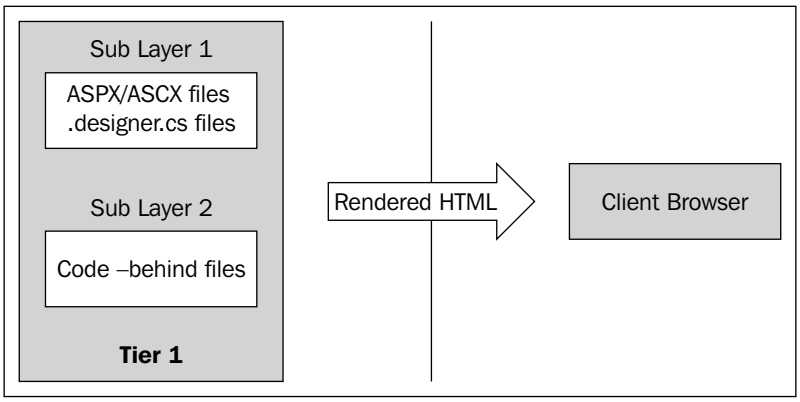
UI Layer



From the given diagram, we can see that we still have one single physical DLL, but the UI code itself is logically separated into two layers—one in the markup code and the other in the code-behind file. Note that, as explained in the beginning of this chapter, this architecture is 1-tier from the application's viewpoint, but overall it is still 3-tier if we consider the client browser as the presentation tier, and the database as the data tier. We are only focusing on the application tier, which houses the UI layer in the above examples.

Limitations of Coding in the UI Layer

Even though we have only one layer, the separation between HTML and UI code, using code-behind files, helps the web designers to work independently of the developers. Both have separate files to work on. This is a recommended practice, as mixing UI markup with logical UI processing code can lead to a spaghetti mixture. Besides this furthermore, the style is more object-oriented than inline coding as we write code in code-behind classes instead of writing it in interpreted free-style blocks as in classic ASP. So code-behind helps us manage and maintain our code in the long run, when compared to inline coding.

However, note that even though we have separated the code into two layers, both of these layers actually belong to the UI layer. Even if we are putting data access or business logic code in the code-behind files, we are still mixing the UI layer with non-UI code, which is not recommended for commercial scalable applications. In the coming chapters you will learn how to further use layers and tiers to make your application more scalable.

But if your project is small, for example, a 4–5 page website for personal use or a small data entry website which is not intended to grow in size, complexity, or user base, and where there is no future need to scale it up, then putting the data access code in the code-behind files is acceptable. Remember—"if it ain't broke, then don't fix it". There is no need to complicate a simple web application if there is no actual requirement to justify doing so. Designing scalable solutions from the start is a good approach, but that does not mean we need to start over-architecting every website we work on without thinking twice about its real use and application. Furthermore, budget constraints sometimes might not allow us to adopt a fully-fledged N-tier scalable solution because the project is either too small or there is no scope for it to grow further, and therefore the project stakeholders might not want to spend too much on its development because building a scalable architecture takes time and will not make economic sense for a small project.

Next, we will learn about Data Source Controls, and how we can put them to best use for small projects.

Data Source Controls

With further refinements to ASP.NET 2.0, Microsoft has added many out-of-the-box controls (apart from the standard web control library). Some of the most useful controls are Data Source Controls, complimenting the feature-rich web server controls such as the GridView and the DetailsView. These controls have made it possible to create applications without writing even a single line of data access code. Now it is possible to create web applications within a short timespan, doing away with many lines of routine data access code.

With Data Source Controls, we can use SQL queries as well as stored procedures, and write custom code too. Although we won't go deep into the details of how to use controls, as there are many freely-available online resources and articles on this subject, we will see how using these controls affects the overall architecture, when to use them and what their disadvantages are.

A Sample Project using Inbuilt Data Source Controls

Let's start with our good old guestbook application! This time we will see how to insert data using data controls with little or no programming. Here is how the form looks: