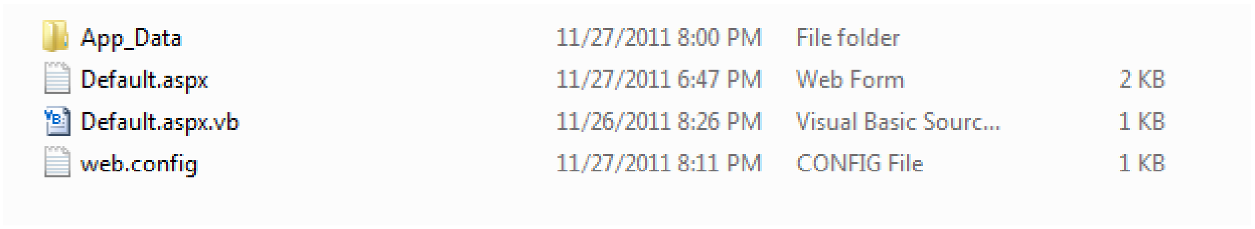
Bill Pegram
December 2, 2011

Deploying ASP.NET Database Application to a Shared Windows Hosting Site – Part 1

I created a simple ASP.NET application that connected to a SQL Server database following the example in *Teach Yourself Asp.NET in 24 Hours*, pp. 577-581. The files in the application were as shown below:



App_Data	11/27/2011 8:00 PM	File folder	
Default.aspx	11/27/2011 6:47 PM	Web Form	2 KB
Default.aspx.vb	11/26/2011 8:26 PM	Visual Basic Sourc...	1 KB
web.config	11/27/2011 8:11 PM	CONFIG File	1 KB

The .mdf SQL Server database is located in the App_Data folder.

I followed the instructions provided at pp. 581-590 to successfully deploy the application to a shared Windows Hosting site. The 3 steps required were as follows:

1. FTP the files (with the exception of the .mdf database file) to a folder (e.g. www.billpegram.com/deploy - this particular example also works if one isn't at the top level, e.g www.billpegram.com/test1/deploy but my understanding is that is not generally the case)
2. Replicate the local SQL Server database on the hosting site – In the Hosting Control Center for my website (hosted at www.godaddy.com), I created a SQLServer database specifying username and password. The GoDaddy setup procedure asks whether you want to be able to access the database externally (specify yes if you want to be able to view your tables and data on the server through the SQL Server Management Studio running on your local PC but this is not necessary in order to deploy your application).

   To replicate the database, I scripted the local database and then executed the script on the server. In Visual Web Developer, go to the Database Explorer, right-click on the data to publish, and choose the Publish to Provider option. After specifying the database to publish, choose the Script to File option.

   In the Hosting Control Center for my hosting provider, there was a Query Analyzer provided for SQL Server and using this Query Analyzer, I executed the script I had created from the local database which then recreated the tables and data.

3. Updating the Connection String in web.config

The web.config file in the local application points at the .mdf file locally; one needs to change this to point at the database on the server. My hosting provider provided several different ways to connect – I used the SQL SqlConnection .NET string approach to create for the following config.sys

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

  <system.web>

    <customErrors mode="Off" />

  </system.web>

  <connectionStrings>

    <add name="ConnectionString" connectionString="Data
Source=anct107com.db.3457421.hostedresource.com; Initial Catalog=xxx; User ID=yyy; Password='zzz';"
providerName="System.Data.SqlClient" />

  </connectionStrings>

</configuration>
```

where xxx is the database name, yyy the username, and zzz the password. The DataSource URL was available to me through the Hosting Control Center for my website.

The   <customErrors mode="Off" /> can be very useful when debugging an application. By default, in IIS7, descriptive error messages are not shown because of security concerns that a descriptive error message can convey information about the inner workings of the application. In classic ASP, there is no way to change this IIS7 behavior (so one has to use IIS6), but in ASP.NET for IIS7 one can elect to show error messages in a variety of ways – e.g. detailed locally but general remotely, email detailed messages, show detailed locally and remotely, etc. The customErrors mode="Off" specification will show detailed error messages locally and globally.

In trying to get the application deployed, a syntax error in the connection string prevented the customErrors = "Off" specification from working and thus I only saw a non-specific error message until I had fixed the connection string so special care needs to be taken for this step.

Part 2 of this writeup will discuss deployment for an application involving membership or roles and where the application involves a number of folders.