ADO.NET | Active Data Objects .NET | part 1 - basics

It is very important for a developer to have good idea about how application can connect to databases as this becomes the base for working with any remote data sources. ADO.NET of .NET provides many different options/styles for communicating with DB's. **This article focuses on basic knowledge that is must for working with ADO.NET as well as to help learners go to next level of ADO.NET i.e LINQ To SQL, EDM and Cloud parts.**

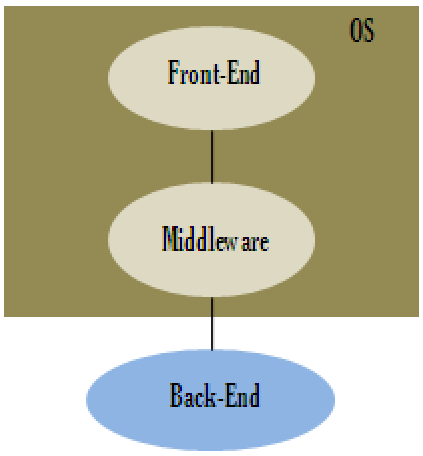
Applications are front-ends. The data in the applications exists only as long as the program is running. They cannot store data because they don't have any storage maintenance. But applications need to store data. So they use exclusive data sources for storing the data.

Data Sources: The data sources can be anything from plain text files to sophisticated RDBMS packages.

i. Flat files, text files, word files, spreadsheets etc.
These data sources have limited size, limited security, no format and they need heavy maintenance.
Ex: Word files, Excel files, Plain text files etc.

ii. Relational Database Management Systems(including DBMS)
These data sources are structured, secured and can contain unlimited data. The data in these sources is fully recoverable.
Ex: Oracle, Sql Server, Sybase etc.

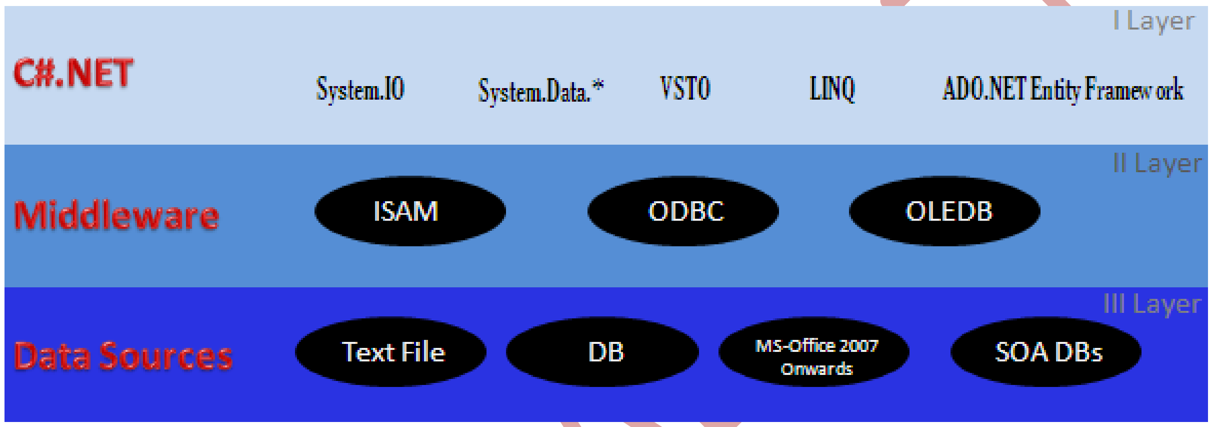
iii. Service-Oriented Databases(SOA Databases)
These are called databases on clouds. These data sources reside on the web and provide storage services to users online. These are gaining prominence in recent times because the users need not bother about the maintenance of the data.
Ex: ASTORIA by Microsoft, WebSphere by IBM etc.



Different data sources observe different protocols for communication. So a normal front-end application cannot interact directly with them. It needs some kind of middleware to interact with each kind of data source. A provider is a set of programs to interact with back-end. The providers designed for .NET are platform independent and hence can be used on any machine.

ADO.NET is a set of libraries which is used for developing database related applications. ADO.NET is a successor of ADO. While ADO is completely COM-based development, ADO.NET is .NET-based. ADO.NET contains the providers which work with different APIs for different data sources.

Ex: ODBC Data Provider for ODBC data sources, OleDb Data Provider for data sources which support OleDb, Oracle Data provider for Oracle database etc.



Common Consumer Objects:

ADO.NET contains common consumer objects for data related access which is independent of database and data providers. These objects work in disconnected environment. In disconnected environment data is always accessed independent of database and no connection and server resources are utilized leading to a good performance. Whereas in connected environment, data is always accessed from database and so it is always dependent on database. Most importantly when user is analysing the data, connection has to remain open which is a load to server and which in turn may result in low performance. As the architecture for accessing data in ADO.NET is purely disconnected, the performance is always high. This disconnected architecture of ADO.NET is supported by the key DataSet and provider specific adapters.

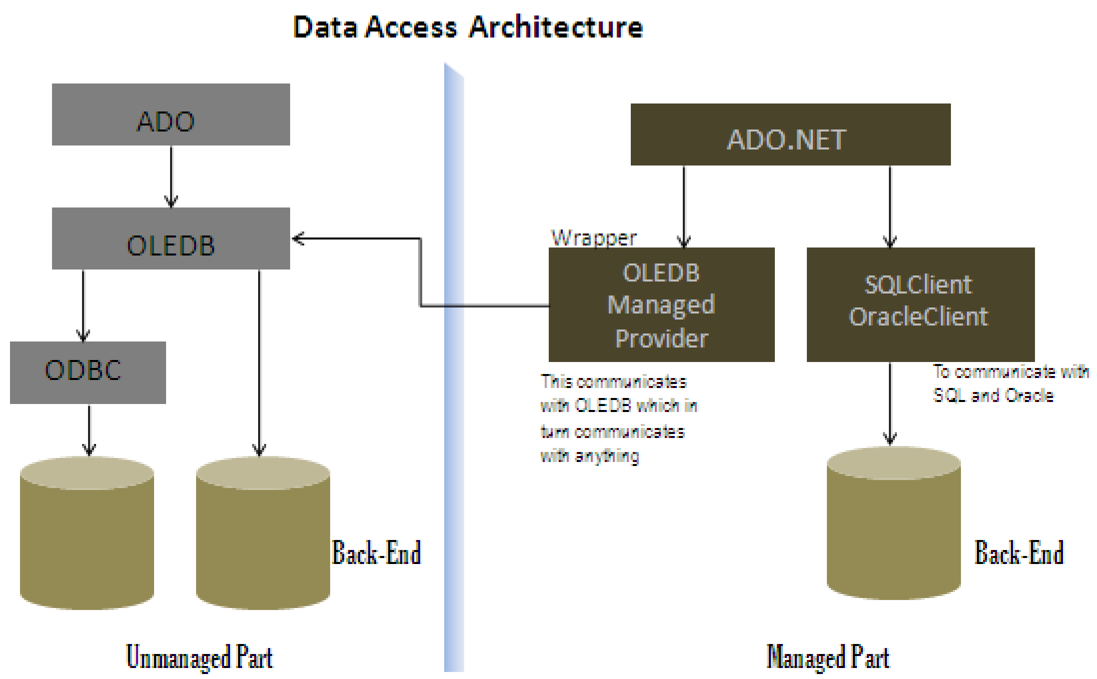
APIs:

ODBC(Open Database Connnectivity): It provides an API which uses SQL Queries to access data. But it has limited features and so not used much when compared to the others.

OLEDB(Object Link and Embedding, Database): OLEDB is an API to access any data stored in a uniform manner. It can be programmed with anything like ODBC, Paint, back-end etc.

Oracle: This works with Oracle database.

Sql: This works with SQL Server.



.NET still depends on OLEDB for communicating with MS-Access, Postgress etc., because it does not have any managed libraries for them.

Differences between ADO and ADO.NET

ADO	**ADO.NET**
1. Designed for connected access(but can also work for disconnected in later versions) 2. Tied to the physical data model 3. The RecordSet is the central data container(RecordSet is the only object in ADO) 4. RecordSet is one table that contains all the data i. Retrieving more than one table or source requires a database join ii. Data is flattened- loses relationships and navigation is sequential 5. Datatypes are bound to COM/COM+ data types 6. Data sharing is via marshalling(process of	1. Designed for disconnected access 2. Can model data logically 3. The DataSet replaces RecordSet(it has two objects DataSet and DataReader) 4. DataSet can contain multiple tables i. Retrieving data from more than one source does not require a join ii. Relationships are preserved and navigation is relational 5. Datatypes are only bound to XML Schema(not to CTS) 6. No datatype conversions required

converting local call to remote call) using DCOM for unilateral behaviour 7. Problems marshalling through firewalls(DCOM, binary) 8. Performance is good and better than ADO.NET	7. XML, like HTML is plain text. So firewall friendly. This is universally accessible and works independent of database. 8. Performance is good but not as much as ADO.

Namespaces:

System.Data:

System.Data is the namespace where the entire ADO.NET base content is present. This namespace contains classes for memory-resident objects into which we retrieve the data from the data source. These common consumer objects are independent of database and providers. That is why all providers interact with each other using common consumer objects (otherwise provider specific classes cannot interact with other provider specific classes).

Some important classes in this namespace are:

i. DataSet
ii. DataTable
iii. DataView
iv. DataRow
v. DataRelation

There are other namespaces containing specific classes for specific data sources. These namespaces include classes that use a common way to interact with different data sources, but the difference is that they use different providers for communication. These classes in each namespace have common names, but are preceded by the API for which they are specific.

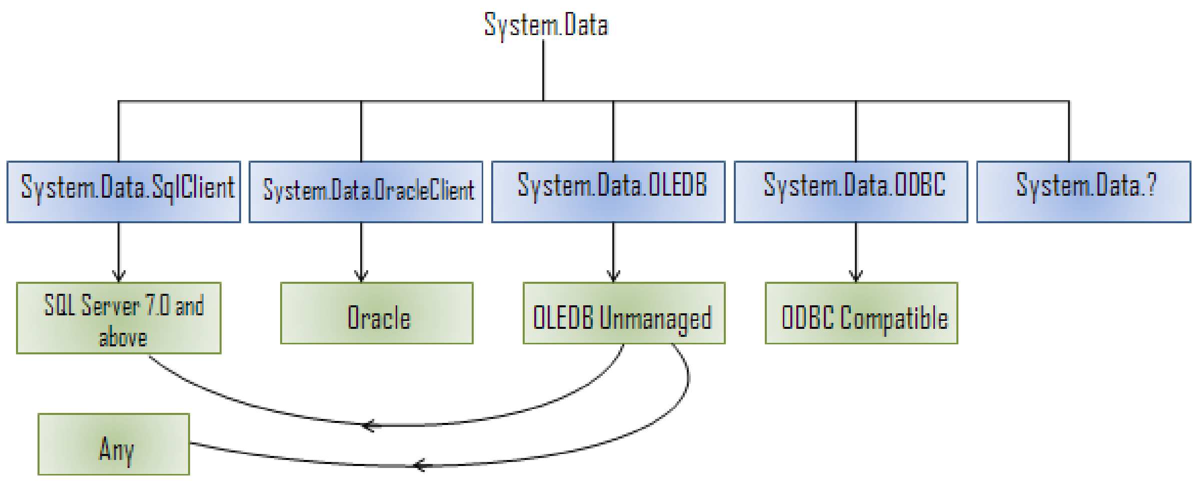
Ex: OdbcConnection class for ODBC connection, OleDbConnection class for OLEDB connection, SqlConnection class for SQL connection etc.

System.Data.Odbc: Contains classes to interact with ODBC data sources. Important classes are OdbcConnection, OdbcCommand, OdbcDataAdapter, OdbcDataReader etc.

System.Data.Oledb: Contains Classes to interact with data sources that support OLEDB. These classes can interact with Oracle, SQL Server, Access and any other data sources. Important classes are OleDbConnection, OleDbCommand, OleDbCommandBuilder etc.

System.Data.SqlClient: Classes to interact with SQL Server. Important classes are SqlConnection, SqlCommand, SqlDataAdapter, SqlCommandBuilder, SqlDbType.

System.Data.OracleClient: Classes to interact with Oracle database. Important classes are OracleConnection, OracleCommand, OracleDataAdapter, OracleCommandBuilder.

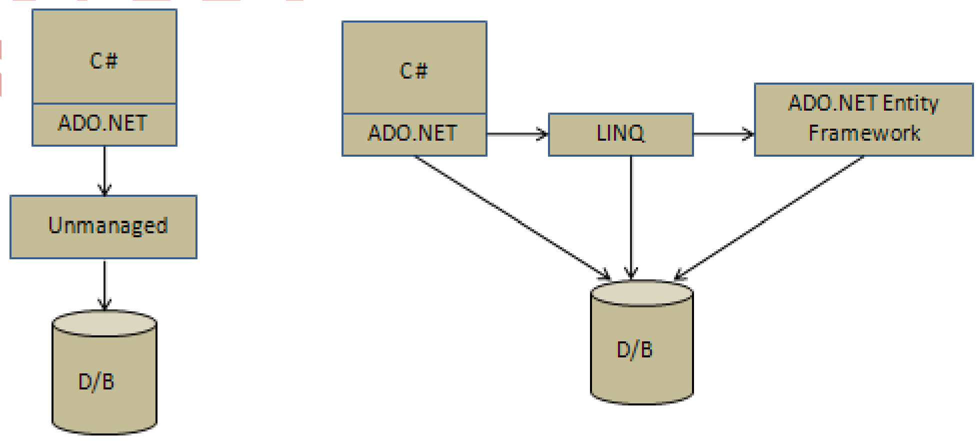


To communicate with a data source, ADO.NET performs the following actions:

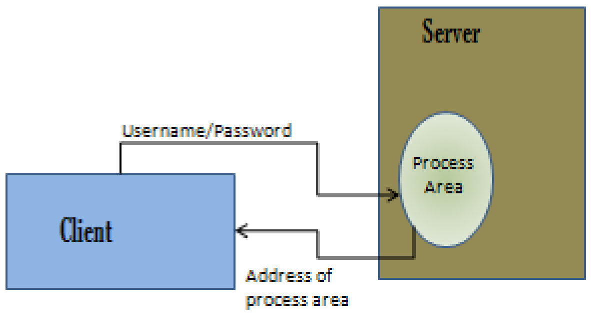
1. Establishes a connection with back-end by passing a connection string. This is done by using the Connection object.
2. Instructs the database as to what actions have to be performed. This is done by using the Command object.
3. Captures the results of database actions. This is done by using the data adapter or other common consumer objects.

Connection Object

This object is responsible for establishing connection between the application and the backend. This can be done with managed as well as unmanaged providers(outdated).



To establish the connection the client has to make a request by supplying the user id and password. If these details are authenticated, a process area is created in server. For each connection request one process area is created. Server gives the address of this process area to client. As long as the client holds this address and as long as the process area exists, connection is established.



All .NET providers provide connection pooling by default. So there no need to explicitly create connection pooling. Because of this connection pooling the disconnected behaviour of .NET objects doesn't affect performance.

Connection Pooling: When connecting to databases it is essential that a connection should be established for as less time as possible, so that other applications waiting for connections can get connected quickly. So a connection should be opened last after performing all the preparation steps required for interacting with the database and it should be closed as soon as the work is completed. This works fine as long as the connection is not needed again. But whenever the connection is needed again, it should be re-established. This becomes costly if it is needed more number of times. To avoid this problem a connection pool is maintained. In a pool the connections are preserved even after they are closed and hence they can be reused for a next request. That is, the process area remains in the pool without getting destroyed and whenever a new connection request is made, simply the address of this process area is given to the requesting client. This means that there is no need to create another process area for a new request which saves time and helps in better performance.

Nearly all databases today support connection pooling. In connection pooling the number of connections preserved depends on the max pool size(the default size is 0). Suppose we specify the max pool size as 20, then 20 connections are created when the first request comes, but only one is allocated to the requesting client. The remaining 19 are preserved for future requests. Even if all the 20 connections are occupied, another connection can be created if a new request comes. But as soon as the work is finished one connection is destroyed i.e. at a time 20 connections are always preserved. There is also a connection timeout property which specifies that the connection should be closed after remaining idle for the given time.

A connection pool is always maintained based on connection string but not on the database or user etc.

Establishing a connection using Connection object:

Connection is established using connection string which is the key for good connection establishment with the back-end. In this connection string we specify the attributes like user id, password, database name, data source name, pooling etc. Depending on these attributes a connection is established between the front-end and the back-end. The important attributes of the connection string are:

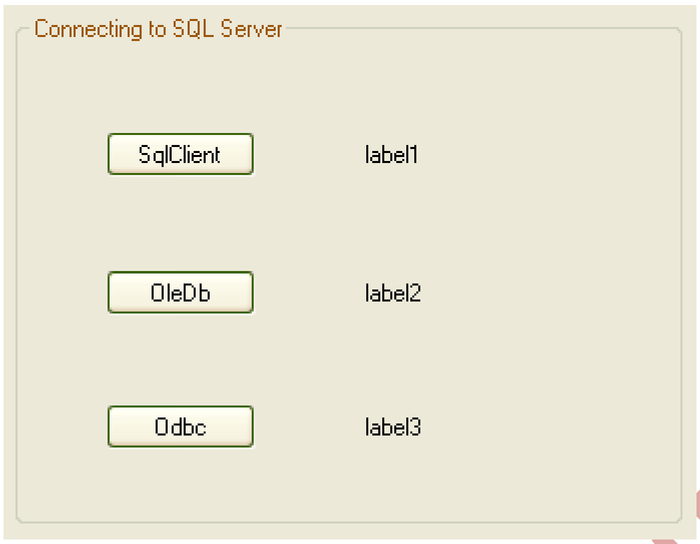
i. **Data Source:** Specifies the data source name
ii. **Server:** The server name where the database resides
iii. **Initial Catalog/Database:** The database name if the data source supports more than one database
iv. **User id/ uid:** The user name for authentication
v. **Password/ pwd:** The corresponding password for authentication
vi. **Pooling = true/false:** Whether the connection should support pooling or not
vii. **Min Connections:** The minimum number of connections that should exist in the pool
viii. **Max Connections:** The maximum number of connections that can exist in the pool
ix. **DSN(Data Source Name):** The data structure name which contains info about the database. This is used in case of ODBC connections.
x. **Multiple Active Recordsets(MARS)= True/False:** Specifies whether more than one recordset can be active at the same time
xi. **Asynchronous Processing = True/False:** Whether the connection should support asynchronous processing or not. Default is false
xii. **Integrated Security=True/False:** Specifies the currently logged in user's credentials for authentication.
xiii. **User Instance=True/ False:** Specifies whether to create new SQL Server instance when connecting. Works only with Windows authentication
xiv. **Persist Security Info=True/False:** Specifies whether any sensitive security information should be retained once the connection is opened.

Other uses of Connection Object:

a) Using connection object we can create some other ADO.NET objects like Command object
b) In ADO.NET transaction management is provided with connection object only. By default all front-end actions are committed actions in database. i.e. whatever actions we perform on the data at front-end are reflected at back-end.

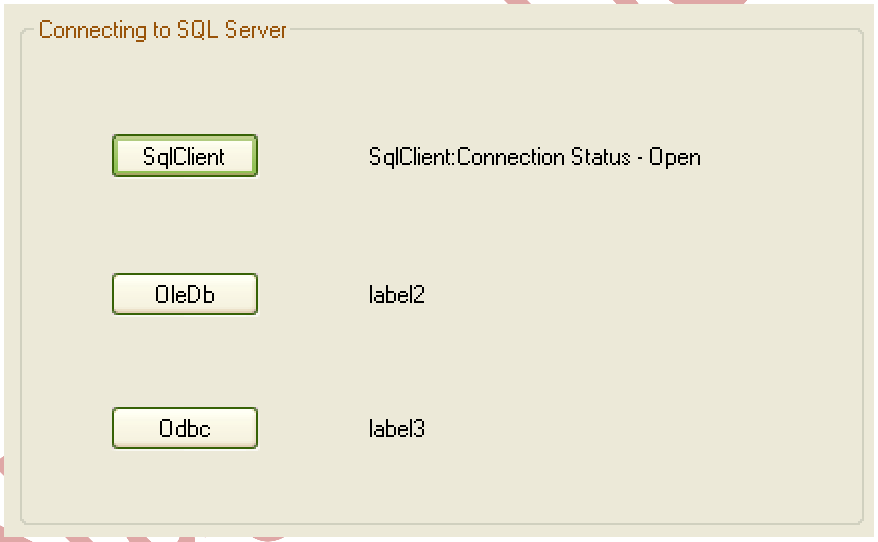
Demo1: To connect to SQL Server using different types of connections like SqlClient, OleDb and Odbc

In this demo we'll just open connections using SqlClient, OleDb and Odbc. For this purpose we design a form with three buttons and three labels, where each button should open one type of connection. When the connection is opened, the status should be displayed by a corresponding label beside the button.



We try to connect to SqlServer in three ways in this same form.

1. Using SqlClient :



When the first button is clicked it should connect to SQL Server using the SqlClient. Here's the code for this button:

This requires the namespace **System.Data.SqlClient**.

```
SqlConnection SqlCon = new SqlConnection("user id=sa; data source=mytoy;
database=pubs");
SqlCon.Open();
label1.Text="SqlClient:Connection Status - " + SqlCon.State.ToString();
```

Here a SQL connection is being created by passing the user id, data source name and the required database. As the password is blank for the given server in this case, there is no need to specify the password. After creating the connection, it should be opened by using the method open().

The connection state can be checked by the connection property called State. Since the connection is successfully opened it returns ‘open’ value which is being displayed by the label.

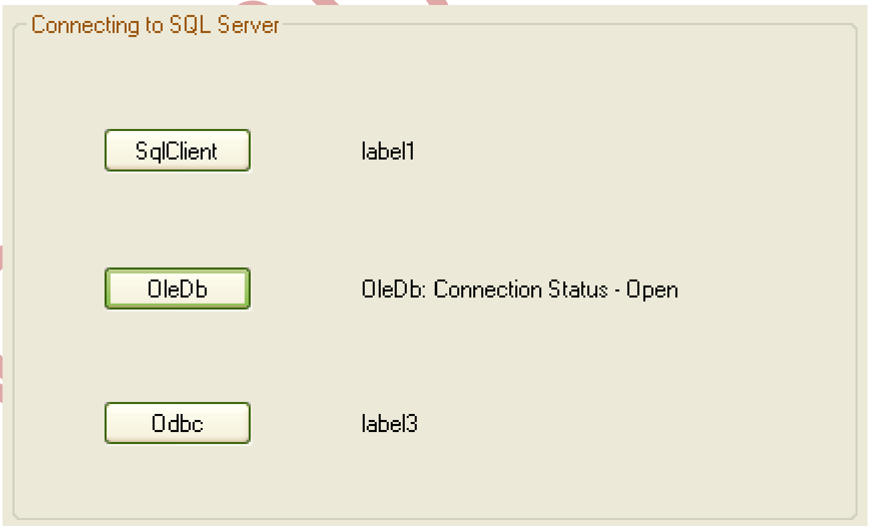
2. Using OleDb:

When the second button is clicked it should connect to SQL Server using OleDb.

This connection requires the namespace **System.Data.OleDb**

The code under the OleDb button click is as follows:

```
OleDbConnection OleDbCon = new OleDbConnection("provider=sqloledb; user id=sa; data
source=mytoy;
                                                database=pubs");
 OleDbCon.Open();
  label2.Text = "OleDb: Connection Status - " + OleDbCon.State.ToString();
```



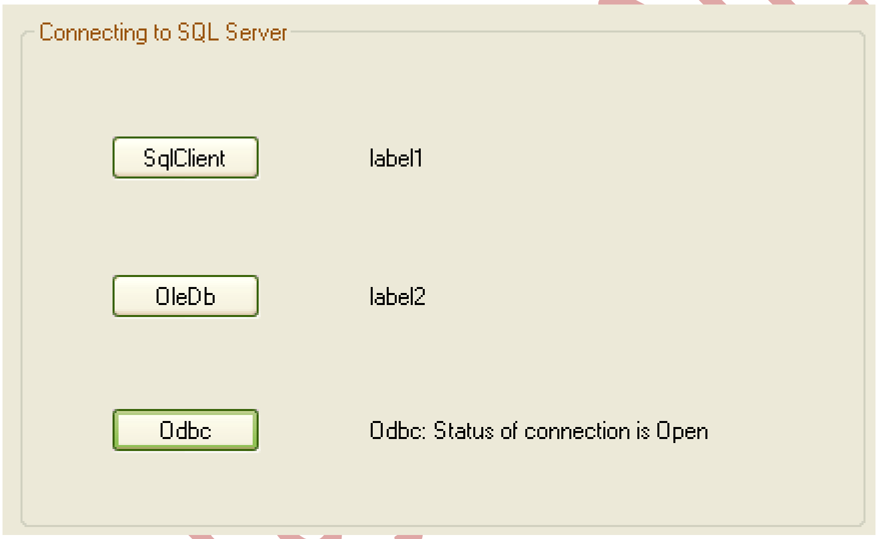
OleDb: Connection Status - Open

Here a OleDb connection is created and like in the previous case the user id, data source and database are specified in the connection string. In this case it is necessary to specify the provider also because OleDb can be used to connect to any kind of data sources. So for each data source it uses a different provider. That is why it is necessary to specify the right provider for the data source being connected. Since the connection here is for SQL Server, the provider is specified as SqlOleDb.

When the connection is successfully opened, the connection state is shown in the label beside the button.

3. Using ODBC:

When the ODBC button is clicked, it should connect to SQL Server using ODBC. For connecting with ODBC, a DSN(Database Source Name) is needed. The ODBC drivers need the information in the DSNs to connect to a data source. For every database, a separate DSN is needed which contains the details like the database name, driver of the database etc. If the user id and password are also included in the DSN details, then there is no need to specify them in the connection string.



Odbc: Status of connection is Open

The procedure to create a DSN is as follows:

In control panel, choose Administrative Tools → Data Sources. In Data Sources select either User DSN or System DSN. If a User DSN is created, it can be used only by the specific user, whereas a System DSN can be used by all the users on that machine. For creating a DSN, select Add and choose the required driver for the data source. In this case, SQL Server is selected as the data source is SQL Server. After that it should be configured properly according to the data source.

While connecting to the SQL Server, the name of the created DSN is specified in the connection string and then the connection is opened. If it is successful the connection status is shown by the label control.

The namespace **System.Data.Odbc** is required for this code.

```
OdbcConnection OdbCon = new OdbcConnection("dsn=SqlConOdbc;data source=mytoy;
                                                    database=pubs");
    OdbcCon.Open();
     label3.Text = "Odbc: Status of connection is " + OdbcCon.State.ToString();
```

In this demo, it is shown how to connect to SQL Server. Similarly, we can connect to any other data source, but only difference is in selection of the namespaces and providers. Nevertheless, the procedure will remain the same.