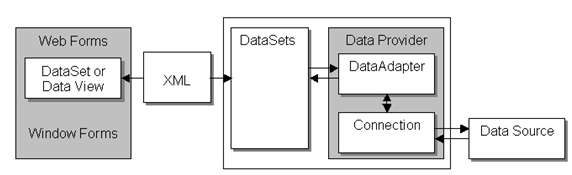
<http://www.c-sharpcorner.com/uploadfile/mahesh/understanding-ado-net-components/>

<http://www.codeproject.com/Articles/8477/Using-ADO-NET-for-beginners>

You can categorize ADO.NET components in three categories: **disconnected**, common or shared and the **.NET data providers**. The disconnected components build the basic ADO.NET architecture. You can use these components (or classes) with or without data providers. For example, you can use a DataTable object with or without providers and

**Shared or common components** are the base classes for data providers and shared by all data providers. The data provider components are specifically designed to work with different kinds of data sources. For example, ODBC data providers work with ODBC data sources and OleDb data providers work with OLE-DB data sources.



A **data provider** is a set of components, such as **Connection**, **Command**, **DataAdapter** and **DataReader**.

The Connection is the first component that talks to a data source. The Connection object establishes a connection to a data source and works as a connection reference in Command and DataAdapter objects.

A Command object executes a SQL query and stored procedures to read, add, update, and delete data of a data source via a DataAdapter.

A DataAdapter is a bridge between a dataset and the connection. It uses Command Object to execute SQL queries and stored procedures.  
  
All data providers share the ADO.NET common components. These components represent the data. Some of the common components are **DataSet**, DataView, and DataViewManager. The DataSet uses XML to store and transfer data between the applications and the data provider.

A **DataSet** is a set of **DataTable** objects.

A **DataTable** represents a database table.

The **DataView** and **DataViewManager** objects provide single or multiple views of a dataset. You can attach a **DataView** or a **DataViewManager** directly to databound controls such as a **DataGrid** or DataList.

Other common components are **DataTable**, **DataRow**, **DataColumn** and so on.

ADO.NET defines **DataSet** and **DataTable** objects which are optimized for moving disconnected sets of data across intranets and Internets, including through firewalls. It also includes the traditional **Connection** and Command objects, as well as an object called a **DataReader** that resembles a forward-only, read-only ADO **recordset**.

There are two central components of **ADO.NET** classes:

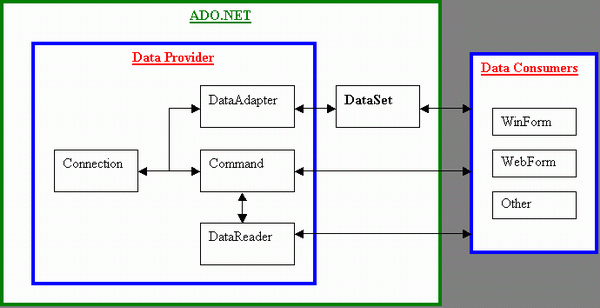
**DataSet**

**Data Provider** (and the .NET Framework)

***Data Provider*** is a set of components that include

* Connection object
* Command object
* DataReader object
* DataAdapter object

***DataSet*** object represents a disconnected cache of data which is made up of **DataTables** and **DataRelations** that represent the result of the command.



# DataAdapter

The [SqlDataAdapter](https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqldataadapter?view=netframework-4.7.1), serves as a bridge between a [DataSet](https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=netframework-4.7.1) and SQL Server for retrieving and saving data. The [SqlDataAdapter](https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqldataadapter?view=netframework-4.7.1) provides this bridge by mapping [Fill](https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbdataadapter.fill?view=netframework-4.7.1), which changes the data in the [DataSet](https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=netframework-4.7.1) 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-4.7.1), 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-4.7.1), using the appropriate Transact-SQL statements against the data source. The update is performed on a by-row basis. For every inserted, modified, and deleted row, the [Update](https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbdataadapter.update?view=netframework-4.7.1) method determines the type of change that has been performed on it (Insert, Update, or Delete). Depending on the type of change, the Insert, Update, or Delete command template executes to propagate the modified row to the data source. When the [SqlDataAdapter](https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqldataadapter?view=netframework-4.7.1) fills a [DataSet](https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=netframework-4.7.1), it creates the necessary tables and columns for the returned data if they do not already exist. However, primary key information is not included in the implicitly created schema unless the [MissingSchemaAction](https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dataadapter.missingschemaaction?view=netframework-4.7.1) property is set to [AddWithKey](https://docs.microsoft.com/en-us/dotnet/api/system.data.missingschemaaction?view=netframework-4.7.1#System_Data_MissingSchemaAction_AddWithKey). You may also have the [SqlDataAdapter](https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqldataadapter?view=netframework-4.7.1) create the schema of the [DataSet](https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=netframework-4.7.1), including primary key information, before filling it with data using FillSchema. For more information, see [Adding Existing Constraints to a DataSet](https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/adding-existing-constraints-to-a-dataset?view=netframework-4.7.1).

[SqlDataAdapter](https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqldataadapter?view=netframework-4.7.1) is used in conjunction with [SqlConnection](https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqlconnection?view=netframework-4.7.1) and [SqlCommand](https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqlcommand?view=netframework-4.7.1) to increase performance when connecting to a SQL Server database

private static DataSet SelectRows(DataSet dataset, string connectionString, string queryString)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

SqlDataAdapter adapter = new SqlDataAdapter();

adapter.SelectCommand = new SqlCommand( queryString, connection);

adapter.Fill(dataset);

return dataset;

}

}