For Beginners – ADO.Net

Contents

Introduction to ADO.NET

- ADO.NET is an object-oriented set of libraries that allows you to interact with data sources.

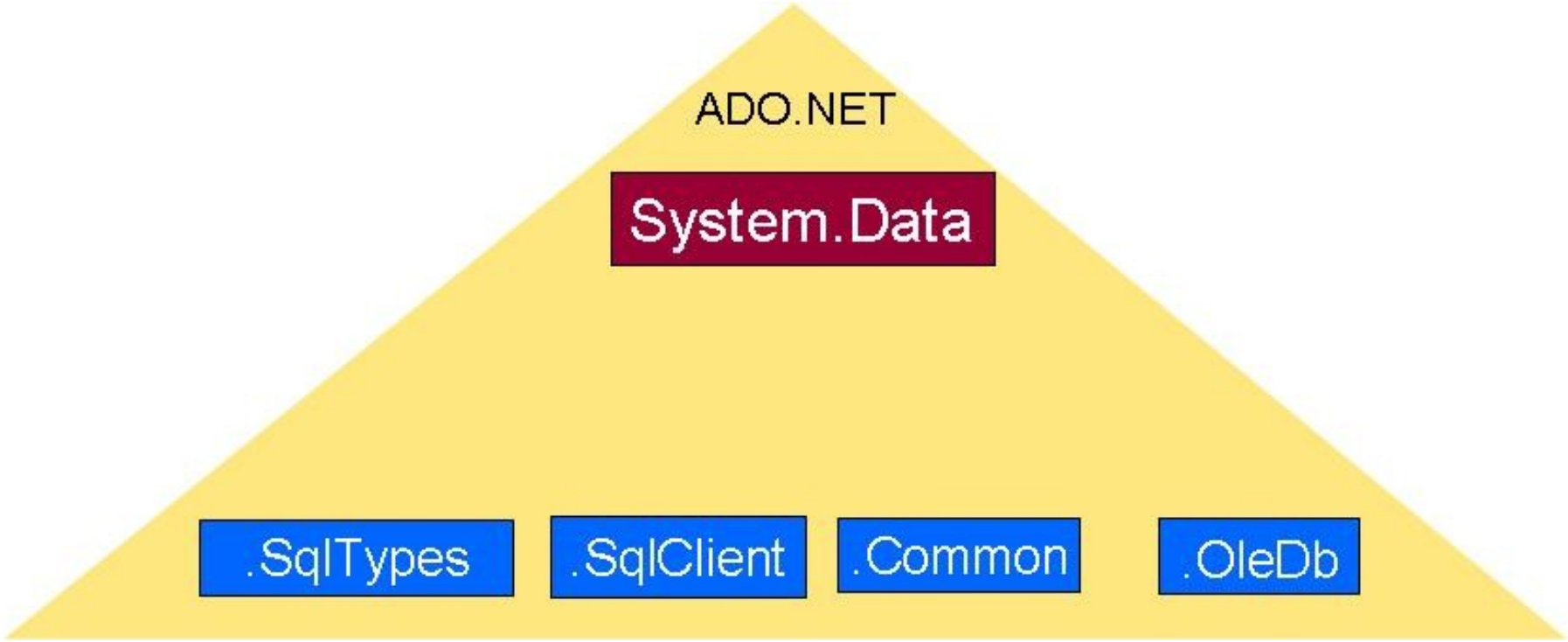
- Commonly, the data source is a database, but it could also be a text file, an Excel spreadsheet, or an XML file.

- ADO.NET provides a relatively common way to interact with data sources, but comes in different sets of libraries for each way you can talk to a data source. These libraries are called Data Providers and are usually named for the protocol or data source type they allow you to interact with.

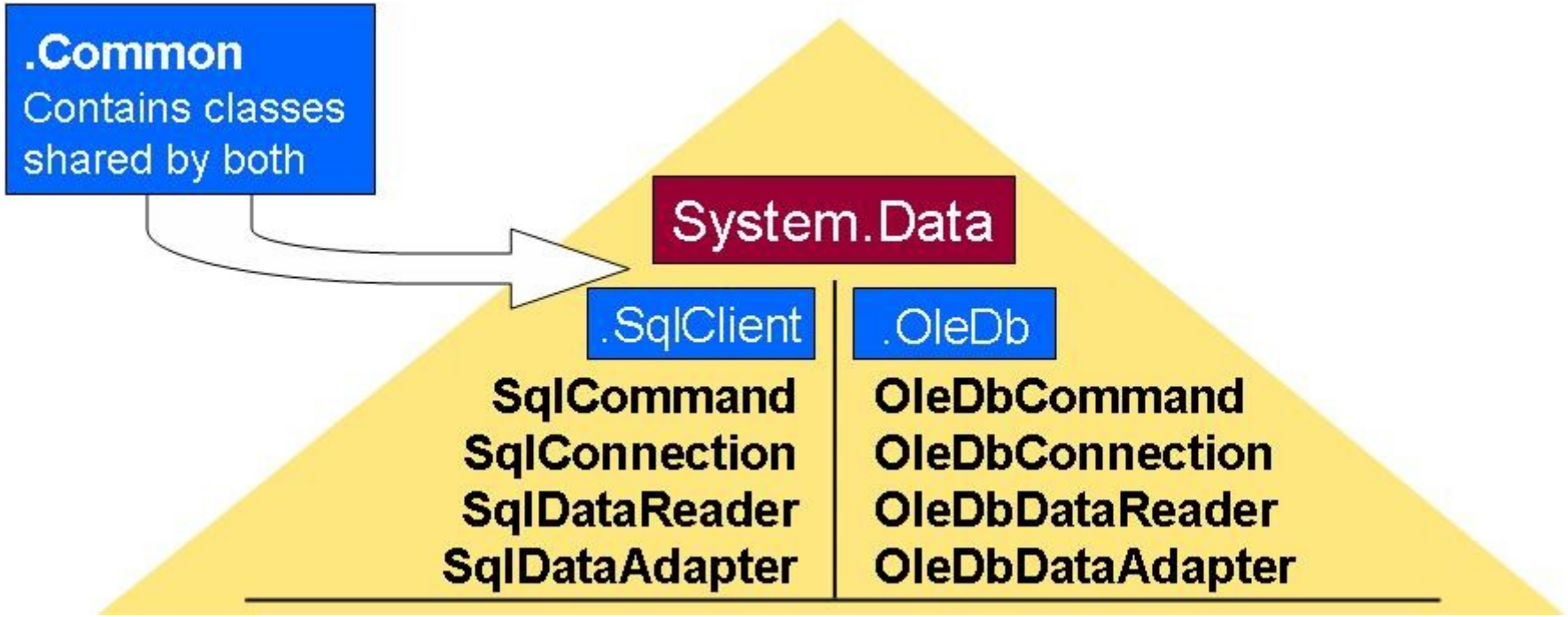
- **ADO.NET Data Providers**

Provider Name	API prefix	Data Source Description
ODBC Data Provider	Odbc	Data Sources with an ODBC interface. Normally older data bases.
OleDb Data Provider	OleDb	Data Sources that expose an OleDb interface, i.e. Access or Excel.
Oracle Data Provider	Oracle	For Oracle Databases.
SQL Data Provider	Sql	For interacting with Microsoft SQL Server.
Borland Data Provider	Bdp	Generic access to many databases such as Interbase, SQL Server, IBM DB2, and Oracle.

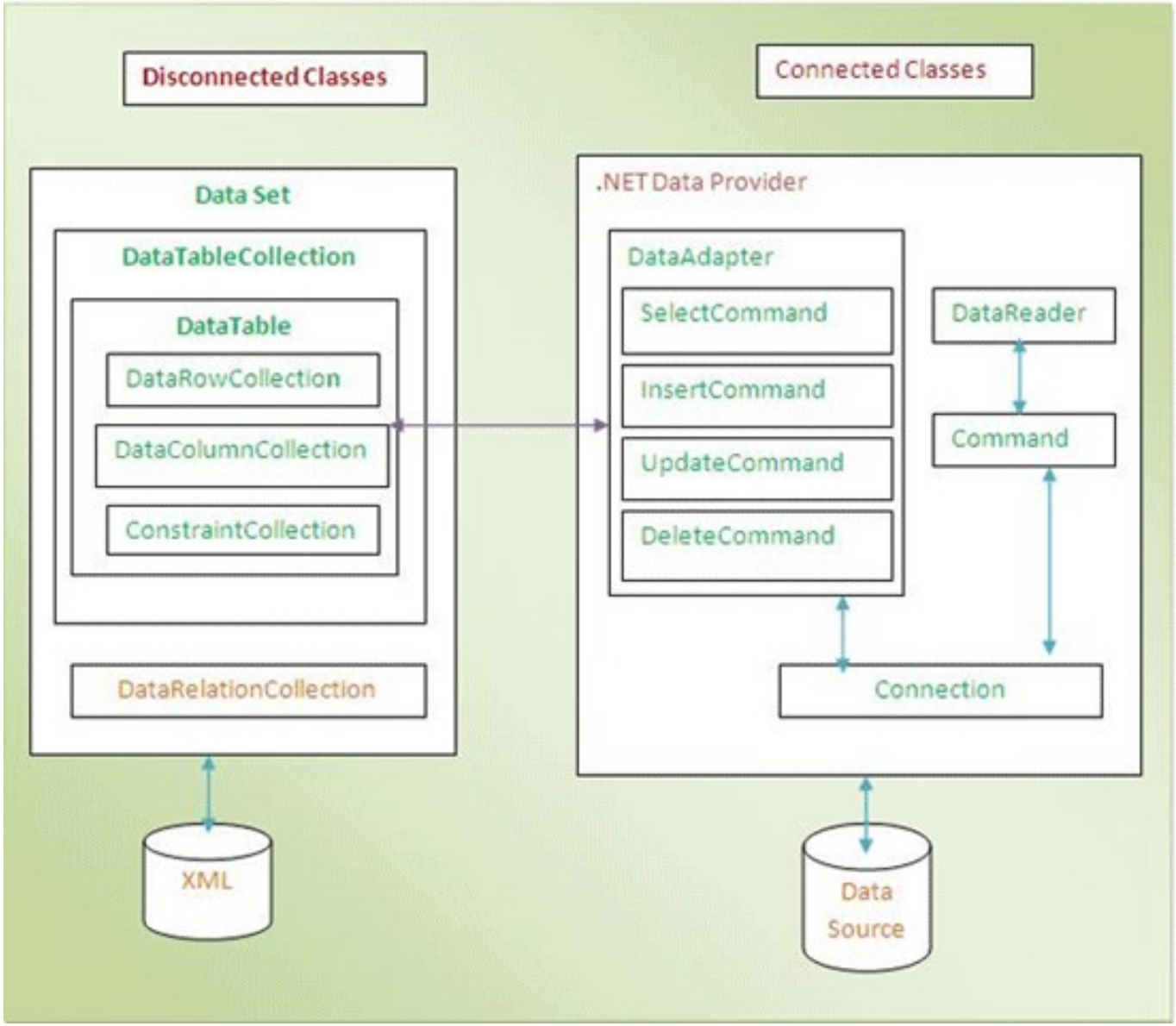
ADO.NET-related Namespaces



.NET Data Providers Hierarchy



ADO.Net Architecture



ADO.NET Objects

- **Connection Object :** The connection helps identify the database server, the database name, user name, password, and other parameters that are required for connecting to the data base. A connection object is used by command objects so they will know which database to execute the command on.

- **Command Object :** Use a command object to send SQL statements to the database. A command object uses a connection object to figure out which database to communicate with.

- **DataReader Object :** The data reader object allows you to obtain the results of a SELECT statement from a command object. For performance reasons, the data returned from a data reader is a fast forward-only stream of data.

- **DataSet Object :** DataSet objects are in-memory representations of data. They contain multiple Datatable objects, which contain columns and rows, just like normal database tables. You can even define relations between tables to create parent-child relationships. The DataSet is an object that is used by all of the Data Providers, which is why it does not have a Data Provider specific prefix.

- **DataAdapter Object :** The data adapter fills a DataSet object when reading the data and writes in a single batch when persisting changes back to the database. A data adapter contains a reference to the connection object and opens and closes the connection automatically when reading from or writing to the database.

- **CommandBuilder :** By default dataadapter contains only the select command and it doesn’tcontain insert, update and delete commands. To create insert, update and delete commands for the dataadapter, commandbuilder is used. It is used only to create these commands for the dataadapter and has no other purpose.

SqlConnection Object

- The connection tells the rest of the ADO.NET code which database it is talking to. It manages all of the low level logic associated with the specific database protocols.

- Declare and instantiate the SqlConnection as shown below:

```
SqlConnection conn = new SqlConnection
        ( "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");
```

- **ADO.NET Connection String Parameters**

Parameter Name	Description
Data Source	Identifies the server. Could be local machine, machine domain name, or IP Address.
Initial Catalog	Database name.
Integrated Security	Set to SSPI to make connection with user's Windows login
User ID	Name of user configured in SQL Server.
Password	Password matching SQL Server User ID.

SqlCommand & SqlDataReader Object

- SqlCommand object allows you to specify what type of interaction you want to perform with a database. For example, you can do select, insert, modify, and delete commands on rows of data in a database table.
- **Creating a SqlCommand Object**

  SqlCommand cmd = new SqlCommand("select CategoryName from Categories", conn);

- A SqlDataReader is a type that is good for reading data in the most efficient manner possible. You can not use it for writing data. SqlDataReaders are often described as fast-forward firehose-like streams of data.
- You can read from SqlDataReader objects in a forward-only sequential manner. Once you've read some data, you must save it because you will not be able to go back and read it again.
- **Creating a SqlDataReader Object**
- Getting an instance of a SqlDataReader is a little different than the way you instantiate other ADO.NET objects. You must call *ExecuteReader* on a command object, like this:

  SqlDataReader rdr = cmd.ExecuteReader();

SqlCommand Methods

- **1) ExecuteReader -Querying Data**
- Used to retrieve data records for viewing. It returns a SqlDataReader object.
- Example:

```
// 1. Instantiate a new command with a query and connection
SqlCommand cmd = new SqlCommand("select CategoryName from Categories", conn);

// 2. Call Execute reader to get query results
SqlDataReader rdr = cmd.ExecuteReader();
```

- **2) ExecuteNonQuery (Insert/Update/Delete)**
- To insert data into a database, use the ExecuteNonQuery method of the SqlCommand object.

```
// prepare command string
 string insertString = @"
     insert into Categories
     (CategoryName, Description)
     values ('Miscellaneous', 'Whatever doesn"t fit elsewhere')";

 // 1. Instantiate a new command with a query and connection
 SqlCommand cmd
= new SqlCommand(insertString, conn);

 // 2. Call ExecuteNonQuery to send command
 cmd.ExecuteNonQuery();
```

- **Similarly ExecuteNonQuery can be used to update a record using below query string**

```
// prepare command string
 string updateString = @"
     update Categories
     set CategoryName = 'Other'
     where CategoryName = 'Miscellaneous'";
```

- **Similarly ExecuteNonQuery can be used to delete a record using below query string**

```
// prepare command string
 string deleteString = @"
     delete from Categories
     where CategoryName = 'Other'";
```

3) ExecuteScalar - Getting Single values

- Used to get count, sum, average, or other aggregated value from a database.

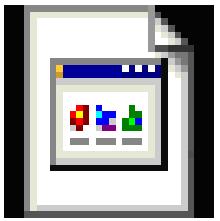
```
// 1. Instantiate a new command
 SqlCommand cmd = new SqlCommand("select count(*) from Categories", conn);

 // 2. Call ExecuteNonQuery to send command
 int count = (int)cmd.ExecuteScalar();
```

- The query in the SqlCommand constructor obtains the count of all records from the Categories table. This query will only return a single value.

Connected Example:

- Example in attached file to retrieve, insert, update, delete record from data table. Double click on below link to open notepad file.



ConnectedExample.txt

Working with Disconnected Data

- **DataSet and SqlDataAdapter**

- A DataSet is an in-memory data store that can hold numerous tables. DataSets only hold data and do not interact with a data source.

- It is the SqlDataAdapter that manages connections with the data source and gives us disconnected behavior. The SqlDataAdapter opens a connection only when required and closes it as soon as it has performed its task.

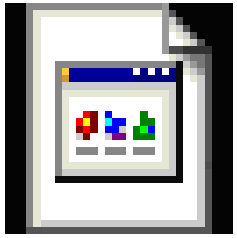
- For example, the SqlDataAdapter performs the following tasks when filling a DataSet with data:
- Open connection
- Retrieve data into DataSet
- Close connection

- and performs the following actions when updating data source with DataSet changes:
- Open connection
- Write changes from DataSet to data source
- Close connection

- In between the Fill and Update operations, data source connections are closed and you are free to read and write data with the DataSet as you need.

Disconnected Example

- Example below in text file use DataSet and SqlDataAdapter to get customer records & display in datagrid. Datagrid is editable &changes can be saved using “Update” button. Double click below file to open.



DisconnectedEmaple.txt

Adding Parameters to Commands

- Using parameterized queries is a three step process:

1) Construct the SqlCommand command string with parameters.

   SqlCommand cmd = new SqlCommand("select * from Customers where city = @City", conn);

1) Declare a SqlParameter object, assigning values as appropriate.

   SqlParameter param = new SqlParameter(); param.ParameterName = "@City"; param.Value = inputCity;

3) Assign the SqlParameter object to the SqlCommand object's Parameters property.

   cmd.Parameters.Add(param);

4) Below is complete console application example double click to open.

AddingParameterExample.txt

Using Stored Procedures

- A stored procedures is a pre-defined, reusable routine that is stored in a database.
- SQL Server compiles stored procedures, which makes them more efficient to use.
- Therefore, rather than dynamically building queries in your code, you can take advantage of the reuse and performance benefits of stored procedures.

- **Executing a Stored Procedure**

```
// 1. create a command object identifying the stored procedure
SqlCommand cmd = new SqlCommand( "<<StoredProcName>>", conn);
// 2. set the command object so it knows to execute a stored procedure
cmd.CommandType = CommandType.StoredProcedure;
```

- **Sending Parameters to Stored Procedures**

```
SqlCommand cmd = new SqlCommand( "CustOrderHist", conn);
// 2. set the command object so it knows to execute a stored procedure
cmd.CommandType = CommandType.StoredProcedure;
// 3. add parameter to command, which will be passed to the stored procedure
cmd.Parameters.Add( new SqlParameter("@CustomerID", custId));
```

- The name of the parameter passed as the first parameter to the SqlParameter constructor must be spelled exactly the same as the stored procedure parameter.

Stored Procedure Example

- Below example uses 2 procedures to get
- top 10 expensive products
- Customer specific orders

Double click to open example below

StoredProcedureExample.txt

Thank You