Store Documents in on-line Briefcase (Part 1)

By Bipin Joshi
www.binaryintellect.net

Web sites often store files directly in some folder of the file system. However, this may not be the most suitable way in all scenarios. Many of you might have used Yahoo Briefcase. It allows you to store any file on the server so that you can access it from anywhere. If you wish to provide such a facility in your web site then file system based storage and retrieval is tedious. Enforcing security and managing storage space quota becomes difficult. More elegant solution can be developed by storing files in database. This way you have full control on who is storing and retrieving files, storage space and usage statistics. In this two part article we are going to develop a web based briefcase application using ASP.NET 2.0 and SQL Server 2005. In this part we kickoff by creating database tables, stored procedures and classes to manage files and folders.

Functionality

Before we begin the development let's decide the functionality expected from the briefcase application.

- The application should allow to manage files and folders via a web based interface
- The application should allow us to create folders and sub folders
- There should not be any limitation on the nesting levels of the folders
- The application should allow us to delete and rename the folders
- The application should allow us to upload files to a specific folder
- The file can be downloaded at any time by navigating to that folder
- We should be able to delete or rename the file
- The creation date and size of the file must be tracked
- If a folder is deleted then its subfolders and files should also be deleted
- Just like file system the application must ensure that there are no duplicate file or folder names in a given scope

Software and Technology Features Used

In order to develop the briefcase application we will use the following software:

- ASP.NET 2.0
- TreeView and GridView web server controls
- ADO.NET 2.0
- Visual Studio 2005 (VWD Express Edition can also be used)
- SQL Server 2005 along with Management Studio

Database Design

To begin with let's define the structure of our database. Open SQL Server 2005 management Studio and create a new database named BriefcaseDb. The Figure 1 shows the New Database dialog of SQL Server 2005 Management Studio.

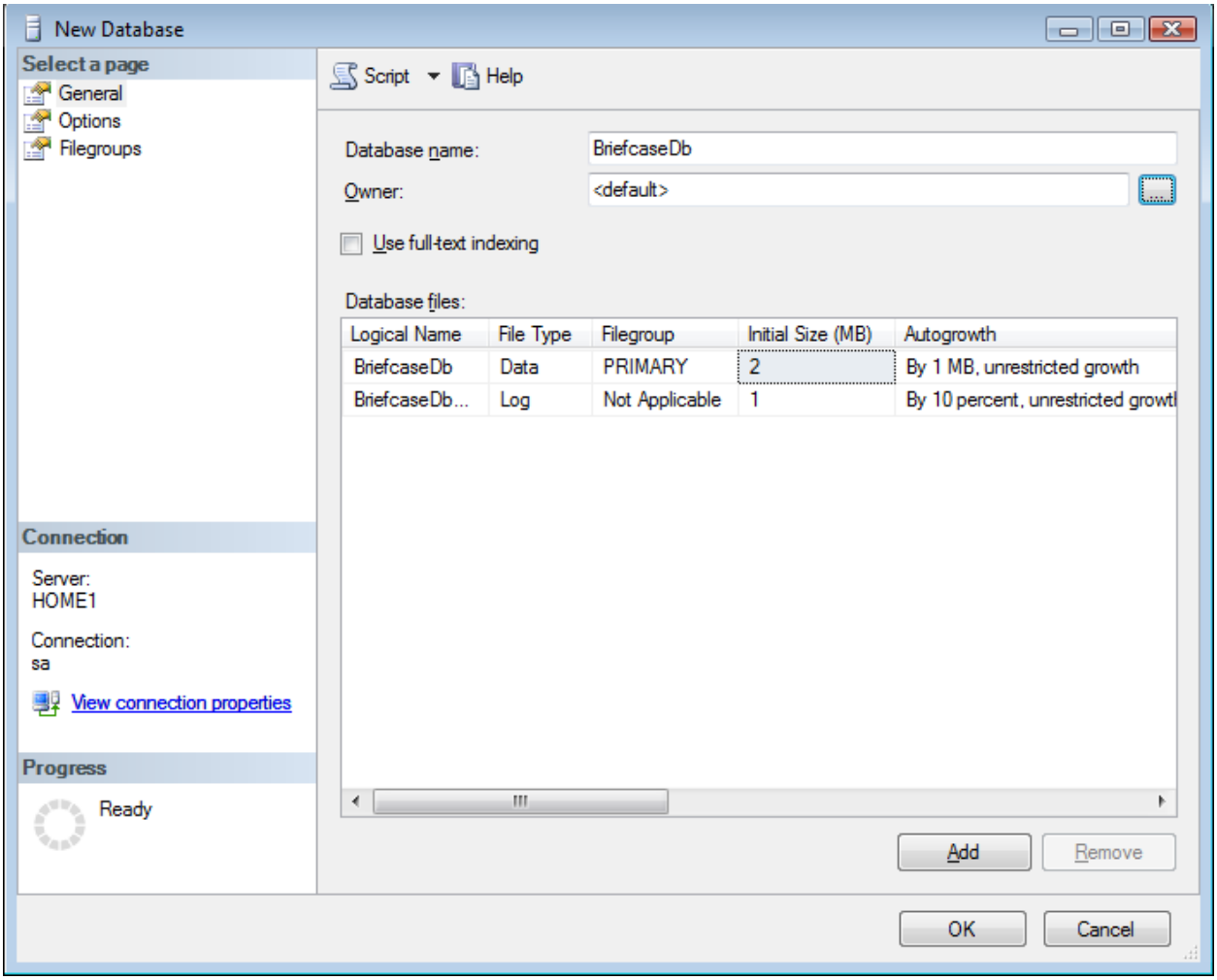

Figure 1: Creating a new database

The database needs to have two tables viz. Folders and Files. These tables will store folders and files respectively. The Figure 2 shows these two tables.

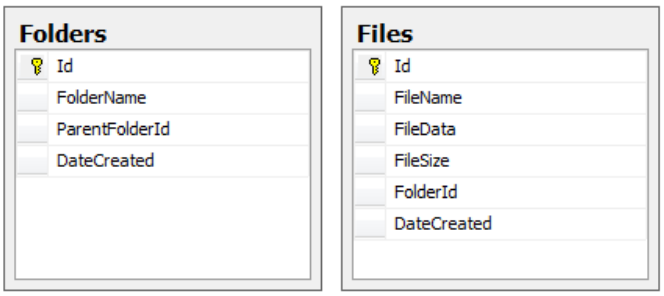

Figure 2: Folders and Files tables

These tables need certain unique keys. The Figure 3 gives the complete T-SQL script to create these tables:

```
CREATE TABLE [dbo].[Folders]
(
     [Id] [int] IDENTITY(1,1) NOT NULL,
     [FolderName] [varchar](255) COLLATE
     SQL_Latin1_General_CP1_CI_AS NULL,
     [ParentFolderId] [int] NULL CONSTRAINT
     [DF_Folders_ParentFolderId]  DEFAULT ((0)),
     [DateCreated] [datetime] NULL,
     CONSTRAINT [PK_Folders] PRIMARY KEY CLUSTERED
     (
          [Id] ASC
     )
     WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY],

     CONSTRAINT [IX_Folders] UNIQUE NONCLUSTERED
     (
          [ParentFolderId] ASC,
          [FolderName] ASC
     )
     WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
)
ON [PRIMARY]

CREATE TABLE [dbo].[Files]
(
     [Id] [int] IDENTITY(1,1) NOT NULL,
     [FileName] [varchar](255) COLLATE
     SQL_Latin1_General_CP1_CI_AS NULL,
     [FileData] [image] NULL,
     [FileSize] [int] NULL,
     [FolderId] [int] NULL,
     [DateCreated] [datetime] NULL,

     CONSTRAINT [PK_Files] PRIMARY KEY CLUSTERED
     (
          [Id] ASC
     )
     WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY],

     CONSTRAINT [IX_Files] UNIQUE NONCLUSTERED
     (
          [FileName] ASC,
```

```
            [FolderId] ASC
      )
      WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
)
ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

Figure 3: T-SQL Script to create required tables

Notice the above script carefully. In addition to primary key constraints it also has certain unique key constraints. The first unique constraint is on FolderName and ParentFolderId columns of Folders table. This constraint ensures that one subfolder names are unique in one folder. Similarly the other unique constraint is on FileName and FolderId columns of Files table. This unique constraint ensures that file names are unique in a given folder.

The columns and their significance of both the tables are summarized in the Figure 4.

Table Name	Column Name	Column Specifications	Description
Folders	Id	Int, Identity	Represents a unique identifier of a folder.
	FolderName	Varchar (255)	Name of the folder. Maximum length can be 255 but can be changed as per your requirement.
	ParentFolderId	Int	This column contains 0 then it is a root folder else subfolder of the folder whose Id is equal to ParentFolderId.
	DateCreated	DateTime	Date and time at which the folder is created.
Files	Id	Int, Identity	Represents a unique identifier of a file.
	FileName	Varchar (255)	Name of the file. Maximum length can be 255 but can be changed as per your requirement.
	FileData	Image	The contents of the file.
	FileSize	Int	Size of file in bytes.
	FolderId	Int	Id of the folder in which the file is stored.
	DateCreated	DateTime	Date and time at which the file is created.

Figure 4: Table schema

Creating Stored Procedures

We will now create some stored procedures for getting the data in and out of the database tables we just created. The Figure 5 lists all the required stored procedures and purpose of each.

Stored Procedure Name	Description
Files_Create	This stored procedure adds a new file to Files table
Files_GetFile	This stored procedure returns details about a file from Files table based on its Id
Files_GetFromFolder	This stored procedure returns a list of files belonging to a particular folder from the Files table
Files_DeleteFromFolder	This stored procedure removes all the files from a specified folder. This is accomplishes by deleting all the records from Files table for that folder.
Files_Delete	This stored procedure removes a file based on its Id by deleting its records from Files table
Files_Rename	This stored procedures renames a file by updating its record in Files table
Folders_Create	This stored procedure creates a new folder by inserting a record in the Folders table
Folders_GetSubFolders	This stored procedure returns all the folders that are subfolders of a specified folder. This is accomplished by returning all the records from Folders table for that matching ParentFolderId
Folders_Delete	This stored procedure deletes a specified folder by deleting its record from Folders table
Folders_DeleteSubFolders	This stored procedure deletes all the subfolders of a specified folder by deleting all the records matching the ParentFolderId
Folders_Rename	This stored procedure renames a folder by updating its record in Folders table

Figure 5: List of stored procedures

As you can see the stored procedures listed in Figure 5 affect Files and Folders tables from the BriefcaseDb database. The stored procedures having prefix of Files_ affect Files table whereas the stored procedures having prefix of Folders_ affect Folders table. The complete T-SQL script to create the stored procedures as listed in Figure 5 is included along with the download associated with this article. The following sections give a overview of the logic involved in creating, renaming, deleting and retrieving files and folders.

Stored procedures for creating files and folders

Stored procedures that create files and folders simply add a new record in Files and Folders respectively. The Files and Folders tables contain identity column Id. This way each newly created file or folder has a unique identifier. This identifier is used later while renaming or deleting the records.

Stored procedures for renaming files and folders

Renaming a file or folder involves updating the corresponding record from Files and Folders tables respectively. The record for a file or folder is updated based on its Id column. For renaming a file FileName column of Files table is updated with the new file name whereas for renaming a folder, FolderName column of Folders table is updated with the new folder name.

Stored procedures for delete files and folders

Deleting files or folders involve deleting records from Files and Folders tables respectively. The record is deleted on the basis of Id column. When a folder is deleted all the files belonging to it must also be deleted. This is done by deleting all the records from Files table where FolderId column value is same as the Id of the folder being deleted. Further, when a folder is deleted all its subfolders must also be deleted. This is accomplished by deleting all the records from Folders table where ParentFolderId column value is same as the Id of the folder being deleted.

Stored procedures for retrieving files and folders

Retrieving files or folders call for executing a SELECT statement against Files or Folders respectively. A single file or folder can be retrieved based on its Id column value. All the files belonging to a folder can be retrieved by matching FolderId column value with the Id of the folder. Similarly, all the subfolders of a folder can be retrieved by matching ParentFolderId column value with the Id of the folder.

Creating a Data Access Class

We need to frequently get the data in and out of the database tables and hence we will create a data access class. To code the data access class, create a new web site with C# as the language. The Figure 4 shows the “New Web Site” dialog of Visual Studio.

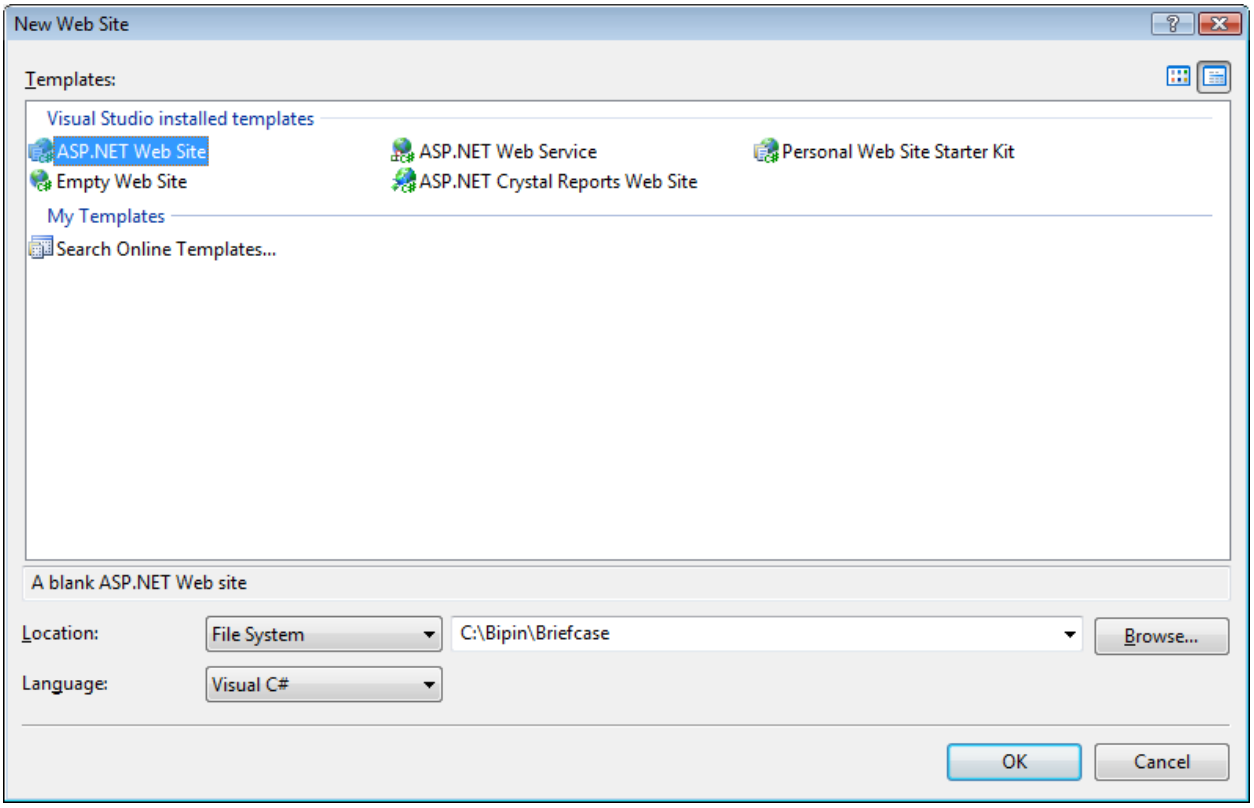

Figure 4: Creating a new Web Site

Then right click on the web site and choose “Add ASP.NET Folder” from the shortcut menu. Further select App_Code to add App_Code folder to your web site. All our classes will reside in this folder. Now add a new class called SqlHelper to the App_Code folder. The SqlHelper will have in all four methods as shown in Figure 6.

Method	Description
ExecuteNonQuery(string)	This method accepts the name of the stored procedure to execute and returns the number of records affected by the query.
ExecuteNonQuery(string,SqlParameter[])	This method accepts the name of the stored procedure to execute and array of SqlParameter objects. It returns the number of records affected by the query.
ExecuteDataSet(string)	This method accepts the name of the stored procedure to execute and returns the results as a DataSet
ExecuteDataSet(string,SqlParameter[])	This method accepts the name of the stored procedure to execute and array of SqlParameter objects. It returns a DataSet filled with the results of the query.

Figure 6: Methods of SqlHelper class

The Figure 7 shows the complete code of the SqlHelper class.

```
namespace Briefcase
{
public static class SqlHelper
{
  private static string strConn;

  static SqlHelper()
  {
   strConn = ConfigurationManager.ConnectionStrings
             ["connectionstring"].ConnectionString;
  }

  public static int ExecuteNonQuery(string sql)
  {
   return ExecuteNonQuery(sql, null);
  }

  public static int ExecuteNonQuery
  (string sql, SqlParameter[] p)
  {
   SqlConnection cnn = new SqlConnection(strConn);
   cnn.Open();
   SqlCommand cmd = new SqlCommand(sql,cnn);
   cmd.CommandType = CommandType.StoredProcedure;
   if (p != null)
   {
    for (int i = 0; i < p.Length; i++)
    {
     cmd.Parameters.Add(p[i]);
    }
  }
  int retval = cmd.ExecuteNonQuery();
  cnn.Close();
  return retval;
 }

  public static DataSet ExecuteDataSet(string sql)
  {
   return ExecuteDataSet(sql, null);
  }

  public static DataSet ExecuteDataSet
  (string sql, SqlParameter[] p)
  {
```

```
    SqlConnection cnn = new SqlConnection(strConn);
    cnn.Open();
    SqlCommand cmd = new SqlCommand(sql, cnn);
    cmd.CommandType = CommandType.StoredProcedure;
    if (p != null)
    {
     for (int i = 0; i < p.Length; i++)
     {
      cmd.Parameters.Add(p[i]);
     }
    }
    SqlDataAdapter da = new SqlDataAdapter();
    da.SelectCommand = cmd;
    DataSet ds = new DataSet();
    da.Fill(ds);
    cnn.Close();
    return ds;
   }
  }
}
```

Figure 7: The SqlHelper class

As you can see from Figure 7, the SqlHelper class picks up the database connection string stored in web.config file and stores it in a static variable. Other static methods such as ExecuteNonQuery() and ExecuteDataSet() use this variable. The Figure 8 shows the <connectionStrings> section of web.config that stores the database connection string.

```
<connectionStrings>
<add name="connectionstring"
connectionString="Data Source=.;initial catalog=briefcasedb;
user id=some_user;password=some_password"
providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Figure 8: Storing database connection string in web.config file.

We will not discuss the code of SqlHelper in more details as it is fairly simple. However, just notice that the CommandType property of the SqlCommand class is set to CommandType.StoredProecedure since all our data access is happening via stored procedures.

Managing Folders

Now that we have created SqlHelper class let's move further and create a class for managing Folders. We need to perform the following operations on the folders:

- Create folders
- Rename folders
- Delete folders
- Retrieve all the subfolders of a folder

To accomplish these tasks we create a class named Folders and add four static methods to it as shown in Figure 9.

Method Name	Description
Create	This method accepts folder name, parent id (if any) and time stamp and adds a new folder entry in the Folders table
Rename	This method accepts Id of the folder to be renamed and its new name and changes the old name to the new one in the Folders table
Delete	This method accepts Id of the folder to be deleted and deletes its entry from the Folders table
DeleteSubFolders	This method accepts the Id of a folder and deletes all its sub folders
GetSubFolders	This method accepts Id of a folder and returns all its subfolders

Figure 9: Methods of Folders class

Each of the method of Folders class is discussed next.

Create a new folder

A new folder is created by calling Create() method of the Folders class. The Figure 10 shows the Create() method.

```
public static int Create
(string FolderName,int ParentFolderId,DateTime DateCreated)
{
 SqlParameter[] p=new SqlParameter[3];
 p[0]=new SqlParameter("@FolderName",FolderName);
 p[1]=new SqlParameter("@ParentFolderId",ParentFolderId);
 p[2]=new SqlParameter("@DateCreated",DateCreated);
 return SqlHelper.ExecuteNonQuery("Folders_Create",p);
}
```

Figure 10: Creating folders

The Create() method accepts folder name, Id of its parent folder and time stamp. If the new folder is supposed to be at top level then parent Id must be passed as 0. It then invokes the ExecuteNonQuery() method of SqlHelper class. Note that the code is calling Folders_Create stored procedure that we created earlier. The Folders table is having a UNIQUE constraint that ensures that folder names are unique under a given scope. The Create() method will throw an exception if the folder name is duplicate in a given context.

Renaming a folder

In order to rename a folder we use Rename() method of Folders class. The Figure 11 shows the code of Rename() method.

```
public static int Rename(int Id, string foldername)
{
 SqlParameter[] p = new SqlParameter[2];
 p[0] = new SqlParameter("@id", Id);
 p[1] = new SqlParameter("@foldername", foldername);
 return SqlHelper.ExecuteNonQuery("Folders_Rename", p);
}
```

Figure 11: Renaming a folder

The Rename() method accepts the Id of a folder that is to be renamed and the new name of the folder. Inside it calls Folders_Rename stored procedure via ExecuteNonQuery() method of the SqlHelper class. Just like Create() method, Rename() method also will throw an exception if there is already a folder with the same name.

Deleting folders

There are two possibilities as far folder deletion is concerned. Firstly we may delete a folder that is not having any subfolder. Secondly, the folder being deleted may contain subfolders. The Delete() and DeleteSubFolders() methods perform the respective operations. The code of both of these methods is shown in Figure 12.

```
public static int Delete(int Id)
{
 SqlParameter[] p=new SqlParameter[1];
 p[0]=new SqlParameter("@Id",Id);
 return SqlHelper.ExecuteNonQuery("Folders_Delete",p);
}

public static int DeleteSubFolders(int ParentFolderId)
{
 SqlParameter[] p = new SqlParameter[1];
 p[0] = new SqlParameter("@Id", ParentFolderId);
 return SqlHelper.ExecuteNonQuery
("Folders_DeleteSubFolders", p);
}
```

Figure 12: Deleting a folder

The Delete() method accepts the folder Id to be deleted. It then calls ExecuteNonQuery() method of SqlHelper and executes Folders_Delete stored procedure.

The DeleteSubFolders() method accepts the Id of a folder whose subfolders are to be deleted. It then executes the Folders_DeleteSubFolders stored procedure by calling ExecuteNonQuery() method of SqlHelper class.

Retrieving a list of subfolders

We need to display subfolders of a folder on the web based user interface that we will build later. Hence, we need a method that will return a list of all the subfolders belonging to a specific folder. The GetSubFolders() method does just that and is shown in Figure 13.

```
public static DataTable GetSubFolders(int Id)
{
 SqlParameter[] p = new SqlParameter[1];
 p[0] = new SqlParameter("@ParentId", Id);
 DataSet ds =
 SqlHelper.ExecuteDataSet("Folders_GetSubFolders", p);
 return ds.Tables[0];
}
```

Figure 13: Retrieving a list of subfolders

The GetSubFolders() method accepts the Id of a folder whose subfolders are to be retrieved and returns a DataTable containing the subfolder information. If the Id parameter is passed as 0 that indicates that all the subfolders at the topmost level are to be retrieved. Internally the GetSubFolders() method executes the Folders_GetSubFolders stored procedure by calling ExecuteDataSet() method of SqlHelper class. The 0^{th} DataTable from the returned DataSet is then returned to the caller.

Managing Files

The way we created a class for managing folders we also create a class called Files for managing files. The File class should allow us to do the following operations:

- Add a new file to a folder
- Delete a single file
- Delete all the files from a specified folder
- Rename a file
- Get all the files belonging to a specific folder

In order to accomplish above tasks we create some static methods to the Files class. These methods are listed in Figure 14.

Method Name	Description
Create	The Create() method creates a new file in the Files table.
Delete	The Delete() method deletes a single file with the specified Id

DeleteFromFolder	The DeleteFromFolder() method deletes all the files from a specific folder
Rename	The Rename() method renames a file
GetFilesFromFolder	The GetFilesFromFolder() method returns a list of all the files from a given folder
GetFile()	The GetFile() method returns information and data about a single file

Figure 14: Methods of Files class

Each of the methods listed in Figure 14 are discussed next.

Creating a file

The Create() method of Files class creates a new file under a specified folder. The Create() method is shown in Figure 15.

```
public static int Create(string FileName,byte[] FileData,int
FileSize,int FolderId,DateTime DateCreated)
{
SqlParameter[] p=new SqlParameter[5];
p[0]=new SqlParameter("@FileName",FileName);
p[1]=new SqlParameter("@FileData",FileData);
p[2]=new SqlParameter("@FileSize",FileSize);
p[3]=new SqlParameter("@FolderId",FolderId);
p[4]=new SqlParameter("@DateCreated",DateCreated);
return SqlHelper.ExecuteNonQuery("Files_Create",p);
}
```

Figure 15: Creating a new file

The Create() method accepts file name, contents of the file, size of the file in bytes, folder Id in which the file is to be created and time stamp. Notice that contents of the file are supplied as byte array. The code then executes the Files_Create stored procedure by calling ExecuteNonQuery() method of SqlHelper class. Recollect that the UNIQUE constraints of Files table ensure that duplicate file names are not stored under a folder.

Renaming a file

The Rename() method renames an existing file and is shown in Figure 16.

```
public static int Rename(int Id,string filename)
{
 SqlParameter[] p = new SqlParameter[2];
 p[0] = new SqlParameter("@id", Id);
 p[1] = new SqlParameter("@filename", filename);
 return SqlHelper.ExecuteNonQuery("Files_Rename", p);
}
```

Figure 16: Renaming a file

The Rename() method accepts the Id of a file that is to be renamed and the new file name. It then executes Files_Rename stored procedure by calling ExecuteNonQuery() method of SqlHelper class.

Deleting files

There are two possibilities as far as file deletion is concerned. We may delete a single file or we may delete the folder. In the former case we need to delete just a single file and Delete() method does that job. In the later case we need to delete all the files from a specified folder. The DeleteFromFolder() method does that task. Both of these methods are displayed in Figure 17.

```
public static int Delete(int Id)
{
 SqlParameter[] p=new SqlParameter[1];
 p[0]=new SqlParameter("@Id",Id);
 return SqlHelper.ExecuteNonQuery("Files_Delete",p);
}

public static int DeleteFromFolder(int folderid)
{
 SqlParameter[] p = new SqlParameter[1];
 p[0] = new SqlParameter("@folderid", folderid);
 return SqlHelper.ExecuteNonQuery("Files_DeleteFromFolder", p);
}
```

Figure 17: Deleting files

The Delete() method accepts the Id of a file that is to be deleted. It then executes Files_Delete stored procedure by calling ExecuteNonQuery() method of SqlHelper class.

The DeleteFromFolder() method accepts the folder Id from which the files are to be deleted. It then executes the Files_DeleteFromFolder stored procedure by calling ExecuteNonQuery() method of SqlHelper class.

Retrieving files

We need to retrieve files for two purposes. Firstly when we wish to download a file we need to get its contents from the Files table. Secondly, when we navigate to a folder we need to display all the files from that folder. The GetFile() and GetFilesFromFolder() methods accomplish these tasks. These methods are listed in Figure 18.

```
public static DataTable GetFile(int Id)
{
 SqlParameter[] p = new SqlParameter[1];
```

```
 p[0] = new SqlParameter("@Id", Id);
 DataSet ds = SqlHelper.ExecuteDataSet("Files_GetFile", p);
 return ds.Tables[0];
}

public static DataTable GetFilesFromFolder(int FolderId)
{
 SqlParameter[] p = new SqlParameter[1];
 p[0] = new SqlParameter("@FolderId", FolderId);
 DataSet ds = SqlHelper.ExecuteDataSet("Files_GetFromFolder",
p);
 return ds.Tables[0];
}
```

Figure 18: Retrieving files

The GetFile() method accepts the Id of a file and returns a DataTable containing details of that file. It retrieves record for that file from Files table by executing Files_GetFile stored procedure.

The GetFilesFromFolder() method accepts the folder Id whose files are to be retrieved. It then executes Files_GetFromFolder stored procedure by calling ExecuteDataSet() method of SqlHelper class. The 0^{th} DataTable from the returned DataSet is returned to the caller.

Summary

Storing files directly in the file system may not be the best way in all the situations. In this article we started developing a web based briefcase application that allows you to create a database driven file system and store files therein. We created database tables and stored procedures that get the data in and out of the tables. We also created three classes namely SqlHelper, Files and Folders. The SqlHelper class is a generic data access layer and encapsulates all the commonly used database operations. The Files and Folders classes internally use SqlHelper and allow you to manage files and folders respectively. In the next article of this two part series we will complete our briefcase by developing a web based user interface.