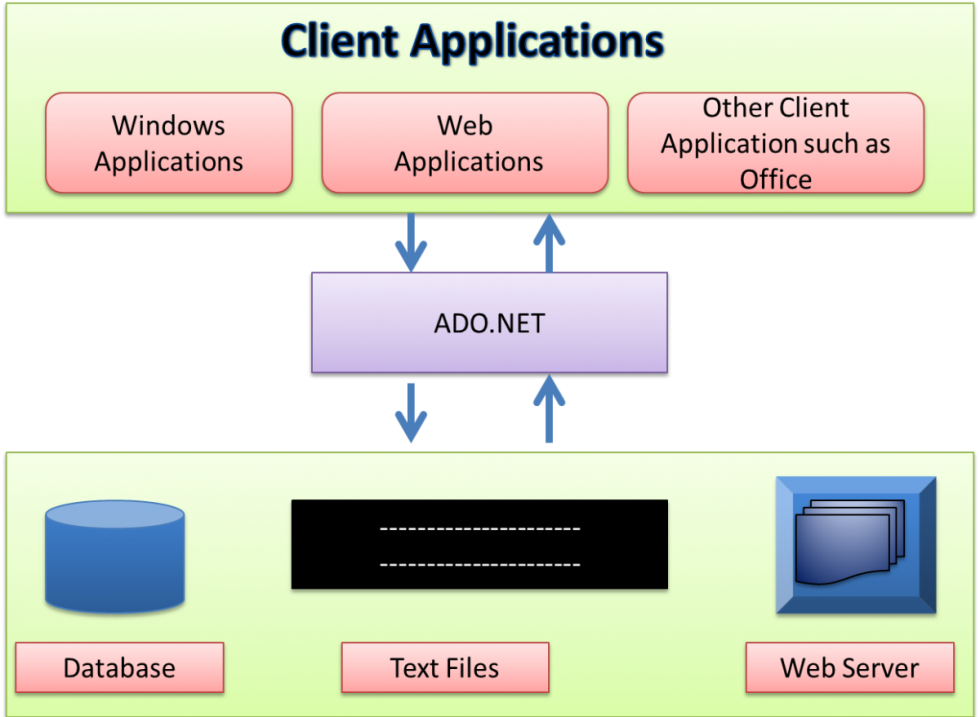
ADO.NET

## ADO.NET is a module of .Net Framework which is used to establish connection between application and data sources. Data sources can be such as SQL Server and XML. ADO.NET consists of classes that can be used to connect, retrieve, insert and delete data.



* ADO is a Microsoft technology
* ADO stands for ActiveX Data Objects
* ADO is a Microsoft Active-X component
* ADO is automatically installed with Microsoft IIS
* ADO is a programming interface to access data in a database

# Connected Vs Disconnected

## A) Connected mode is faster in performance because it works in forward only-read only manner where as disconnected get low in

## speed and performance because it can insert, update, delete and select data.

## B) Connected mode can hold the data of single table where as disconnected can hold multiple tables of data.

## C) Connected mode is forward only-read only objects where as Disconnected can process data in any dimension.

# ADO.NET Components

## The two main components of ADO.NET for accessing and manipulating data are the

## 1. .Net Framework Data Providers

## 2. Dataset

## .

# .NET Framework Data Providers

## The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.

## The Connection object provides connectivity to a data source.

## The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.

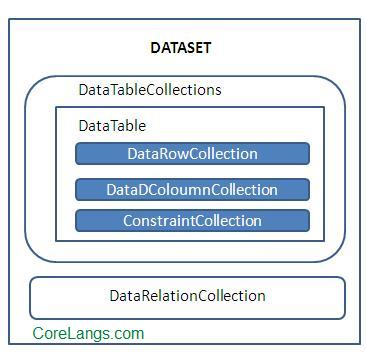
## The DataReader provides a high-performance stream of data from the data source. Finally, the DataAdapter provides the bridge between the DataSet object and the data source.

## The DataAdapter uses Command objects to execute SQL commands at the data source to both load the DataSet with data and reconcile changes that were made to the data in the DataSet back to the data source.

## **DataSet**

## The DataSet contains a collection of one or more DataTable objects consisting of rows and columns of data, and also primary key, foreign key, constraint, and relation information about the data in the DataTable objects. For more information, see DataSets, DataTables, and DataViews.

## It is explicitly designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application.



Dataset

## Figure represents the ADO.NET components model and how they work together: fig3.2.gif

## **ADO.NET Architecture**

## asp.net-ado.net-architecture

## ADO.NET consist of a set of Objects that expose data access services to the .NET environment. It is a data access technology from Microsoft .Net Framework , which provides communication between relational and non relational systems through a common set of components .

## System.Data namespace is the core of ADO.NET and it contains classes used by all data providers. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate ADO.NET data access code.

## **Data Providers and DataSet**

## The two key components of ADO.NET are Data Providers and DataSet . The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases. DataSet class provides mechanisms for managing data when it is disconnected from the data source.

## **Data Providers**

## The .Net Framework includes mainly three Data Providers for ADO.NET. They are the Microsoft SQL Server Data Provider , OLEDB Data Provider and ODBC Data Provider . SQL Server uses the SqlConnection object , OLEDB uses the OleDbConnection Object and ODBC uses OdbcConnection Object respectively.

|  |  |  |  |
| --- | --- | --- | --- |
| Provider Name | API prefix | Namespace | Data Source Description |
| ODBC Data Provider | Odbc | System.data.Odbc | Data Sources with an ODBC interface. Normally older databases. |
| OleDb Data Provider | OleDb | System.data.oledb | Data Sources that expose an OleDb interface, i.e. Access or Excel. |
| Oracle Data Provider | Oracle | system.data.oracleclient | For Oracle Databases. |
| SQL Data Provider | SQL | System.data.SqlClient | For interacting with Microsoft SQL Server. |

## A data provider contains Connection, Command, DataAdapter, and DataReader objects. These four objects provides the functionality of Data Providers in the ADO.NET.

## **Connection**

## The Connection Object provides physical connection to the Data Source. Connection object needs the necessary information to recognize the data source and to log on to it properly, this information is provided through a connection string.

## **Command**

## The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source. The command object provides a number of Execute methods that can be used to perform the SQL queries in a variety of fashions.

## **DataReader**

## 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. DataReader requires a live connection with the databse and provides a very intelligent way of consuming all or part of the result set.

## **DataAdapter**

## DataAdapter Object populate a Dataset Object with results from a Data Source . It is a special class whose purpose is to bridge the gap between the disconnected Dataset objects and the physical data source.

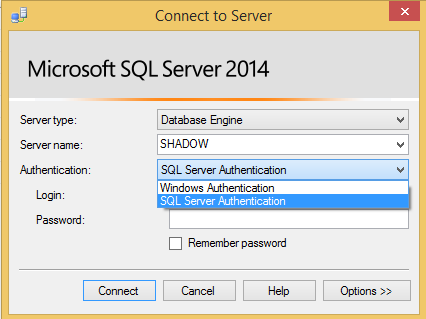
## **DataSet**

## DataSet provides a disconnected representation of result sets from the Data Source, and it is completely independent from the Data Source. DataSet provides much greater flexibility when dealing with related Result Sets.

## DataSet contains rows, columns,primary keys, constraints, and relations with other DataTable objects. It consists of a collection of DataTable objects that you can relate to each other with DataRelation objects. The DataAdapter Object provides a bridge between the DataSet and the Data Source.

C# ADO.NET Connection String

Connection String is a normal String representation which contains Database connection information to establish the connection between Database and the Application. The Connection String includes parameters such as the name of the driver, Server name and Database name , as well as security information such as user name and password.



In the above diagram it shows data source(server name) SHADOW and types of authentication windows and sql server authentication.

Generally connection string authentication can be classified into two types.

1. Sql server authentication
2. Windows application

1.Sql server authentication.(Standard Connection with username and password)

Authentication is used to connect sql server database by using the sql server credential along with data source and initialcatalog.

Syntax:

connetionString="DataSource=ServerName;InitialCatalog=DatabaseName;UserID=UserName;Password=Password"

EX.

connetionString="Data Source=Shadow;InitialCatalog=Test;UserID=sa;Password=sa123"

Data source:

Data source name i.esql server name where data resides in the schema.

Initial catalog(Schema):

In data source(server) we have number of databases(Schema). Test is the schema name. server contains number of schema(database).

User Id:

This id is given to the sql server login name as part of credential i.e (sa);

Password :

This password is given to sql server login password as part of credential i.e (sa123)

Note: credential = User Id + Password;

2. Windows authentication

Authentication is used to connect sql server database by using the windows credential along with data source and initialcatalog.

Syntax :

connetionString="Data Source=ServerName;InitialCatalog=DatabaseName; Integrated Security=SSPI;"

Ex.

connetionString="Data Source=Shadow;InitialCatalog=Test;Integrated Security=True;"

Here everything is same as sql server authentication except Integrated security. It is the windows authentication used to give when computer is switched on.

Working With connection Object:

The Connection Object is Handling the part of physical communication between the C# application and the SQL ServerDatabase . An instance of the Connection class in C# is supported the Data Provider for SQL Server Database. The Connection instance takes Connection String as argument and pass the value to the Constructor statement.

Creating connection to the MSSQL Server.

It is used to establish an open connection to the SQL Server database. It is a sealed class so that cannot be inherited. **SqlConnection** class uses **SqlDataAdapter** and **SqlCommand** classes together to increase performance when connecting to a Microsoft SQL Server database.

Connection does not close explicitly even it goes out of scope. Therefore, you must explicitly close the connection by calling Close() method.

## **SqlConnection Signature :**

public sealed class SqlConnection : System.Data.Common.DbConnection, ICloneable, IDisposable

SqlConnection Constructors

|  |  |
| --- | --- |
| Constructors | Description |
| SqlConnection() | It is used to initializes a new instance of the SqlConnection class. |
| SqlConnection(String) | It is used to initialize a new instance of the SqlConnection class and takes connection string as an argument. |
| SqlConnection(String,SqlCredential) | It is used to initialize a new instance of the SqlConnection class that takes two parameters. First is connection string and second is sql credentials.that takes two parameters. First is connection string and second is sql credentials. |

### **SqlConnection Methods**

|  |  |
| --- | --- |
| Method | Description |
| BeginTransaction() | It is used to start a database transaction. |
| ChangeDatabase(String) | It is used to change the current database for an open SqlConnection. |
| ChangePassword(String, String) | It changes the SQL Server password for the user indicated in the connection string. |
| Close() | It is used to close the connection to the database. |
| CreateCommand() | It enlists in the specified transaction as a distributed transaction. |
| GetSchema() | It returns schema information for the data source of this SqlConnection. |
| Open() | It is used to open a database connection. |
| ResetStatistics() | It resets all values if statistics gathering is enabled. |

## **SqlConnection Example**

Now, let's create an example that establishes a connection to the SQL Server. We have created a **Student** database and will use it to connect. Look at the following C# code.

using (SqlConnection connection = new SqlConnection(connectionString))

{

  connection.Open();

}

**Using** block is used to close the connection automatically. We don't need to call close () method explicitly, **using** block do this for ours implicitly when the code exits the block.

**// Program.cs**

using System;

using System.Data.SqlClient;

namespace AdoNetConsoleApplication

{

    class Program

    {

        static void Main(string[] args)

        {

            new Program().Connecting();

        }

        public void Connecting()

       {

            using (

                     // Creating Connection

                     SqlConnection con = new SqlConnection("data source=.; database=student; inegrated security=True")

                 )

            {

                con.Open();

                Console.WriteLine("Connection Established Successfully");

            }

        }

    }

}

# ADO.NET SqlCommand Class

This class is used to store and execute SQL statement for SQL Server database. It is a sealed class so that cannot be inherited.

## **SqlCommand Signature**

public sealed class SqlCommand : System.Data.Common.DbCommand, ICloneable, IDisposable

### **Constructors**

This class provides the following constructors.

|  |  |
| --- | --- |
| Constructor | Description |
| SqlCommand() | It is used to initialize a new instance of the SqlCommand class. |
| SqlCommand(String) | It is used to initialize a new instance of the SqlCommand class with a string parameter. |
| SqlCommand(String, SqlConnection) | It is used to initialize a new instance of the SqlCommand class. It takes two parameters, first is query string and second is connection string. |
| SqlCommand(String, SqlConnection, SqlTransaction) | It is used to initialize a new instance of the SqlCommand class. It takes three parameters query, connection and transaction string respectively. |
| SqlCommand(String, SqlConnection, SqlTransaction, SqlCommandColumnEncryptionSetting) | It Initializes a new instance of the SqlCommand class with specified command text, connection, transaction, and encryption setting. |

### **Methods**

|  |  |
| --- | --- |
| Method | Description |
| BeginExecuteNonQuery() | It is used to Initiate the asynchronous execution of the SQL statement described by this SqlCommand. |
| Cancel() | It tries to cancel the execution of a SqlCommand. |
| Clone() | It creates a new SqlCommand object that is a copy of the current instance. |
| CreateParameter() | It creates a new instance of a SqlParameter object. |
| ExecuteReader() | It is used to send the CommandText to the Connection and builds a SqlDataReader. |
| ExecuteXmlReader() | It is used to send the CommandText to the Connection and builds an XmlReader object. |
| ExecuteScalar() | It executes the query and returns the first column of the first row in the result set. Additional columns or rows are ignored. |
| Prepare() | It is used to create a prepared version of the command by using the instance of SQL Server. |
| ResetCommandTimeout() | It is used to reset the CommandTimeout property to its default value. |

## **Example**

In this example, we are creating a **SqlCommand** instance and executing a SQL statement.

**// Program.cs**

using System;

using System.Data.SqlClient;

namespace AdoNetConsoleApplication

{

    class Program

    {

        static void Main(string[] args)

        {

            new Program().CreateTable();

        }

        public void CreateTable()

        {

            SqlConnection con = null;

            try

            {

                // Creating Connection

                con = new SqlConnection("data source=.; database=student; integrated security=SSPI");

              // writing sql query

            SqlCommand cm = new SqlCommand("select \* from student", con);

                 // Opening Connection

                con.Open();

                 // Executing the SQL query

                 SqlDataReader sdr = cm.ExecuteReader();

                 while (sdr.Read())

                 {

                     Console.WriteLine(sdr["name"]+" "+ sdr["email"]);

                 }

             }

             catch (Exception e)

             {

                 Console.WriteLine("OOPs, something went wrong." + e);

             }

             // Closing the connection

             finally

             {

                 con.Close();

             }

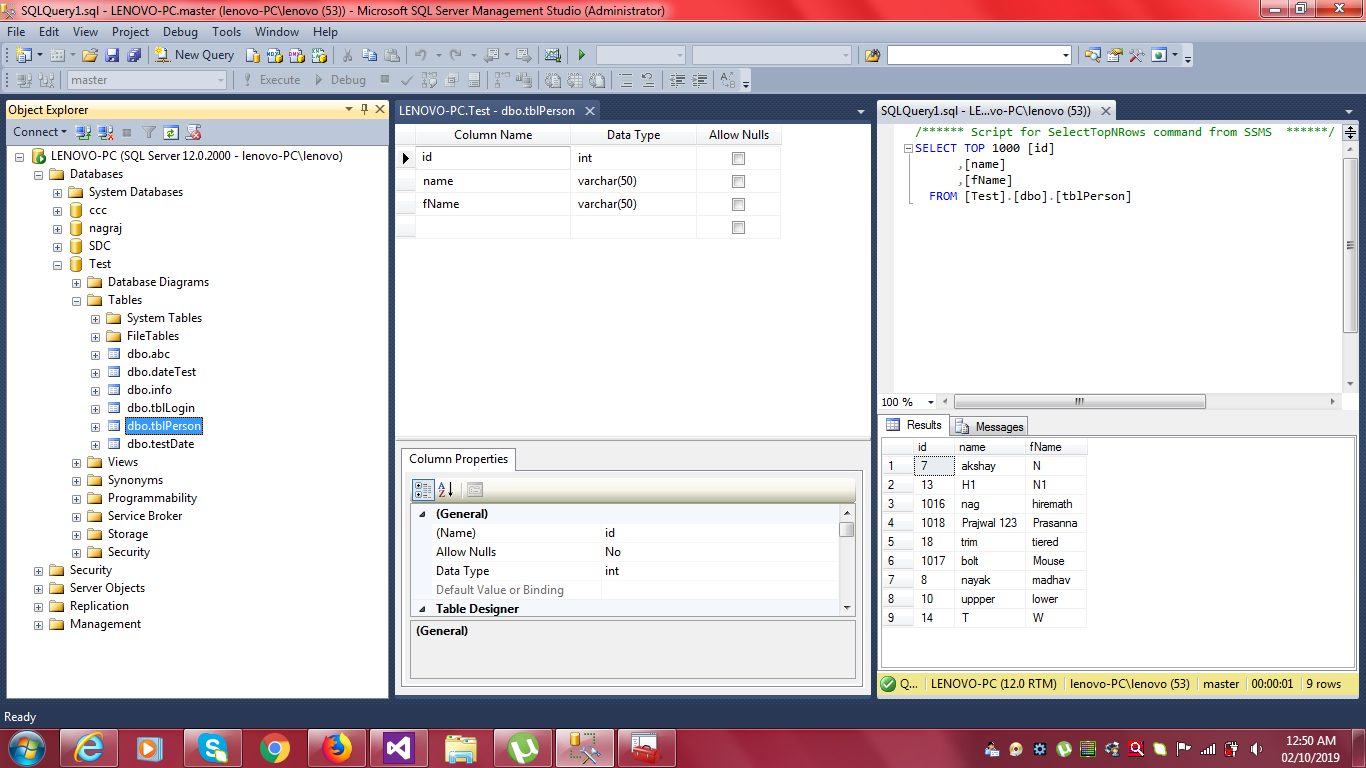
         }

     }

}

DATABASE:

Here we will assume that database is exists for inserting, updating and deleting the records in MSSql 2014.



Inserting Record:

**// Program.cs**

using System;

using System.Data.SqlClient;

namespace AdoNetConsoleApplication

{

    class Program

    {

        static void Main(string[] args)

        {

// Creating connection string

string conString = "Data Source=LENOVO-PC;Initial Catalog=test;Integrated Security=true";

SqlConnection con = new SqlConnection(conString);

// Creating command

string cmdString="insert into tblPerson values (‘Nagabhushan’,’Hiremath’)";

SqlCommand cmd = new SqlCommand(cmdString, con);

// Open database connection

con.Open();

// Execute command

int intRowsEfct = cmd.ExecuteNonQuery();

// Close database connection

con.Close();

}

    }

}

 Updating record:

**// Program.cs**

using System;

using System.Data.SqlClient;

namespace AdoNetConsoleApplication

{

    class Program

    {

        static void Main(string[] args)

        {

// Creating connection string

string conString = "Data Source=LENOVO-PC;Initial Catalog=test;Integrated Security=true";

// SqlConnection object

SqlConnection con = new SqlConnection(conString);

// Creating command string

string cmdString = "update tblPerson set name='Akash', fname='S Hiremath' where id=2;

// SqlCommand object

SqlCommand cmd = new SqlCommand(cmdString, con);

// Open database connection

con.Open();

// Execute command

int i = cmd.ExecuteNonQuery();

// close the database connection

con.Close();

}

    }

}

Deleting a Record:

**// Program.cs**

using System;

using System.Data.SqlClient;

namespace AdoNetConsoleApplication

{

    class Program

    {

        static void Main(string[] args)

        {

// Creating connection string

string conString = "Data Source=LENOVO-PC;Initial Catalog=test;Integrated Security=true";

SqlConnection con = new SqlConnection(conString);

// Creating command

string cmdString = "delete tblperson where id=2;

SqlCommand cmd = new SqlCommand(cmdString, con);

// Open database connection

con.Open();

// Execute command

int i = cmd.ExecuteNonQuery();

// close the database connection

con.Close();

}

    }

}

Creating Dataset From DataAdapter:

The [SqlDataAdapter](https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqldataadapter?view=netframework-4.8), serves as a bridge between a [DataSet](https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=netframework-4.8) and SQL Server for retrieving and saving data. The [SqlDataAdapter](https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqldataadapter?view=netframework-4.8) provides this bridge by mapping [Fill](https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbdataadapter.fill?view=netframework-4.8), which changes the data in the [DataSet](https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=netframework-4.8) to match the data in the data source, and [Update](https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbdataadapter.update?view=netframework-4.8), which changes the data in the data source to match the data in the [DataSet](https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=netframework-4.8), using the appropriate Transact-SQL statements against the data source.

**// Program.cs**

using System;

using System.Data;

using System.Data.SqlClient;

namespace AdoNetConsoleApplication

{

    class Program

    {

        static void Main(string[] args)

        {

// Creating connection string

string conString = "Data Source=LENOVO-PC;Initial Catalog=test;Integrated Security=true";

SqlConnection con = new SqlConnection(conString);

// Creating command

string cmdString = "select \*from tblPerson";

SqlCommand cmd = new SqlCommand(cmdString, con);

// Creating dataset object

DataSet ds = new DataSet();

//DataTable dt = new DataTable();

// Creating dataAdapter object

SqlDataAdapter da = new SqlDataAdapter(cmd);

// Fills the dataSet using dataAdapter object

da.Fill(ds);

// then data can be viewed by any control like gridView etc…

}

    }

}