

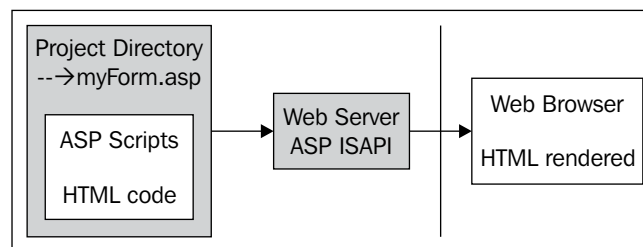
In this chapter, in order to simplify the understanding of application architecture, we will be considering tiers from the application's stand point and therefore ignoring the database (data tier) and client browser (presentation tier). So a single ASP.NET web application, in monolithic terms, is 1-tier. We will see how to break this 1-tier 1-layer web application further into layers and tiers, understanding and analyzing the needs and effects of each step.

Usually, it takes a lot of experience working with different types of architectures to become familiar with the advantages and disadvantages of each approach. A developer who has worked only in 3-tier (or higher) applications may find it very difficult to conceptualize and adapt to a 2-tier approach even though it may be more suitable for his project. He will feel more comfortable in the n-tier based architecture even when it is not required. That is why it is very important to study the 1-tier and 2-tier designs and analyze their pros and cons, to appreciate the usefulness and the real need of breaking it further into multiple tiers and layers.

In this chapter, we will focus on how to break the monolithic default ASP.NET architecture into multiple layers and tiers and see when and where to use this style. We will also see how we can logically break the 2-tier style into different layers for more flexibility and better code management.

## Classic ASP Style: Inline Coding

Firstly, we will study the classic inline style of coding, which was the only option available during the good old ASP 3.0 days. This was a mix of interpreted ASP scripts and HTML code. In terms of architecture, there was not much flexibility, although developers used classes to bring some object oriented flavor to the projects, but these were not pure OO classes. Core features such as inheritance were not supported. Moreover, there was lot of effort involved in coding these classes, so most developers preferred to mix coding that was much faster in terms of development time. At a high level, an ASP project configuration would usually follow the given diagram:




In this diagram, we have ASP script files in the web server directory being processed by ASP ISAPI (**I**nternet **S**erver **A**pplication **P**rogramming **I**nterface) DLL in IIS and rendered in the client browser. ISAPI DLL is a part of IIS itself, and not a separate process such as the ASP.NET runtime engine.

Here is some classic ASP sample code:

```
<%@ language="JScript" %>
<% var    pubName ="Publisher",
        pageTitle="Publisher Page Title"
        Response.Write(pubName) %>

<html>
<head><title><% =pageTitle %></title>
</head>
<body>
<div> <% =pubName %> </div>
</body>
</html>
```

The above ASP code is a simple example which clearly highlights the fact that the pre-ASP net coding style was a messy mixture of HTML and ASP scripts. In this particular style, we had no logical or physical separation of the web application code. It followed a single-layer style—everything was done in the UI layer (which is a part of the application tier, and is different from the presentation tier). With none of the indispensable modern day programming features such as debugging support and IntelliSense, maintenance of such code was nearly a nightmare.

 Programming languages are either compiled or interpreted. ASP code was interpreted line by line, unlike modern higher-level compiled languages such as C++ and C#. Because interpreted code needs to be converted line by line into machine code at runtime, it is usually slower than compiled code, where the entire program is converted into machine instructions in one batch, typically long before any of it is run.

Then came ASP.NET, doing away with the interpreted ASP scripts and introducing a much faster compilation model along with strongly-typed languages such as C# and VB.NET, in addition to numerous other benefits, making it a strong leap from ASP.

Although not recommended, ASP.NET still allows the use of the inline coding model using <script> block constructs for C# and VB.NET code. We don't need to go deeper into inline coding, but here is a simple example of how an ASP 3.0 developer might have intuitively coded a simple project in ASP.NET without using any code-behind.