

ADO.NET uses a multilayer architecture that has components such as the Connection, Reader, Command, Adapter and DataSet objects.

ADO.NET introduced data providers that are a set of special classes to access a specific database, execute SQL commands and retrieve data.

Some examples of data providers include SQL server providers, OLE DB and Oracle providers.

**ADO.NET Namespaces**

|  |  |
| --- | --- |
| Namespaces | Description |
| **System.Data** | Contains the definition for columns, relations, tables, database, rows, views and constraints. |
| **System.Data.SqlClient** | Contains the classes that are used to connect to a MS SQL server database such *as SqlCommand, SqlConnection,*and*SqlDataAdapter*. |
| **System.Data.Odbc** | Contains classes required to connect to most ODBC drivers. These classes include *OdbcCommand* and *OdbcConnection*. |
| **System.Data.OracleClient** | Contains classes such as *OracleConnection*and*OracleCommand* required to connect to an Oracle database. |

**ADO.NET Data Providers**

The [.Net Framework](http://vb.net-informations.com/framework/what_is_net_framework.htm) includes mainly three [Data Providers](http://vb.net-informations.com/ado.net-dataproviders/ado.net-dataproviders-tutorial.htm) for ADO.NET. The Microsoft [SQL Server](http://vb.net-informations.com/ado.net-dataproviders/ado.net-sqlconnection.htm) , [OLEDB](http://vb.net-informations.com/ado.net-dataproviders/ado.net-oledbconnection.htm) and [ODBC](http://vb.net-informations.com/ado.net-dataproviders/ado.net-odbcconnection.htm) are the main Data Providers included in the .Net Framework.

Data provider is used to connect to the database, execute commands and retrieve the record.

The .NET Framework provides the following data providers that we can use in our application.

|  |  |
| --- | --- |
| **.NET Framework data provider** | **Description** |
| .NET Framework Data Provider for SQL Server | It provides data access for Microsoft SQL Server. It requires the **System.Data.SqlClient**namespace. |
| .NET Framework Data Provider for OLE DB | It is used to connect with OLE DB. It requires the **System.Data.OleDb** namespace. |
| .NET Framework Data Provider for ODBC | It is used to connect to data sources by using ODBC. It requires the **System.Data.Odbc** namespace. |
| .NET Framework Data Provider for Oracle | It is used for Oracle data sources. It uses the **System.Data.OracleClient** namespace. |

### **NET Framework Data Providers Objects**

Following are the core object of Data Providers.

|  |  |
| --- | --- |
| **Object** | **Description** |
| Connection | It is used to establish a connection to a specific data source. |
| Command | It is used to execute queries to perform database operations. |
| DataReader | It is used to read data from data source. The DbDataReader is a base class for all DataReader objects. |
| DataAdapter | It populates a DataSet and resolves updates with the data source. The base class for all DataAdapter objects is the DbDataAdapter class. |

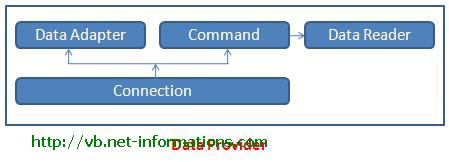
### **.NET Framework Data Provider for SQL Server**

The .NET Framework Data Provider for SQL Server classes is located in the **System.Data.SqlClient** namespace. We can include this namespace in our C# application by using the following syntax.

1. using System.Data.SqlClient;

This namespace contains the following important classes.

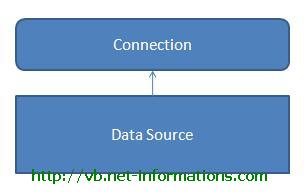
|  |  |
| --- | --- |
| **Class** | **Description** |
| SqlConnection | It is used to create SQL Server connection. This class cannot be inherited. |
| SqlCommand | It is used to execute database queries. This class cannot be inherited. |
| SqlDataAdapter | It represents a set of data commands and a database connection that are used to fill the DataSet. This class cannot be inherited. |
| SqlDataReader | It is used to read rows from a SQL Server database. This class cannot be inherited. |
| SqlException | This class is used to throw SQL exceptions. It throws an exception when an error is occurred. This class cannot be inherited. |



* The four Objects from the .Net Framework provide the functionality of Data Providers in ADO.NET.
* They are Connection Object, Command Object , DataReader Object and DataAdapter Object.
* The Connection Object provides physical connection to the Data Source.
* The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source.
* The DataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data.
* Finally the DataAdapter Object , which populate a Dataset Object with results from a Data Source . The following link shows in details about these Objects.

# ADO.NET Connection Object

* The Connection Object is a part of ADO.NET [Data Provider](http://vb.net-informations.com/ado.net-dataproviders/ado.net-dataproviders-tutorial.htm) and it is a unique session with the Data Source.
* In [.Net Framework](http://vb.net-informations.com/framework/what_is_net_framework.htm) the Connection Object is Handling the part of physical communication between the application and the Data Source.
* Depends on the parameter specified in the Connection String , [ADO.NET](http://vb.net-informations.com/ado.net/ado.net-architecture.htm) Connection Object connect to the specified Database and open a connection between the application and the Database .
* When the connection is established , SQL Commands may be executed, with the help of the Connection Object, to retrieve or manipulate data in the Database.
* Once the Database activity is over , Connection should be closed and release the resources.
* When the connection of an object is instantiated, the constructor takes a connection string that contains the information about the database server, server type, database name, connection type, and database user credentials
* A connection string is usually stored in the web.config file or app.config file of an application.



## **What namespace or provider is used for connection class?**

ADO.NET provides connection to multiple providers. Each provider has a functionality to connect with different database. Here is a list of data providers in ADO.NET and their purpose.

* Data Provider for SQL Server (System.Data.SqlClient).
* Data Provider for MS ACCESS (System.Data.OleDb).
* Data Provider for MYSQL (System.Data.Odbc).
* Data Provider for ORACLE (System.Data.OracleClient).

How to use connection class with this provider is given below-

* Connection object for SQL Server (SqlConnection).
* Connection object for MSACCESS (OleDbConnection).
* Connection object for MYSQL (OdbcConnaction).
* Connection object for ORACLE (OracleConnection).

## **Connection to an ADO.NET Database**

Before working with the database, you must import a data provider namespace, by placing the following in the beginning your code module.  
  
For SqlClient .NET data provider namespace import code:

1. Using System.Data.SqlClient

Similarly, for OLE DB, ODBC, OracleClient .NET data provides namespace import code:

1. Using System.Data.OleDb
2. Using System.Data.Odbc
3. Using System.Data.OracleClient

Now, we have to declare a connection string, which is usually defined in the App.Config or Web Config file, so its availalbe in your application. The typical entry of a connection string is written below:

1. <connectionStrings>
2. <add name="" connectionString="" providerName=""/>
3. </connectionStrings>

Now, if your connection string is pointing to SQL Server database like “EmployeeDataBase”, here os the connection string with   
  
Establish connection string in Web Config file using the below code:

1. <connectionStrings>
2. <add name="Constr" connectionString="Data Source= RaviSERVER\RaviSERVER;Initial Catalog= EmployeeDataBase;User ID=sa,pwd=sa123" providerName="System.Data.SqlClient"/>
3. </connectionStrings>

Now, we create a SqlConnection. We can also pass a connection string direct in the constructor.

1. SqlConnection \_Con = **new** SqlConnection("Data Source= (local); Initial Catalog= EmployeeDataBase; User ID=User Name; pwd=User Password" Integrated Security=”True”);

In the connection string:

* Data Source: This identifies the Server name, which could be the local machine, machine domain name or IP address
* Initial Catalog: This identifies the database name.
* Integrated Security: When you have started database authentication login with Windows authentication, Integrated Security specifies Integrated Security=”True” in connection string, else when you have started the database authentication login with Server authentication Integrated Security specifies Integrated Security=”false” in the connection string
* User Id: Name of the user configured in SQL Server.
* Password: Password matching SQL Server User ID.

If the connection string is stored in the config file with its name Constr”, we can read as following:

1. String \_ConStr = System.Configuration.ConfigurationManager.ConnectionStrings ["Constr"].Connection String;

## **Properties of connection object**

|  |  |
| --- | --- |
| **Property** | **Description** |
| Attributes | We can get or set attributes of the connection object. |
| Command Timeout | By Command time out, we can get or set number of seconds to wait, while attempting to execute a command. |
| Connection Timeout | By Connection time out, we can get or set number of seconds to wait for the connection to open. |
| Connection String | Connection string is used to establish and create connection to data source by using server name, database name, user id and password. |
| Default Database | It gets or returns default database name. |
| Provider | By this property, we can get or set provider name. |
| State | By this property, we can check your current connection open or close before connection opening or closing |

## **Command Object**

The command object is one of the basic components of ADO .NET.

1. The Command Object uses the connection object to execute SQL queries.
2. The queries can be in the Form of Inline text, Stored Procedures or direct Table access.
3. An important feature of Command object is that it can be used to execute queries and Stored Procedures with Parameters.
4. If a select query is issued, the result set it returns is usually stored in either a DataSet or a DataReader object.

**Associated Properties of SqlCommand class**

The properties associated with SqlCommand class are shown in the Table below.

|  |  |  |
| --- | --- | --- |
| Property | Type of Access | Description |
| Connection | Read/Write | The SqlConnection object that is used by the command object to execute SQL queries or Stored Procedure. |
| CommandText | Read/Write | Represents the T-SQL Statement or the name of the Stored Procedure. |
| CommandType | Read/Write | This property indicates how the CommandText property should be interpreted. The possible values are:   1. Text (T-SQL Statement) 2. StoredProcedure (Stored Procedure Name) 3. TableDirect |
| CommandTimeout | Read/Write | This property indicates the time to wait when executing a particular command.    Default Time for Execution of Command is 30 Seconds.    The Command is aborted after it times out and an exception is thrown. |

Now, Let us have a look at various Execute Methods that can be called from a Command Object.

|  |  |
| --- | --- |
| Property | Description |
| ExecuteNonQuery | This method executes the command specifies and returns the number of rows affected. |
| ExecuteReader | The ExecuteReader method executes the command specified and returns an instance of SqlDataReader class. |
| ExecuteScalar | This method executes the command specified and returns the first column of first row of the result set. The remaining rows and column are ignored. |
| ExecuteXMLReader | This method executes the command specified and returns an instance of XmlReader class. This method can be used to return the result set in the form of an XML document |

These are quite often methods in the Command objects. Let us now see each of these with an example

## **ExecuteNonQuery**

1. The ExecuteNonQuery method is used to execute the command and return the number of rows affected.
2. The ExecuteNonQuery method cannot be used to return the result set.

**Snippets working with ExecuteNonQuery**

1. **public** **void** CallExecuteNonQuery()
2. {
3. SqlConnection conn = **new** SqlConnection();
4. conn.ConnectionString = ConfigurationManager.ConnectionStrings["connString"].ConnectionString;
5. **try**
6. {
7. SqlCommand cmd = **new** SqlCommand();
8. cmd.Connection = conn;
9. cmd.CommandText = "DELETE FROM EMP WHERE DEPTNO = 40";
10. cmd.CommandType = CommandType.Text;
11. conn.Open();
12. Int32 RowsAffected = cmd.ExecuteNonQuery();
13. MessageBox.Show(RowsAffected + " rows affected", "Message");
14. cmd.Dispose();
15. conn.Dispose();
16. }
17. **catch** (Exception ex)
18. {
19. MessageBox.Show(ex.Message);
20. }
21. }

Here we have configured a connection string for a Data Source in the App.config file. The content of the App.config file is shown below.

1. <?xml version="1.0" encoding="utf-8" ?>
2. <configuration>
3. <connectionStrings>
4. <add name ="connString" connectionString ="Data Source=ServerName;Initial Catalog=DatabaseName;User Id=User Name; Password=password;" providerName ="System.Data.SqlClient"/>
5. </connectionStrings>
6. </configuration>

To reference this connection we need to add a reference to System.Configuration namespace from:

And add the namespace using System.Configuration;

In the code snippet, we create an instance of a SqlCommand class; it can be a SqlConnection class. We pass the command object the Connection, CommandText, CommandType etc. The connection is then opened, and the command is executed against the connection and the affected rows are returned by the ExecuteNonQuery Method. Then we dispose the command object and the connection object is created.

## **ExecuteReader Method**

1. The DataReader object is a forward-only and read-only cursor.
2. It requires a live connection to the Data Source.
3. The DataReader object cannot be directly instantiated. Instead, we must call the ExecuteReader() Method of the command object to obtain a valid DataReader object.
   1. SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

**Coding**

1. **public** **void** CallExecuteReader()
2. {
3. SqlConnection conn = **new** SqlConnection();
4. conn.ConnectionString = ConfigurationManager.ConnectionStrings["connString"].ConnectionString;
5. **try**
6. {
7. SqlCommand cmd = **new** SqlCommand();
8. cmd.Connection = conn;
9. cmd.CommandText = "SELECT EMPNO,ENAME FROM EMP";
10. cmd.CommandType = CommandType.Text;
11. conn.Open();
12. SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);
13. **if** (reader.HasRows)
14. {
15. **while** (reader.Read())
16. {
17. MessageBox.Show("Employee No: " + reader["EMPNO"].ToString() + " Name :" + reader["ENAME"].ToString());
18. }
19. }
20. cmd.Dispose();
21. conn.Dispose();
22. }
23. **catch** (Exception ex)
24. {
25. MessageBox.Show(ex.Message);
26. }
27. }

The reader.HasRows property returns a boolean value indicating whether rows are returned by the method.

The reader.Read() is used to loop through the result set that is returned by the ExecuteReader method.

## **ExecuteScalar Method**

1. The ExecuteScalar Method in SqlCommandObject returns the first column of the first row after executing the query against the Data Source.
2. If the result set contains more than one column or rows, it takes only the first column of the first row. All other values are ignored.
3. If the result set is empty it will return null.

**Coding**

1. **public** **void** CallExecuteScalar()
2. {
3. SqlConnection conn = **new** SqlConnection();
4. conn.ConnectionString = ConfigurationManager.ConnectionStrings["connString"].ConnectionString;
5. **try**
6. {
7. SqlCommand cmd = **new** SqlCommand();
8. cmd.Connection = conn;
9. cmd.CommandText = "SELECT SUM(SAL) SAL FROM EMP";
10. cmd.CommandType = CommandType.Text;
11. conn.Open();
12. Int32 TotalSalary = Convert.ToInt32(cmd.ExecuteScalar());
13. MessageBox.Show("Total Salary is : " + TotalSalary.ToString());
14. cmd.Dispose();
15. conn.Dispose();
16. }
17. **catch** (Exception ex)
18. {
19. MessageBox.Show(ex.Message);
20. }
21. }

It is best to use ExecuteScalar Method when we use functions like SUM(),COUNT() etc. since it uses fewer resources than the ExecuteReader method.

## **ExecuteXmlReader**

The execute reader method is flexible when we need the result set in the form of an XML document. The ExecuteXmlReader methods returns an Instance of XmlReader class.

**Coding**

1. **public** **void** CallExecuteRow()
2. {
3. SqlConnection conn = **new** SqlConnection();
4. conn.ConnectionString = ConfigurationManager.ConnectionStrings["connString"].ConnectionString;
5. **try**
6. {
7. SqlCommand cmd = **new** SqlCommand("SELECT \* FROM EMP FOR XML RAW('EMPLOYEE'), ROOT('EMP'), ELEMENTS", conn);
8. conn.Open();
9. XmlReader xmlreader = cmd.ExecuteXmlReader();
10. XmlDocument xdoc = **new** XmlDocument();
11. **while** (xmlreader.Read())
12. {
13. xdoc.Load(xmlreader);
14. }
15. xdoc.Save("D:\\Employees.xml");
16. }
17. **catch** (Exception ex)
18. {
19. MessageBox.Show(ex.Message);
20. }
21. }

Here we create an instance of the SqlConnection class and create a SqlCommand connection and pass the SqlCommand class the SQL Statement that returns the XML data. Using the XmlDocument class we load the XmlReader object and save it to the File System using the Save method.

### WHAT IS DATAREADERS?

The **DataReader** object in C# ADO.NET allows you to retrieve data from database in read-only and forward-only mode. It means you can only read and display data but can’t update or delete data. If you want to make modification in retrieved data you need to use DataAdapter instead of DataReader.

### WHY AND WHEN TO USE DATAREADER?

When you want to only display information or search result, you can use DataReader. There are various advantages of using DataReader like:

**1.** The retrieved data is stored in the network buffer in the client and then the client can read data using Read method. As data gets stored in the client network buffer it increases application performance significantly.

**2.** By default **DataReader** stores only one row at a time in memory. It reduces system overhead.

### METHODS AND PROPERTIES OF DATAREADER

#### Properties

|  |  |
| --- | --- |
| **PROPERTY** | **DESCRIPTION** |
| Depth | Indicates the depth of nesting for row |
| FieldCount | Returns number of columns in a row |
| IsClosed | Indicates whether a data reader is closed |
| Item | Gets the value of a column in native format |
| RecordsAffected | Number of row affected after a transaction |

#### Methods

|  |  |
| --- | --- |
| **METHOD** | **DESCRIPTION** |
| Close | Closes a DataRaeder object. |
| Read | Reads next record in the data reader. |