ADO.NET for Beginners

Accessing Database

Accessing database using ADO.NET in C# or VB.NET This tutorial will teach you Database concepts and ADO.NET in a very simple and easy-to-understand manner with many code snippets and samples. This is primarily meant for beginners and if you are looking for any advanced ADO.NET topics, this may not be the right page for you.

Database Concepts

Database is the media to store data. If you have an application that has to store and retrieve data, your application must be using a database.

A File is the simplest form of saving the data in the disk, but is not the most efficient way of managing application data. A database is basically a collection of one or more files, but in a custom format, and data is organized in a specific format such a way that it can be retrieved and stored very efficiently.

Some examples for databases are:

- MS Access
- SQL Server
- Oracle

MS Access is a very light weight database provided by Microsoft for applications with less number of users and relatively small quantity of data. MS Access saves data into database files with the extension .mdb. Usually, MS Access comes along with MS Office package. If you already have the .mdb database file, you can freely use it with your application and you do not need MS Access software. The MS Access software is required only if you want to directly open the database and manipulate the data or change the database schema.

SQL Server (Microsoft product) and Oracle (Oracle Corp.) are more complex, advanced, relational databases and they are much more expensive. It can support large number of users and very high quantity of data. If you are developing a software, which might be accessed simultaneously by 100s of users or if you expect your data may grow 100s of MBs, you might consider one of these. (We are learning Microsoft .NET, so you might want to consider the SQL Server than Oracle, for which Microsoft provides special data access components!!)

In this tutorial, we will be using only MS Access for simplicity. Most of the samples provided in this site uses MS Access database for simplicity and easy download.
ADO.NET

ADO.NET is the data access model that comes with the .NET Framework. ADO.NET provides the classes required to communicate with any database source (including Oracle, Sybase, Microsoft Access, Xml, and even text files).

DataAccess Providers in .NET

ADO.NET comes with few providers, including:
- OleDb
- SqlClient

There are other providers available, but we are not including them here as this tutorial is meant for beginners! When you want them, search for ADO.NET providers in Google or MSDN.

Microsoft made the SQL Server. So they gave a separate provider, specifically made for SQL Server. We can use the OleDb provider for all other database sources including MS Access, Oracle, Sybase etc. There is a separate provider available for Oracle.

A DATA PROVIDER is a set of classes that can be used to access, retrieve and manipulate data from the databases.

Both OleDb and SqlClient has its own set of classes, but they have the same concepts. We would like to classify the classes into two broad categories (this is not a microsoft classification, anyway!)

- Classes for communicate with database
- Classes for holding/manipulating data

The job of first category of classes is to communicate with database and send or retrieve data from the database. The second category of the classes will be used as a carrier of data.

Classes for communicating with database

The Connection, Command, DataReader, and DataAdapter objects are the core elements of the ADO.NET provider model.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>SqlClient Objects</th>
<th>OleDb Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Establishes a connection to a specific data source.</td>
<td>SqlConnection</td>
<td>OleDbConnection</td>
</tr>
</tbody>
</table>
Each provider may have classes equivalent to above objects. The name of the classes vary slightly to represent the provider type appropriately.

Depending on the type of database you work on, you will have to choose either OleDb or SqlClient (or, some other provider) objects. Since all our samples use MS Access database, we will be using OleDb objects in all the samples. If you like to use SqlServer, you just need to replace the OleDb objects with the equivalent SqlClient objects.

**Classes for holding data**

The following are the main classes used to hold data in Ado.NET:

- **DataSet**
- **DataTable**
- **DataRow**

  - A DataSet is an in-memory representation of the database.
  - A DataTable is a database table.
  - A DataRow represents a record in a database table.

We can use the DataAdapter or DataReader to populate data in DataSet. Once we populate data from database, we can loop through all Tables in the DataSet and through each record in each Table.

On the first look, this may look bit confusing, but once you understand the concept and get familiar with the Ado.NET classes, you will appreciate the power and flexibility of Ado.NET.

Soon, we will publish several ADO.NET samples here. Please check back soon.
Create, Read, Update, Delete - ADO.NET sample

Basic Database operations using ADO.NET - CRUD (Create, Read, Update, Delete)

This chapter demonstrates the basic database operations using the ADO.NET classes. The sample code in this chapter uses the OleDb Provider.

Here is some sample code to execute a simple query.

```csharp
string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\Samples\\Employee.mdb";

OleDbConnection myConnection = new OleDbConnection( connectionString );
myConnection.Open();

string query = "insert into EMPLOYEE_TABLE (EmployeeID, Name, Address) VALUES (101, 'John', '3960 CliffValley Way')";

OleDbCommand myCommand = new OleDbCommand();
myCommand.CommandText = query;
myCommand.Connection = myConnection;
myCommand.ExecuteNonQuery();

myConnection.Close();
```

Let us analyze the code. First we have declared a connection string. The connection string points to an MS Access database. Before you execute this code, make sure you have the database in the path specified. Or, change the path accordingly.

In the next step, we are creating a OleDbConnection object and passing the connection string to this object. The line 'myConnection.Open();' will open a connection to the MS Access database specified in the connection string. If the database doesn't exist or if it is not able to open a connection for some other reason, the 'Open' call will fail.

Next step is, creating a OleDbCommand object. This command object is used to execute sql statements and uses the connection opened by the OleDbConnection object.

Note that before executing a command, we have to establish a valid connection to the database.

And finally, after we have executed with the command, we will close the connection.
The above sample code executes a sql statement and returns no data from database. We are calling the method 'ExecuteNonQuery()' on the command object. If we have a 'select ...' statement which returns data from database, we cannot use the 'ExecuteNonQuery()' method.

The following sample demonstrates using OleDbDataAdapter Object and DataSet to retrieve data from database.

```csharp
string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Samples\Employee.mdb";
OleDbConnection myConnection = new OleDbConnection( connectionString );

string query = "select * from EMPLOYEE_TABLE";

OleDbDataAdapter myAdapter = new OleDbDataAdapter( query, myConnection );
DataSet employeeData = new DataSet();
myAdapter.Fill ( employeeData );
```

Here we are creating a OleDbConnection object and we are just passing the object to the OleDbDataAdapter object. Also, we pass the 'select ...' query to the OleDbDataAdapter. Next, we call the '.Fill()' method of the OleDbDataAdapter. This step will populate the dataset (called 'employeeData') with the data retrieved for the sql statement 'select * from EMPLOYEE'.

As you already know, a DataSet can contain a collection of tables. But in our case, our sql statement will retrieve data from only one table. So, our DataSet will have only one table.

We can iterate through the table in the dataset and retrieve all the records. See the following code demonstrating this:

```csharp
string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Samples\Employee.mdb";
OleDbConnection myConnection = new OleDbConnection( connectionString );

string query = "select * from EMPLOYEE_TABLE";

OleDbDataAdapter myAdapter = new OleDbDataAdapter( query, myConnection );
DataSet employeeData = new DataSet();
myAdapter.Fill ( employeeData );

// Repeat for each table in the DataSet collection.
foreach ( DataTable table in employeeData.Tables )
{
    // Repeat for each row in the table.
    foreach ( DataRow row in table.Rows )
    {
        MessageBox.Show( "Employee Number : " + row["EmployeeNumber"].ToString() );
    }
}
MessageBox.Show("Name : " + row["Name"].ToString());
MessageBox.Show("Address : " + row["Address"].ToString());
}
**DataSet, DataTable, DataRow**

*Manipulating Data using DataSet and DataTable*

DataSet and DataTable are the key components in ADO.NET programming. In simple words, DataSet represents an in memory representation of the database. We can load an entire database into a DataSet and manipulate the data in memory. If you are more familiar with DataSet, you can Add, Edit and Update data in the dataset and then just call a single method 'AcceptChanges()' which will save all the changes back to the database.

- A DataSet contains one or more DataTables
- A DataTable contains DataRows.

What is DataSet?

A DataSet is an in memory representation of data loaded from any data source. Even though the most common data source is database, we can use DataSet to load data from other data sources including XML files etc. In this article, we will talk about the role of DataSet in manipulating data from database.

In .NET, a DataSet is a class provided by the .NET Framework. The DataSet class exposes several properties and methods that can be used to retrieve, manipulate and save data from various data sources.

Just like any other classes in object oriented programming, we have to create an instance of DataSet class to work with data. Typically, we may create a new instance of a DataSet and use other classes provided by .NET Framework to populate the DataSet. See the following example:

```csharp
string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\Samples\\Employee.mdb";
OleDbConnection myConnection = new OleDbConnection( connectionString );
string query = "select * from EMPLOYEE_TABLE";
OleDbDataAdapter myAdapter = new OleDbDataAdapter( query, myConnection );

DataSet employeeData = new DataSet();
myAdapter.Fill ( employeeData );
```

Here we are creating a OleDbConnection object and we are just passing the object to the OleDbDataAdapter object. Also, we pass the 'select ...' query to the OleDbDataAdapter.
Next, we call the '.Fill()' method of the OleDbDataAdapter. This step will populate the dataset (called 'employeeData') with the data retrieved for the sql statement 'select * from EMPLOYEE'.

As you already know, a DataSet can contain a collection of tables. But in the above case, our sql statement will retrieve data from only one table. So, our DataSet will have only one table.

Commonly used properties and methods of DataSet

**Property : Tables**

The Tables property allows us to retrieve the tables contained in the DataSet. This property returns a DataTableCollection object. The following sample code demonstrates iterating through the collection of tables in a data set and print the name of all the tables.

```csharp
DataSet employeeData = new DataSet();
myAdapter.Fill( employeeData );

// Repeat for each table in the DataSet collection.
foreach ( DataTable table in employeeData.Tables )
{  
    MessageBox.Show ( table.TableName );
}
```

Or, you can use the indexer to access any specific table in the collection.

```csharp
DataSet employeeData = new DataSet();
myAdapter.Fill( employeeData );

// Repeat for each table in the DataSet collection.
for ( int i = 0; i < employeeData.Tables.Count; i++ )
{
    DataTable table = employeeData.Tables[i];
    MessageBox.Show ( table.TableName );
}
```

**Method : GetXml()**

The GetXml() method returns the XML representation of the data from the DataSet.

```csharp
DataSet employeeData = new DataSet();
myAdapter.Fill( employeeData );

string xmlData = employeeData.GetXml();
```
Method : WriteXml(...)

The WriteXml() method allows to save XML representation of the data from the DataSet to an XML file. There are many overloaded method available, which takes various parameters. The example shown below takes a file name as parameter and saves the data in DataSet into xml format to the file name specified as parameter. We can optionally save only the data or both data and schema.

```csharp
DataSet employeeData = new DataSet();
myAdapter.Fill( employeeData );
employeeData.WriteXml( "c:\MyData.xml" );
```

Method : ReadXml(...)

The ReadXml() method allows to load the DataSet from an XML representation of the data. There are many overloaded method available, which takes various parameters. The example shown below takes a file name as parameter and loads the data from XML file into the DataSet. This method can be used to load either the data only or both data and schema from the XML.

```csharp
DataSet employeeData = new DataSet();
employeeData.ReadXml( "c:\MyData.xml" );
```

There is another method called 'ReadXmlSchema()', which can be used to load only the schema from a file.

The methods WriteXml() and ReadXml() are useful to save the data from a database into some temporary files, transport to other places or keep it as a local file and load later. Many applications, including the SpiderAlerts tool available for download from this site, uses DataSet to manipulate data and saves/retrieves them from local disk using the WriteXml() and ReadXml() methods.

The SpiderAlerts tool communicates with webservises in our site and retrieves the alerts in the form of a DataSet. Once the Alerts are retrieved, it is saved into local computer using the WriteXml method. (This implementation may be changed soon in the future versions of this tool. We are considering saving(serializing) the DataSet into Isolated Storage (IsolatedStorage is a new feature part of the .NET Framework - it is a kind of hidden file system).

You can read more about Isolated Storage here:
The source code for the SpiderAlerts tool will be available in this site's projects section soon.

More about DataTable and DataRow

**DataTable class in ADO.NET**

A DataTable is a class in .NET Framework and in simple words a DataTable object represents a table from a database.

DataSet and DataTable are the key components in ADO.NET programming. While DataSet can be used to represent a database as a whole, a DataTable object can be used to represent a table in the Database/DataSet. A DataSet can contain several DataTables.

In typical database oriented applications, DataSet and DataTable are used a lot to manipulate data. DataAdapter or other classes can be used to populate a DataSet. Once a DataSet is populated, we can access the DataTables contained within the DataSet.

Just like any database table contains multiple rows (records), a DataTable can contain multiple DataRows. Each row contains multiple fields representing each column in the table.

The typical process to retrieve records from a database in ADO.NET includes the following steps:

- Open a connection to database
- Use a data adapter to fill a DataSet.
- Access the DataTable contained in the DataSet
- Access the DataRows contained in the DataTable.

The following sample code explains these steps. This sample code retrieves data from an MS Access database.

```csharp
string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\Samples\\Employee.mdb";
OleDbConnection myConnection = new OleDbConnection( connectionString );

string query = "select * from EMPLOYEE_TABLE";

OleDbDataAdapter myAdapter = new OleDbDataAdapter( query, myConnection );
DataSet employeeData = new DataSet();
myAdapter.Fill( employeeData );

foreach ( DataTable table in employeeData.Tables )
{
    foreach ( DataRow row in table.Rows )
    {
        MessageBox.Show( "Employee Number : " + 
```
row["EmployeeNumber"].ToString() );
MessageBox.Show("Name : " + row["Name"].ToString() );
MessageBox.Show("Address : " + row["Address"].ToString() );
} 

How to create a DataTable

In most of the cases, we just access the DataTable in a DataSet. We do not need to create a new instance of the DataTable. When a DataSet is populated from database, the DataTable is created with proper schema and data.

If we explicitly create DataTable, we have to create the proper schema. It is bit confusing if you are not very familiar with the database structure and schema.