9. ADO.NET

ADO.NET is an application programming interface for interaction with data sources using .NET programming code. ADO.NET consists of a proprietary set of Microsoft objects that allow developers to access and efficiently manage data from relational and non-relational databases, including MS Access, Sybase, MS SQL Server, Oracle and even Excel just to name a few. So, if we need to write a program that provides connection to a database, we can use ADO.NET objects in our application to perform database transactions. The System.Data namespace consists of the sub-namespaces and classes that constitute Microsoft's ADO.NET architecture.

ADO.NET is part of an overall Microsoft's data access strategy for universal data access, which attempts to permit connectivity to the vast array of existing and future data sources. (Using the optimized objects in the System.Data.SqlClient namespace we can connect to MS SQL Server databases.) In order for universal data access to work, Microsoft provides for interfaces between .NET programs and third party data sources. This is where OleDb comes in.

Using the objects in the System.Data.OleDb namespace we can connect to OleDb data sources. OleDb objects enable connection to just about any data source.

The primary focus of this chapter is on the ADO.NET classes that enable us to open a connection to a data source, get data from it and put the data into a DataSet, which we examined in a previous chapter. We can then close the connection to the data source. In a nutshell, ADO.NET allows us to connect to and disconnect from a data source, get data from a data source and view and manipulate data. We will then look at an alternative to the DataSet construct and use a DataReader.

9.1 DATABASE CONNECTION

To interact with a database, we first need to establish a persistent connection to it through the use of an OleDbConnection object, which represents a unique connection to a data source. An instance of this class specifies the connection provider and the name and path of the database to which our application will connect.

9.2 DATAADAPTER

An OleDbDataAdapter is a conduit object that allows us to retrieve data from a database by using Structured Query Language, which we will look at in depth in a later chapter. Then, if need be the DataAdapter can also send updated data back to the database to make changes in the data, based on operations performed while the DataSet held the data. In an effort to make multi-tiered applications more efficient, data processing is turning to a message-based approach that revolves around chunks of information.

After we have created an OleDbDataAdapter, we will create a DataSet object in which to place the data the DataAdapter returns to us. Unlike the DataSet example shown in a previous chapter, the we will not have to construct the DataSet's DataTable ourselves. Rather, the DataAdapter will create the DataSet's schema for us.

A DataSet can be thought of as in-memory representation of a relational database, complete with tables, columns, rows, and relationships but is separate and distinct from any data source; does not interact directly with the database and is only a cache of data, with database-like structures within it. DataSets can be used for storing, remoting and programming against flat, XML and relational data.

There are the many important public properties and methods of the DataSet class, including the Tables property that returns a pointer to a table in the collection of DataTables. DataSets are made up of a collection of DataTables and DataRelations. DataTables are in turn made up of collections of DataColumns, DataRows and Constraints. As with a database, the actual data is contained in the Rows collection of DataRow objects

and constraints maintain the data, entity and relational integrity of the data through the ForeignKeyContraints, the UniqueContraints and the PrimaryKey. The DataRelation collection acts as an interface between related rows in different tables.

DataSet Object	
DataTable Collection	DataRelation Collection
Columns (DataColumnCollection)	
DataColumns	
Rows (DataRowCollection)	
DataRows	
Constraints	
Constraint	

In the following code sample, we will see code snippets to build a DataSet with a DataTable. In many situations, however, an OleDbDataAdapter will build these structures automatically. Nonetheless, an understanding of how a DataSet is constructed is absolutely necessary. So, we will build a DataSet from the ground up.

9.3 CODE SAMPLE: DATASET_EXAMPLE

STEP 1: Create a new VB Console Application program named DataSet_Example.
STEP 2: In the main function, create a new DataSet.

```
Dim m_DataSet as DataSet = new DataSet
```

Because DataTables actually hold the data in a DataSet, DataTables are the main topic of discussion. A DataTable holds a Columns collection, which defines the table's schema, a Rows collection, which contains the records in DataRow objects, and Constraints, which ensure the integrity of the data along with the PrimaryKey of the DataTable. We can add a DataTable to a DataSet's collection of Tables using the overloaded Add method.

STEP 3: Now, create a DataTable.

```
Dim m_DataTable as DataTable = new DataTable( "MyStockData" )
```

The DataTable's Columns property returns a pointer to a DataColumnCollection, an object that holds a collection of DataColumn objects and defines the schema of the table. Usually the DataColumnCollection is defined automatically by a DataAdapter's Fill method, and we can then access the DataColumnCollection through the DataTable's Columns property. Because the DataColumnCollection inherits from the CollectionBase class, it uses the Add, Remove, Item and Count methods to insert, delete, get a specified DataColumn from and count the number of DataColumn objects within it. As we will see, in some cases, we may want to define the schema ourselves using the DataTable's Columns properties and methods.

We can add DataColums to the DataColumnCollection using the Columns.Add method.

STEP 4: Create a DataColumn and add it to the DataTable.

```
Dim m_DataColumn as DataColumn = new DataColumn( "ClosingPrices" )
m_DataTable.Columns.Add( m_DataColumn )
```

9.4 ROWS, DATAROWCOLLECTIONS AND DATAROWS

The Rows property of a DataTable returns a reference to a DataRowCollection, a collection that contains the data in DataRow objects. Because the DataRowCollection inherits from the Collection class, it uses the Add, Remove, Item and Count methods to insert, delete, get a

specified DataRow from and count the number of DataColumn objects within it. So, we can add DataRows to the DataTable through the its Rows property using the Rows.Add method.

STEP 5: Create a new DataRow, adding it to the DataTable and define a value for the first Row's item 0.

```
m_DataTable.Rows.Add( m_DataTable.NewRow() )
m_DataTable.Rows( 0 )( 0 ) = 65.24
```

STEP 6: Finally, add the DataTable to the DataSet.

```
m_DataSet.Tables.Add( m_DataTable )
```

We can reference a specific cell to retrieve the data:

```
Console.WriteLine( m_DataSet.Tables( 0 ).Rows( 0 )( 0 ).ToString() )
```

Let's take a look at the entire program.

VB

```
Sub Main()

    ' Create a DataSet, a DataTable and a DataColumn.
    Dim m_DataSet As New DataSet
    Dim m_DataTable As New DataTable("MyStockData")
    Dim m_DataColumn As New DataColumn("ClosingPrices")

    m_DataTable.Columns.Add(m_DataColumn)
    m_DataTable.Rows.Add(m_DataTable.NewRow())
    m_DataTable.Rows(0)(0) = 65.24
    m_DataSet.Tables.Add(m_DataTable)

    Console.WriteLine(m_DataSet.Tables(0).Rows(0)(0).ToString())
End Sub
```

C#

```
using System;
using System.Data;

namespace ConsoleApplication2
{
class Program
{
static void Main(string[] args)
{
   // Create a DataSet, a DataTable and a DataColumn.
   DataSet m_DataSet = new DataSet();
   DataTable m_DataTable = new DataTable("MyStockData");
   DataColumn m_DataColumn = new DataColumn("ClosingPrices");

   m_DataTable.Columns.Add(m_DataColumn);
   m_DataTable.Rows.Add(m_DataTable.NewRow());
   m_DataTable.Rows[ 0 ][ 0 ] = 65.24;
   m_DataSet.Tables.Add(m_DataTable);
```

```
   Console.WriteLine(m_DataSet.Tables[ 0 ].Rows[ 0 ][ 0 ].ToString());
}
}
}
```

9.5 SAMPLE CODE: ADO.NET_EXAMPLE

Let's create a simple .NET program that will illustrate the use of ADO.NET objects for database connectivity.

STEP 1: Create a new Windows Application named ADONET_Example.

STEP 2: On your Form1, place a button, two textboxes and a large datagridview. You can leave the controls with their default names.

STEP 3: To your Form1 code window, add the code to use the namespaces System.Data.OleDb and System.Text.

STEP 4: To Form1, add managed pointers for an OleDbConnection object named m_Conn, an OleDbDataAdapter named m_Adapter, and a DataSet named m_DataSet as shown.

STEP 5: Add the code for the button1_Click event as shown as well as the code for the ColumnAverage function.

Form1.h

```
Imports System
Imports System.Data.OleDb

Public Class Form1

Private m_Conn as OleDbConnection
Private m_Adapter as OleDbDataAdapter
Private m_DataSet as DataSet

Private Sub Button1_Click(…)

   m_Conn = new OleDbConnection( "Provider=Microsoft.Jet.OLEDB.4.0;_
                                  Data Source=C:\Temp\Finance.mdb")
   m_Adapter = new OleDbDataAdapter( "Select * From AXP", m_Conn )
   m_DataSet = new DataSet;

   ‘ Get the data from the database.
   m_Adapter.Fill( m_DataSet, "AXPData" )

   ‘ Close the connection.
   m_Conn.Close()

   ‘ Bind the DataSource to the DataGrid.
   dataGridView1.DataSource = m_DataSet.Tables( 0 )

   ‘ Refer to a specific element in the DataSet.
   textBox1.Text = m_DataSet.Tables( 0 ).Rows( 0 )( 5 )

   ‘ Pass the collection of DataRows to a function.
   Dim double as m_Avg = ColumnAverage( m_DataSet.Tables( 0 ).Rows, 5 )
```

```
   textBox2.Text = m_Avg

End Sub

Private Functions ColumnAverage( ByVal m_DataPoints as DataRowCollection, _
                                 ByVal m_Column as Integer) As Double

   Dim m_Total as Double = 0;
   Dim x as Integer = 0
   For x = 0 To m_DataPoints->Count

      m_Total += m_DataPoints( x ).ItemArray( m_Column )

   Next x
   return m_Total / m_DataPoints.Count

End Function
```

In the button1_Click event above, the first line creates an OleDbConnection object called m_Conn and supplies the connection string. In this case, the Microsoft JET driver is specified as well as the path to the MS Access database known as Finance.mdb. A few lines later, we call the m_Conn.Open() method. At that point, the database connection is made.

The second line of code creates an OleDbDataAdapter object. Two arguments are passed to its constructor: a string containing an SQL statement and the database connection against which the SQL statement will be executed, namely m_Conn. The third line of code creates a DataSet object called m_DataSet.

Once our three objects are created and the connection is open, we can execute the SQL statement by calling the m_Adapter.Fill() method. This method can take two arguments. The first argument is the DataSet that will hold all of the data returned by the SQL query and the second is a String that represents our name for the resulting DataTable. Once the data is in the DataSet, we close the connection to the database.

At this point in the program, all of the data from the table named AXP in the database, now exists in memory in m_DataSet. We display the data by binding dataGridView1 to the specified DataSet and DataTable. (Incidentally, the DataGridView is an excellent grid object.)

As in the DataSet example we looked at earlier in the chapter, we can retrieve any specific element in the DataTable by referencing its DataSet, its DataTable, its row and column. As you will see, the DataAdapter has constructed the DataSet with a schema identical to the AXP table in the Finance database.

Now that the data is in memory we can perform mathematical operations on it. In its current form, the DataSet consists of a data column and open, high low, close and volume columns. In the example, we have created a function that will simple average the elements in a column. So, we can pass a managed pointer to the DataRowCollection.

Notice that the ColumnAverage() function accepts a reference to a DataRowCollection and an integer specifying the column to be averaged. In this case, we are averaging column number 5, the volume column, so our program will print into the label the average volume. The calling statement uses the Rows property, which returns a reference to a DataRowCollection as mentioned earlier.

9.6 Using Excel as a Data Source

Had we wanted to use Excel as a data source, we could easily accomplish this. First off, we would create an Excel spreadsheet named Book1.xls. Then, in Excel we name a range in the

spreadsheet to be used as the table name. Say, "Prices." Then we simply modify the following lines of code:

```
m_Conn = new OleDbConnection( "Provider=Microsoft.Jet.OLEDB.4.0;_
                          Data Source=C:\\Book1.xls;Extended Properties=Excel 9.0")
m_Adapter = new OleDbDataAdapter( "Select * From Prices", m_Conn )
```

9.7 WRITING XML FROM A DATASET

In the button2_Click event function, we can call the m_DataSet.WriteXML() method. In one line of code, the method will convert the entire DataSet into an XML document and in this case save it in the path defined.

STEP 6: Add another button, button2, to your Form1.

STEP 7: Add the following code to the button2_Click event handler as shown.

```
   m_DataSet.WriteXml( "C:\\myXML.xml" )
```

9.8 UPDATING A DATABASE WITH CHANGES IN A DATASET

If we make a change to the DataSet and we want to have the database itself reflect the change, we can use the OleDbDataAdapter's Update method to automatically create and execute the proper SQL INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the DataSet.

STEP 9: Add another button, button3, to your Form1.

STEP 9: Add the following code to the button3_Click event handler as shown.

STEP 10: Run the program and click button1. Once the data appears in the DataGridView, change the data in one of the cells and then click button3.

```
m_Conn.Open()
Dim m_Builder as New OleDbCommandBuilder( m_Adapter )

Dim m_Int as Integer = m_Adapter.Update( m_DataSet.Tables( 0 ) )
m_Conn.Close()
```

STEP 11: Close your application and reopen it. Again, click button1 and you should see that the change you made to the DataSet in STEP 10, which was saved to the database, is now in fact changed in the database.

9.9 RETRIEVING DATA WITH A DATAREADER

As we have chosen to use it in this chapter, a DataSet is essentially an in-memory representation of a database. Not all times, however, are we interested in pulling all the data into memory and storing it. As you can imagine, this has performance implications. If we are interested only in reading data, and not storing it, there is a faster way using a OleDbDataReader. A DataReader allows us to read very quickly a stream of data rows as they arrive from a data source. Once we have read it, we can choose to save the data or drop it. If we don't need to store the data, we are better off using a DataReader performance-wise.

STEP 12: Add another button, button4, to your Form1 and a large textbox, textBox3, with the Multiline property set to true.

STEP 13: Add the following code to the button4_Click event handler as shown.

```
m_Conn.Open()
Dim m_Command as New OleDbCommand( "SELECT * FROM IBM", m_Conn )
```

```
Dim m_Reader as OleDbDataReader = m_Command->ExecuteReader()

While ( m_Reader.Read() )
    textBox3.AppendText( m_Reader.GetDateTime( 0 ) + vbTab +
                         m_Reader.GetDouble( 4 ) + vbCrLf )
End While
m_Reader.Close()
m_Conn.Close()
```

9.10 SUMMARY

In this chapter, we learned how to use the basic functionalities of Microsoft's ADO.NET technology. Specifically, we reviewed OleDbConnections, OleDbDataAdapters, DataSets, DataGridViews and DataRowCollections to view and manipulated data from a database. Also, we looked at using Excel as a data source, updating data sources, and using a DataReader as an alternative to the DataSet model.

In this chapter, we briefly discussed the ADO.NET architecture and some of the OleDb objects for connecting to databases. Specifically, we looked at a model for database interaction that includes the use of OleDbConnection objects, OleDbDataAdapters and DataSets. DataSets contain DataTables, which in turn contain collections of DataColumns and DataRows. Understanding the structure of a DataSet allows us to access the data within the DataSet.

Chapter Problems

1.) What is an OleDbConnection object? What is an OleDbDataAdapter?
2.) Describe the model we use to interact with a database.
3.) Describe the structure of a DataSet object.
4.) What code can we use to access an specific item of data within a DataSet
5.) Write the lines of code necessary to add a DataRow to a DataTable named myDataTable.

Project One

The Finance.mdb database contains several tables. Create a Windows application that gets all the columns from the IBM table and displays it in a DataGrid.

Further, your program should allow the user to enter an index number corresponding to a specific row in the DataTable. In labels, print out the date, open, high, low and closing prices and the volume associated with this index.

Project Two

Create a Windows application that connects to the SPY table in the Finance.mdb database and downloads all the columns into a DataSet. Create a one-dimensional array and populate it with the daily log returns from the dataset. Add the function definitions for the four moments of a distribution: Average(), Variance(), Skew() and Kurtosis(). Print out these values in two labels.

What can we say about the distribution of returns on the SPY over the data set from the values you have calculated.