, and how a 1-tier architecture can be used for simple applications in ASP.NET. Then we learned how we can logically partition this basic 1-tier 1-layer architectural style into two sub-layers, using code-behind files. We also studied how declarative code-less programming using new Data Source Controls follows a single-layer style.

In a nutshell:

- Classic inline coding should not be used unless absolutely necessary. One case supporting it would be a project that mixes classic ASP and ASP.NET, or a project already built using this style of coding.
- Data Source Controls (except Object Data Source) are only good for small projects which will never need to be scaled up in the future. For commercial-level projects, it's very important to logically break the code into layers.

- The code-behind style is much more flexible, object-oriented, and scalable for commercial projects. We can break these layers further, into more layers, for better code management and maintainability, as we will see in the coming chapters of this book.

We saw that although we have separated the HTML and code elements, all of the code is still in a single code-behind class (1-tier 1-layer style). For a decently-sized commercial application, having data access and business logic code in the same code-behind file is not a good practice, as code-behind belongs to the UI layer. For further loose coupling, the UI layer should handle only the UI events and should not contain data access or any business logic code. In the coming chapters, we will learn how to break up this 1-tier 1-layer architecture into an n-tier and n-layer architecture so that we can achieve a higher degree of loose coupling, and make our applications more scalable and robust.