Code Repository Exercises

Component-based Architecture

Purpose: The purpose of this exercise is to show you can use classes to create a tiered (or component-based) application architecture using a data tier, a business tier, and a presentation tier. You will use a pre-built class called GenericDataAccess to connect to the database and perform all data operations. This class will be called by a business tier class, AuthorAccess, which will then pass the data to the web page. You will see how a component-based architecture can improve maintainability, portability, and scalability by separating application functions.

Directory Name: Components

Instructions:

In this exercise, you will use component-based programming techniques re-create the functionality of the *AuthorManager3.aspx* page you created in the ADO exercise. Do the following:

1. Copy *Authormanager3.aspx* from the ADO directory and paste it in to the directory created for this exercise. Rename it *AuthorManager_Component.aspx*.
2. Download the starting files for this exercise. Place the two .cs files in the App_Code directory of your application. These two files are:
   a. *GenericDataAccess.cs*: This is a static class adapted from the BalloonShop demo application. Its responsibility is to handle all interactions with the database. It has the following methods:
      i. *EcecuteSelectCommand*: executes a select command and returns the results as a DataTable object
      ii. *ExecuteNonQuery*: executes an update, insert, or delete command and returns an integer representing the number of rows affected by the operation
      iii. *ExecuteScalar*: executes a select command that returns a single (scalar) value, as opposed to a set of data (rarely used)
      iv. *CreateCommand*: returns a DbCommand object that has been configured with the appropriate connection string from web.config. The DbCommand object is initialized to invoke a stored procedure.
   b. *AuthorAccess*.cs*:* This is a static class responsible for handling all database transactions dealing with author information. It uses methods of the *GenericDataAccess* class to perform its operations. This class has five methods that correspond to the select/insert/update/delete operations currently performed on the *AuthorManager3.aspx* page.
3. Go back to the *AuthorAccess.cs* file you downloaded. Two of the methods of this class, *GetAllAuthorNames* and *UpdateAuthor*, have been fully implemented. These methods utilize methods from the *GenericDataAccess* class to invoke calls to the stored procedures you created in the ADO.NET exercise. Complete the implementation of the *GetAuthorInfoByID, InsertAuthor*

and *DeleteAuthor* methods (the method declarations are already in place, you just need to fill in the body). Use the fully implemented methods as a guide.

4. Go to the *AuthorManager_Component.aspx* page you copied in step 1. Replace the data access code on the page with calls to the appropriate methods in the *AuthorAccess.cs* class. Specifically, the methods should be called in the following places:
    a. The *FillAuthorList* method should call *AuthorAccess.GetAllAuthorNames*
    b. The *lstAuthor_SelectedIndexChanged* method should call *AuthorAccess.GetAuthorInfoByID*
    c. The *cmdInsert_Click* method should call *AuthorAccess.InsertAuthor*
    d. The *cmdUpdate_Click* method should call *AuthorAccess.UpdateAuthor*
    e. The *cmdDelete_Click* method should call *AuthorAccess.DeleteAuthor*
5. Run your page and test each operation (select/update/insert/delete). It should work exactly like the earlier version of the page created in the ADO exercise.

Header on Default.aspx: Component-based Architecture

Pages linked from Default.aspx: *AuthorManager_Component.aspx*