In this section we are going to present a code sample for each of the methods listed above. With these samples you should get a better understanding of how to use these methods while developing your data access methods.

Be sure to add references to the following assemblies:

```
Microsoft.Practices.EnterpriseLibrary.Configuration.dll
Microsoft.Practices.EnterpriseLibrary.Data.dll
```

You can find these assemblies in the following path:

```
[Drive Name]:\Program Files\Microsoft Enterprise Library\bin
```

In each code file you will need to include the following lines:

```
using Microsoft.Practices.EnterpriseLibrary.Configuration;
using Microsoft.Practices.EnterpriseLibrary.Data;
```

The database used for the following code samples is the Northwind database that ships with Microsoft SQL Server, and the data table used is the Customers table.

Generate a DataReader Object

This code sample shows how to get a DataReader object using the ExecuteDataReader() method.

Below is the stored procedure that returns all records from the Customers table.

Listing 1

```
CREATE PROCEDURE [dbo].[GetCustomerList]
AS
SET NOCOUNT ON
    SELECT
        *
    FROM
        [Customers]
GO
```

The following C# code executes the above stored procedure and obtains the result as a DataReader object:

Listing 2

```
try
{
    // Create DataBase Instance
    Database db = DatabaseFactory.CreateDatabase();
    // Initialize the Stored Procedure
    DBCommandWrapper dbCommandWrapper = GetStoredProcCommandWrapper("GetCustomerList");
    using (IDataReader dataReader = db.ExecuteReader(dbCommandWrapper))
    {
        while (dataReader.Read())
        {
            Response.Write("CustomerID : " +
                dataReader["CustomerID"].ToString()+"<br>");
            Response.Write("CompanyName : " +
            dataReader["CompanyName"].ToString()+"<br><br>");
        }
    }
}
catch (Exception ex)
{
    Response.Write(ex.ToString());
}
```

As you can see, I start by instantiating a new database object as follows:

```
Database db = DatabaseFactory.CreateDatabase();
```

Then, I define a wrapper object which encapsulates the stored procedure's name.

```
DBCommandWrapper dbCommandWrapper = GetStoredProcCommandWrapper("GetCustomerList");
```

Finally, using the generated DataReader object, I was able to display the CustomerID and CompanyName values for each record in the Customers table.

```
using (IDataReader dataReader = db.ExecuteReader(dbCommandWrapper))
    {
       while (dataReader.Read())
       {
```

Generate a DataSet Object

This code sample shows how to get a DataSet object using the ExecuteDataSet() method. The same stored procedure used above in Listing 1 is used in this sample code.

Listing 3

```
try
{
   // Create DataBase Instance
   Database db = DatabaseFactory.CreateDatabase();
   // Initialize the Stored Procedure
   DBCommandWrapper dbCommandWrapper =
   db.GetStoredProcCommandWrapper("GetCustomerList");
   // Generate DataSet object
   DataSet ds = null;
   ds = db.ExecuteDataSet(dbCommandWrapper);
   // Fill DataGrid
   if ( ds.Tables[0].Rows.Count > 0 )
   {
      // Supposing we have a datagrid whose id is DataGrid1
      DataGrid1.DataSource = ds.Tables[0].DefaultView;
      DataGrid1.DataBind();
   }
   else
      Response.Write("No rows were selected");
}
catch (Exception ex)
{
   Response.Write(ex.ToString());
}
```

The same initial steps are used in this code sample as in the previous code sample. However, in this sample I am generating a DataSet object, and then using the following code to check whether the DataSet contains any records:

```
   if ( ds.Tables[0].Rows.Count > 0 )
```

Based on that check, I choose whether to bind the DefaultView of the DataSet to a DataGrid, or to show a message indicating that no records were returned.

Retrieve a Single Row

To retrieve a single row from a database data table, it is recommended that you use a DataReader object. First of all, a SQL Server stored procedure is shown, then followed by the C# code that will execute the stored procedure and return a DataReader object, holding a single row.

Below is the stored procedure that returns a single record from the Customers table.

Listing 4

```
CREATE PROCEDURE [dbo].[GetOneCustomer]
(
   @CustomerID NCHAR(5)
)
```

```
)
AS
SET NOCOUNT ON
    SELECT
        *
    FROM
        [Customers]
    WHERE
        [CustomerID] LIKE @CustomerID
GO
```

The following C# code executes the above stored procedure and obtains a single row in a DataReader object:

Listing 5

```
try
{
    // Create DataBase Instance
    Database db = DatabaseFactory.CreateDatabase();
    // Initialize the Stored Procedure
    DBCommandWrapper dbCommandWrapper = db.GetStoredProcCommandWrapper("GetOneCustomer");
    dbCommandWrapper.AddInParameter("@CustomerID",DbType.String,"ALFKI");
    using (IDataReader dataReader = db.ExecuteReader(dbCommandWrapper))
    {
        if (dataReader.Read())
        {
            Response.Write("CustomerID : " + dataReader["CustomerID"].ToString()+"<br>");
            Response.Write("CompanyName : " + dataReader["CompanyName"].ToString()+"<br><br>");
        }
    }
}
catch (Exception ex)
{
    Response.Write(ex.ToString());
}
```

As you can see in the code above, a new method of the dbCommandWrapper class was used, which is the AddInParameter() method. In the above sample code, this method is used to pass the required @CustomerID parameter to the stored procedure.

The stored procedure in this code sample will query the Customers table and retrieve the row where the CustomerID column matches the CustomerID parameter provided. This explains the need to add an input parameter to the dbCommandWrapper object, as it is a parameter that is required by the stored procedure. In addition to the AddInputParameter() method, there is the AddOutputParameter() method, which I will address in the next section.

Retrieve More than One Column in a Row

The code sample above retrieves an entire record from a database table. Sometimes however we need only specific fields from a row in a database table. In this case we need to supply the stored procedure with not only an input parameter as in the previous section, but also one or more output parameters to store the result of the stored procedure.

The following is an example of a stored procedure which will take as input the CustomerID value and return only two fields: CompanyName and ContactName. For this limited result set, we have to use the AddOutputParameter() method. This method is used to return the output of a stored procedure.

It is recommended when executing a stored procedure with output parameters to use the ExecuteNonQuery() method instead of the ExecuteDataReader() method.

The SQL stored procedure is as follows:

Listing 6

```
CREATE PROCEDURE [dbo].[GetCustomerMultipleFields]
(
    @CustomerID NCHAR(5),
    @CompanyName NVARCHAR(40) OUTPUT,
```

```
    @ContactName NVARCHAR(30) OUTPUT
)
AS
SET NOCOUNT ON
    SELECT
        @CompanyName = [CompanyName],
        @ContactName = [ContactName]
    FROM
        [Customers]
    WHERE
        [CustomerID] LIKE @CustomerID
GO
```

The following C# code executes the above stored procedure and uses the AddOutputParameter() method to obtain the results:

Listing 7

```
try
{
    // Create DataBase Instance
    Database db = DatabaseFactory.CreateDatabase();
    // Initialize the Stored Procedure
    DBCommandWrapper dbCommandWrapper = db.GetStoredProcCommandWrapper("GetCustomerMultipleFields");
    dbCommandWrapper.AddInParameter("@CustomerID", DbType.String, "ALFKI");
    dbCommandWrapper.AddOutParameter("@CompanyName", DbType.String, 40);
    dbCommandWrapper.AddOutParameter("@ContactName", DbType.String, 30);
    //Execute the stored procedure
    db.ExecuteNonQuery(dbCommandWrapper);
    //Display results of the query
    string results = string.Format("Company Name : {0}, Contact Name {1},",
         dbCommandWrapper.GetParameterValue("@CompanyName"),
         dbCommandWrapper.GetParameterValue("@ContactName"));
    Response.Write(results);
}
catch (Exception ex)
{
    Response.Write(ex.ToString());
}
```

In the above code, I passed a customer ID value to the stored procedure as an input parameter. As the stored procedure is designed to return two values, CompanyName and ContactName, these were identified as output parameters.

To display the returned values we used another method of dbCommandWrapper, which is GetParameterValue("ParameterName"). This method returns the value of the parameter specified.

Retrieve a Single Field

This is the case where one needs to retrieve a single column of a row inside a database data table. However, if you are just selecting a single column in a row based on a certain primary key, then no need to use output parameters, instead use the ExecuteScalar() method, which will return the single selected column in the stored procedure. The stored procedure used in this section is as follows:

Listing 8

```
CREATE PROCEDURE [dbo].[GetCustomerSingleColumn]
(
    @CustomerID NCHAR(5)
)
AS
SET NOCOUNT ON
    SELECT
        [CompanyName]
    FROM
        [Customers]
    WHERE
        [CustomerID] LIKE @CustomerID
```

```
GO
```

The C# code used to execute the above stored procedure is as follows:

Listing 9

```
try
{
   // Create DataBase Instance
   Database db = DatabaseFactory.CreateDatabase();
   // Initialize the Stored Procedure
   DBCommandWrapper dbCommandWrapper = db.GetStoredProcCommandWrapper("GetCustomerSingleColumn");
   dbCommandWrapper.AddInParameter("@CustomerID", DbType.String, "ALFKI");
   //Execute the stored procedure
   string GetCompanyName = (string) db.ExecuteScalar(dbCommandWrapper);
   //Display results of the query
   string results = string.Format("Company Name : {0}",GetCompanyName);
   Response.Write(results);
}
catch (Exception ex)
{
   Response.Write(ex.ToString());
}
```

The code is straight forward. I am using ExecuteScalar method to retrieve the returned value, in this case the CompanyName, and then writing the company name to the page.

Insert a New Record

This code sample shows how to insert a new record into the Customers table.

The stored procedure that is used is as follows:

Listing 10

```
CREATE PROCEDURE InsertRecordIntoCustomer
(
   @CustomerID NCHAR(5),
   @CompanyName VARCHAR(50)
)
AS
DECLARE @Result int
IF EXISTS
(
   SELECT
      NULL
         FROM
      [Customers]
         WHERE
      [CustomerID] LIKE @CustomerID
)
   BEGIN
      SELECT @Result = -1
      END
ELSE
   BEGIN
      INSERT INTO [Customers]
         (
         [CustomerID],
         [CompanyName]
         )
         VALUES
         (
         @CustomerID,
         @CompanyName
         )
      SELECT @Result = @@ERROR
      END
RETURN @Result
GO
```

The procedure starts by checking whether the record to insert is already present in the Customers table. If it is found, a value of -1 is returned. If not, the record is inserted and a

value of 0 is returned.

The following C# code will execute the above stored procedure. Executing the above stored procedure includes both input and output parameters. Therefore, it is recommended that the ExecuteNonQuery() method is used.

Listing 11

```
try
{
   // Create DataBase Instance
   Database db = DatabaseFactory.CreateDatabase();
   // Initialize the Stored Procedure
   DBCommandWrapper dbCommandWrapper = db.GetStoredProcCommandWrapper("InsertRecordIntoCustomer");
   // If There are output parameters, use ExecuteNonQuery only, better performance
   dbCommandWrapper.AddInParameter("@CustomerID", DbType.String, "JohnY");
   dbCommandWrapper.AddInParameter("@CompanyName", DbType.String, "Microsoft");
   object Internalvalue = new object();
   dbCommandWrapper.AddParameter("@Result", DbType.Int32, ParameterDirection.ReturnValue,
        "@Result", DataRowVersion.Default,Internalvalue);
   // Get output
   int GetResult = 0;
   // Execute Stored Procedure
   db.ExecuteNonQuery(dbCommandWrapper);
   GetResult = (int)dbCommandWrapper.GetParameterValue("@Result");
   switch ( GetResult )
   {
      case 0:
         Response.Write("Record Inserted.");
         break;
      case -1:
         Response.Write("Record Already Found.");
         break;
      default:
         Response.Write("Record Not Inserted.");
         break;
   }
}
catch (Exception ex)
{
   Response.Write(ex.ToString());
}
```

As you can see, the code is very simple. Two input parameters are added to the dbCommandWrapper, representing part of the record to be inserted in the Customers table. A return parameter is also added, representing the result returned by the stored procedure that indicates whether the record was inserted successfully or not. Once again, notice the use of ExecuteNonQuery() method to perform this task.

Update a Record

This code sample shows how to update an existing record in the Customers table.

The stored procedure that is used is as follows:

Listing 12

```
CREATE PROCEDURE [dbo].[CustomersUpdate]
(
   @CustomerID NCHAR(5),
   @CompanyName NVARCHAR(40)
)
AS
SET NOCOUNT ON
DECLARE @Result INT
IF NOT EXISTS
(
    SELECT
      NULL
         FROM
      [Customers]
```

```
        WHERE
        [CustomerID] LIKE @CustomerID
)
    BEGIN
        SELECT @Result = -1
            END
ELSE
    BEGIN
        UPDATE
            [Customers]
        SET
            [CompanyName] = @CompanyName
        WHERE
            [CustomerID] = @CustomerID

        SELECT @Result = @@ERROR
    END
RETURN @Result
GO
CREATE PROCEDURE [dbo].[GetCustomerList]
AS
SET NOCOUNT ON
    SELECT
        *
    FROM
        [Customers]
GO
```

This stored procedure operates in a manner similar to that shown in the previous code sample. The procedure starts by checking whether the record to be updated already exists. If the record does not exist in the data table a value of -1 is returned. If it exists, the record is updated and a value of 0 is returned.

The following C# code updates a data table record:

Listing 13

```
try
{
    // Create DataBase Instance
    Database db = DatabaseFactory.CreateDatabase();
    // Initialize the Stored Procedure
    DBCommandWrapper dbCommandWrapper = db.GetStoredProcCommandWrapper("CustomersUpdate");
    // If There are output parameters, use ExecuteNonQuery only, better performance
    dbCommandWrapper.AddInParameter("@CustomerID", DbType.String, "Wessam");
    dbCommandWrapper.AddInParameter("@CompanyName", DbType.String, "TerraVision");
    object Internalvalue = new object();
    dbCommandWrapper.AddParameter("@Result", DbType.Int32, ParameterDirection.ReturnValue,
         "@Result", DataRowVersion.Default, Internalvalue);
    // Get output
    int GetResult = 0;
    // Execute Stored Procedure
    db.ExecuteNonQuery(dbCommandWrapper);
    GetResult = (int)dbCommandWrapper.GetParameterValue("@Result");
    switch ( GetResult )
    {
        case 0:
            Response.Write("Record Updated.");
            break;
        case -1:
            Response.Write("Record Not Found.");
            break;
        default:
            Response.Write("Record Not Updated.");
            break;
    }
}
catch (Exception ex)
{
    Response.Write(ex.ToString());
}
```

The above code is straight forward. Input and output parameters were added to the

dbCommandWrapper object. An output parameter is added that represents the result returned by the stored procedure about the status of the update process. Then a call to execute the stored procedure was issued. If the returned value is zero, this means that the record was inserted successfully and the message "Record Updated" is written to the page. On the other hand, if the returned value is -1, this means that the record to be updated was not found in the data table, and the message "Record Not Found" is written to the page. Any other value of the returned variable means that an unknown error has occurred and the following message is written to the page: "Record Not Updated".

Delete a Record

The previous code samples showed how to retrieve, insert, and update data in a database. This final code sample shows how to delete a record from a database table.

Following is the stored procedure used to delete a record from the Customers table.

Listing 14

```
CREATE PROCEDURE [dbo].[CustomersDelete]
(
    @CustomerID NCHAR(5)
)
AS
SET NOCOUNT ON
DECLARE @Result INT
IF NOT EXISTS
(
     SELECT
       NULL
          FROM
       [Customers]
          WHERE
       [CustomerID] LIKE @CustomerID
)
    BEGIN
       SELECT @Result = -1
          END
ELSE
    BEGIN
       DELETE
          [Customers]
       WHERE
          [CustomerID] LIKE @CustomerID

       SELECT @Result = @@ERROR
          END
RETURN @Result
GO
```

Once again the stored procedure starts by checking whether the record to be deleted actually exists in the database table. If not found, a value of -1 is returned, otherwise the record is deleted and a value of 0 is returned.

The following C# code shows how to run the above stored procedure:

Listing 15

```
try
{
    // Create DataBase Instance
    Database db = DatabaseFactory.CreateDatabase();
    // Initialize the Stored Procedure
    DBCommandWrapper dbCommandWrapper = db.GetStoredProcCommandWrapper("CustomersDelete");
    // If There are output parameters, use ExecuteNonQuery only, better performance
    dbCommandWrapper.AddInParameter("@CustomerID", DbType.String, "JohnY");
    object Internalvalue = new object();
    dbCommandWrapper.AddParameter("@Result", DbType.Int32, ParameterDirection.ReturnValue,
         "@Result", DataRowVersion.Default, Internalvalue);
    // Get output
    int GetResult = 0;
```

```
    // Execute Stored Procedure
    db.ExecuteNonQuery(dbCommandWrapper);
    GetResult = (int)dbCommandWrapper.GetParameterValue("@Result");
    switch ( GetResult )
    {
        case 0:
            Response.Write("Record Deleted.");
            break;
        case -1:
            Response.Write("Record Not Found.");
            break;
        default:
            Response.Write("Record Not Deleted.");
            break;
    }
}
catch (Exception ex)
{
    Response.Write(ex.ToString());
}
```

In the above code, an input parameter is added to the dbCommandWrapper class to be able to execute the stored procedure. A return parameter is also added that represents the result returned by the stored procedure about the status of the delete process. As in the previous code sample, the returned value is checked against three different values. If it has a value of zero, this means the record was deleted successfully, and a message “Record Deleted” is written to the page. If it has a value of -1, this means that the record to be deleted was not found in the database, and a message “Record Not Found” is written to the page. Finally, any other value means that an error occurred when the stored procedure was executing, and a message “Record Not Deleted” is written to the page.