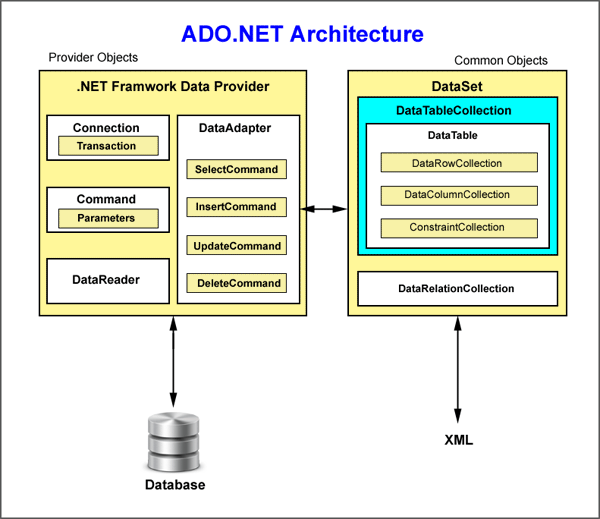
ADO.NET is a data access technology from the Microsoft .NET Framework that provides communication between relational and non-relational systems through a common set of components. ADO.NET is a set of computer software components that programmers can use to access data and data services from a database. It is a part of the base class library that is included with the Microsoft .NET Framework. It is commonly used by programmers to access and modify data stored in relational database systems, though it can also access data in non-relational data sources. ADO.NET is sometimes considered an evolution of ActiveX Data Objects (ADO) technology, but was changed so extensively that it can be considered an entirely new product.

ADO.NET Architecture  
  
  
  
Advantages of ADO.NET

ADO.NET offers several advantages over previous Microsoft data access technologies, including ADO. Few advantages are listed below:

Single Object-oriented API

ADO.NET provides a single object-oriented set of classes. There are different data providers to work with different data sources but the programming model for all these data providers work in the same way. You should be aware of only one data provider. You just need to change class names and connection strings.

The ADO.NET classes are easy to use and understand, as they are object-oriented in nature.

Managed Code

The ADO.NET classes are managed classes. CLR takes care of language independency and automatic resource management.

Deployment

Microsoft uses MDAC (Microsoft Data Access Component), which is used as ActiveX component in .NET Framework (X is extensible component, when X is written after a term means extensible). .NET components takes care of deployment which was difficult previous technologies used in deployment.

XML Support

ADO.NET data is cached and transferred in XML (EXtensible Markup Language) format. XML provide fast access of data for desktop and distributed applications. XML is plain text designed to transport and store data and is self-descriptive.

Visual Data Components

.NET offers ADO.NET components and data-bound control to work in visual form.  You can use these components without writing long codes and can achieve result in no time.

Performance and scalability

Performance and scalability are two major factors when developing web-based application and services. Disconnected cached data in XML help in performance and scalability.

Difference between ADO and ADO.NET

|  |  |
| --- | --- |
| ADO | ADO.NET |
| ADO has one main object that is used to reference data, called the RecordSet object. | ADO.NET provides objects that allow you to access data in various ways. The DataSetobject allows you to store the relational model of your [database](https://www.c-sharpcorner.com/UploadFile/puranindia/what-is-ado-net/WhatisADONET.aspx).  MARS (Multiple Active Result Sets) is implemented in ADO.NET |
| You can only work on connected manner. This means that when you access data, such as viewing and updating data, it is real-time, with a connection being used all the time. This is barring, of course, you programming special routines to pull all your data into temporary tables.    In connected model you always get refreshed data. | ADO.NET uses data in a disconnected fashion. When you access data, ADO.NET makes a copy of the data using XML. ADO.NET only holds the connection open long enough to either pull down the data or to make any requested updates. This makes ADO.NET efficient to use for [Web applications](https://www.c-sharpcorner.com/UploadFile/puranindia/what-is-ado-net/WhatisADONET.aspx). It's also decent for desktop applications.    You can work on connected and disconnected manner.    In disconnected model you will get old data as you are editing it. Outlook is an example of disconnected model. We work on offline object model and when connection is required it is connected.    Connected object can be used on disconnected object. |
| Whereas ADO allows you to persist records in XML format. | ADO.NET allows you to manipulate your data using XML as the primary means. This is nice when you are working with other business applications and also helps when you are working with firewalls because data is passed as HTML and XML. |
| ADO allows you to create client-side cursors only. | ADO.NET gives you the choice of either using client-side or server-side cursors. In ADO.NET, classes actually handle the work of cursors. The developer has the freedom of choice in [internet](https://www.c-sharpcorner.com/UploadFile/puranindia/what-is-ado-net/WhatisADONET.aspx) development, for creating efficient applications. |

ADO.NET DataAdapter and Dataset

DataAdapter is a part of the ADO.NET Data Provider. Dataset represents a collection of data retrieved from the Data Source and saving data to the Data Source. We can use Dataset in combination with DataAdapter class.

These two objects combine to enable both data access and data manipulation capabilities. Functionally DataAdapter is the complex Object when compare to other Objects in the Data Provider.

Overview of a DataSet

A DataSet, a resident of the System.Data namespace, is most precisely defined as a provider-neutral, in-memory, and disconnected relational data structure. It provides support for the standard view, add, remove, and update data operations for the data it represents, and it isn’t limited only to database data. DataSet is composed of several components, including:

**DataTable**: One or more DataTable objects represent the data from a source in familiar row and column format.

**DataRow:** Each DataTable contains multiple DataRow objects that contain the data from the source in a record or record-like layout.

**DataColumn:** Multiple DataColumn objects define the columns for each DataRow.

#### DataAdapter

A data adapter (DataAdapter class) is an object used to exchange data between a data source and a data set. In other words, a data adapter is used to read data from a data source into a data set, as well as writing changed data from the data set back to the data source.

A data adapter is an integral part of a .NET Data Provider. In general, a data adapter is configurable to allow you to specify what data to move into and out of the data set. This configuration takes the form of a collection of either SQL statements or stored procedures

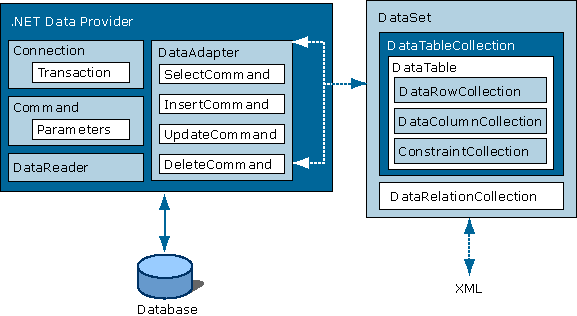
### DATA ADAPTERS PROPERTIES AND METHODS

#### Properties:

|  |  |
| --- | --- |
| Properties | Description |
| **DeleteCommand** | It is used for Deleting Records from DataSource |
| **InsertCommand** | It is used for adding New Record to a DataSource |
| **SelectCommand** | It is used for Selecting Records from a DataSource |
| **UpdateCommand** | It is used for Updating Records in a DataSource. |
| **TableMapping** | It is used for mapping actual database tables and datasets. |

#### Methods:

|  |  |
| --- | --- |
| Method | Description |
| **Fill** | This method Fills Records from DataAdapters to DataSets. |
| **Update** | This method update DataSource with DataSets. |



.

#### Mechanisms

In general, a data adapter exchanges data between a single data-source table and a single DataTable within a DataSet. If a DataSet contains multiple tables, the usual strategy is to have multiple data adapters with each data adapter exchanging data between a single DataTable and its corresponding data-source table.

**Reading from a data source**  
When you want to populate a data table in a data set from a table in a data source, you call a data adapter method to execute SQL or stored procedure to read data into the data table.

**Writing to a data source**  
When you want to update a table in a data source from a data table in a data set, you call a data adapter method to execute SQL or stored procedure to make the actual update in the database.

#### Command Objects

With an adapter you can read, add, update and delete records in a data source. You can specify how each of these operations should occur by specifying the appropriate command object - implemented as a property on the data adapter:

**SelectCommand**  
SQL or stored procedure that retrieves rows from a data source. This is the most commonly used command. Usually when you instantiate an instance of a DataAdapter, you either pass a command object or a SQL string - this command object or SQL string is actually used to initialize the SelectCommand object.

**InsertCommand**  
SQL or stored procedure for inserting rows into the data source.

**UpdateCommand**  
SQL or stored procedure for modifying rows in a data source.

**DeleteCommand**  
SQL or stored procedure for deleting rows from a data source.

.

#### When to Use a Data Adapter

In general, use a data adapter when you need to perform one of the following:

Retrieve data from a data source into a data table within a data set

**Transmit changes made to a data set table back to the data source**Transmits contents of a data table to the data source using the Update method. When this method is called, the state of each row is examined to determine if the row has been deleted, updated, or inserted. Depending on the state of each row, the appropriate DeleteCommand, UpdateCommand or InsertCommand is executed

The job of translating between the provider-neutral DataSet and provider-specific connection and command objects falls to the DataAdapter. The DataAdapter for a DataSet takes care of retrieving, updating, inserting, and deleting data by making use of up to four internal command objects, which you can set or retrieve through this set of four public properties:

SelectCommand: Retrieves data from the data source to populate a DataSet

InsertCommand: Inserts new data to the data source from new data added to a DataSet

UpdateCommand: Updates existing data from changes made to the data in the DataSet

DeleteCommand: Deletes existing data when data is deleted from a DataSet

The DataAdapter class is abstract, so you can’t create it directly. Instead, you’ll need to use one of the provider-specific derived child classes. For example, to build a DataSet based on the contents of a hypothetical Customers table in a SQL Server database, you’d make use of the System.Data.SqlClient.SqlDataAdapter class. You could use something similar to the following C# code to accomplish this task, assuming that the SqlConnectioncn has already been created:

SqlCommand cm = new SqlCommand("SELECT \* FROM Customers;",conn);

SqlDataAdapter sda = new SqlDataAdapter();

sda.SelectCommand = cm;

DataSet ds = new DataSet();

sda.Fill(ds);

Progra

mming Example in C# Console:

using **System**;

using **System**.**Data**.**SqlClient**;

using **System**.**Data**;

namespace **DataSet\_Example**

{

class **Program**

{

static void **Main**(string[] args)

{

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial

Catalog=ComputerShop;Integrated Security=True";

string querystring = "Select \* from Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(querystring, **ConString**);

**DataSet** ds = new **DataSet**();

adapter.**Fill**(ds, "Items");

**Console**.**WriteLine**(ds.**GetXml**());

**Console**.**ReadKey**();

}

}

}

Output

1

LED **Screen**

$120

2017-01-27T00:00:00+05:30

2

USB **Keyboard**

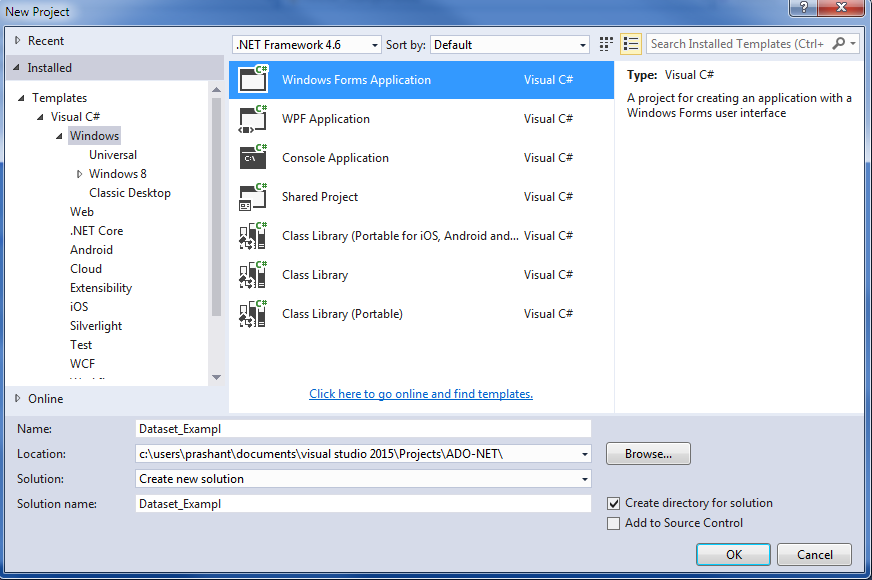
$20

2017-05-25T00:00:00+05:30

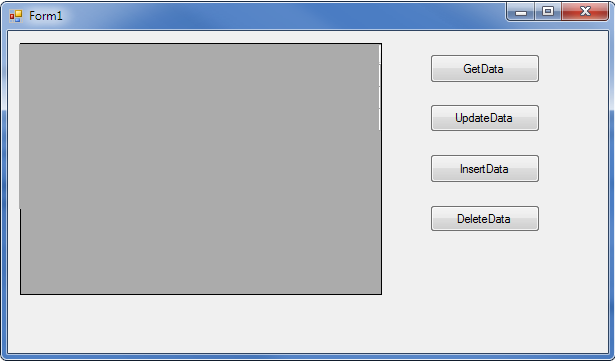
### DATASET EXAMPLE WITH GRID VIEW

Mostly DataSet is used with GridView in ASP.Net. Explaination of DataAdapters and DataSets with Examples.

**1.** Open **Visual Studio 2015** and Go to **File** > **New** >  
**Project**. Create a New Windows Forms Application Dataset\_Example.

[](https://www.completecsharptutorial.com/wp-content/uploads/2017/07/1step.png)

**2.** Drag a GridView and a Button like that.

[](https://www.completecsharptutorial.com/wp-content/uploads/2017/07/2step.png)

##### Extract Data from DataSet to GridView

using **System**;

using **System**.**Data**;

using **System**.**Data**.**SqlClient**;

using **System**.**Windows**.**Forms**;

namespace **DataSet\_Exampl**

{

public partial class **Form1** : **Form**

{

public **Form1**()

{

**InitializeComponent**();

}

private void btnGetData\_Click(object sender, **EventArgs** e)

{

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**,**ConString**);

**DataSet** set = new **DataSet**();

adapter.**Fill**(set, "Items");

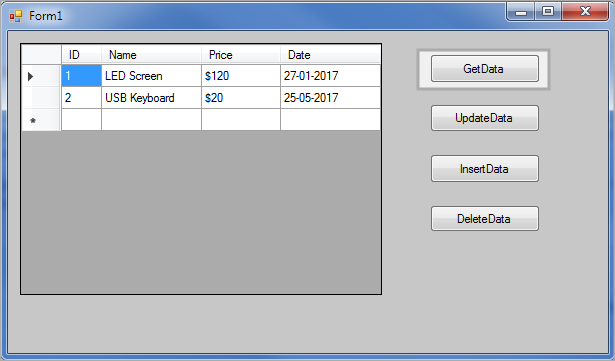
dataGridView1.**DataSource** = set.**Tables**["Items"];

}

}

}

Output

[](https://www.completecsharptutorial.com/wp-content/uploads/2017/07/3step.png)

### UPDATING, INSERTING, AND DELETING RECORDS IN A DATASET

After populating dataset, you can update, insert or delete a record from the dataset. Here is a full programming example.

##### Adding New Row in DataTable

private void btnInsert\_Click(object sender, **EventArgs** e)

{

//Fill Dataset

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**, **ConString**);

**DataSet** set = new **DataSet**();

adapter.**Fill**(set, "Items");

//Adding New Row to DataSet

**DataRow** row = set.**Tables**["Items"].**NewRow**();

row["Name"] = "4GB DDR3 RAM";

row ["Price"] = "$50";

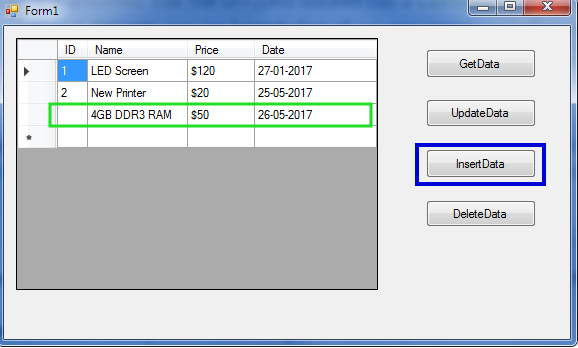
row ["Date"] = "26 May 2017";

set.**Tables**["Items"].**Rows**.**Add**(row);

dataGridView1.**DataSource** = set.**Tables**["Items"];

}

Output

[](https://www.completecsharptutorial.com/wp-content/uploads/2017/07/Insert.png)

Insert Row in Dataset

##### Edit or Update Row in DataSet

If you don’t know row index or unique row number still you can update or edit row in dataset by using following method.

private void btnUpdate\_Click(object sender, **EventArgs** e)

{

//Fill Dataset

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**, **ConString**);

**DataSet** set = new **DataSet**();

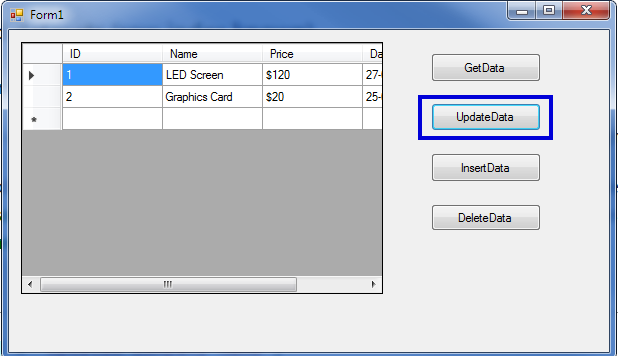
adapter.**Fill**(set, "Items");

set.**Tables**["Items"].**Rows**[1]["Name"] = "Graphics Card";

dataGridView1.**DataSource** = set.**Tables**["Items"];

}

Output

[](https://www.completecsharptutorial.com/wp-content/uploads/2017/07/update.png)

##### Delete Row in DataSet

You can delete row from dataset using **Delete()** Method.

private void btnDelete\_Click(object sender, **EventArgs** e)

{

//Fill Dataset

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**, **ConString**);

**DataSet** set = new **DataSet**();

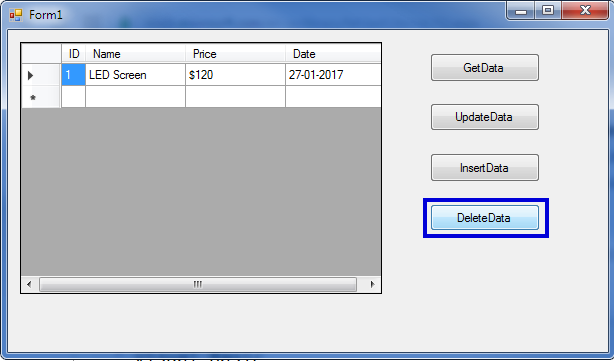
adapter.**Fill**(set, "Items");

set. **Tables**["Items"].**Rows**[1].**Delete**();

dataGridView1.**DataSource** = set.**Tables**["Items"];

}

Output

[](https://www.completecsharptutorial.com/wp-content/uploads/2017/07/delete.png)

### SAVE DATASET CHANGES TO DATABASE

After Modifying Dataset, you can save Dataset changes to database.

Programming Example

private void btnSave\_Click(object sender, **EventArgs** e)

{

//Fill Dataset

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**, **ConString**);

**DataSet** set = new **DataSet**();

adapter.**Fill**(set, "Items");

//Adding New Row to DataSet and Update

**DataRow** row = set.**Tables**["Items"].**NewRow**();

row["Name"] = "4GB DDR3 RAM";

row["Price"] = "$50";

row["Date"] = "26 May 2017";

set.**Tables**["Items"].**Rows**.**Add**(row);

//Updating Database Table

**SqlCommandBuilder** builder = new **SqlCommandBuilder**(adapter);

adapter.**Update**(set.**Tables**["Items"]);

**MessageBox**.**Show**("DataSet Saved to Database Successfully");

}

Output

DataSet Saved to Database Successfully **\_**

### COMPLETE PROGRAM

using **System**;

using **System**.**Data**;

using **System**.**Data**.**SqlClient**;

using **System**.**Windows**.**Forms**;

namespace **DataSet\_Exampl**

{

public partial class **Form1** : **Form**

{

public **Form1**()

{

**InitializeComponent**();

}

private void btnGetData\_Click(object sender, **EventArgs** e)

{

//Fill DataSet

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**,**ConString**);

**DataSet** set = new **DataSet**();

adapter.**Fill**(set, "Items");

dataGridView1.**DataSource** = set.**Tables**["Items"];

}

private void btnUpdate\_Click(object sender, **EventArgs** e)

{

//Fill Dataset

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**, **ConString**);

**DataSet** set = new **DataSet**();

adapter.**Fill**(set, "Items");

set.**Tables**["Items"].**Rows**[1]["Name"] = "Graphics Card";

dataGridView1.**DataSource** = set.**Tables**["Items"];

}

private void btnInsert\_Click(object sender, **EventArgs** e)

{

//Fill Dataset

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**, **ConString**);

**DataSet** set = new **DataSet**();

adapter.**Fill**(set, "Items");

//Adding New Row to DataSet

**DataRow** row = set.**Tables**["Items"].**NewRow**();

row["ID"] = 3;

row["Name"] = "4GB DDR3 RAM";

row["Price"] = "$50";

row["Date"] = "26 May 2017";

set.**Tables**["Items"].**Rows**.**Add**(row);

dataGridView1.**DataSource** = set.**Tables**["Items"];

}

private void btnDelete\_Click(object sender, **EventArgs** e)

{

//Fill Dataset

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**, **ConString**);

**DataSet** set = new **DataSet**();

adapter.**Fill**(set, "Items");

set.**Tables**["Items"].**Rows**[1].**Delete**();

dataGridView1.**DataSource** = set.**Tables**["Items"];

}

private void btnSave\_Click(object sender, **EventArgs** e)

{

//Fill Dataset

string **ConString** = @"Data Source=.\SQLEXPRESS;Initial Catalog=ComputerShop;Integrated Security=True";

string **Query** = "SELECT \* FROM Items";

**SqlDataAdapter** adapter = new **SqlDataAdapter**(**Query**, **ConString**);

**DataSet** set = new **DataSet**();

adapter.**Fill**(set, "Items");

//Adding New Row to DataSet and Update

**DataRow** row = set.**Tables**["Items"].**NewRow**();

row["Name"] = "4GB DDR3 RAM";

row["Price"] = "$50";

row["Date"] = "26 May 2017";

set.**Tables**["Items"].**Rows**.**Add**(row);

//Updating Database Table

**SqlCommandBuilder** builder = new **SqlCommandBuilder**(adapter);

adapter.**Update**(set.**Tables**["Items"]);

**MessageBox**.**Show**("DataSet Saved to Database Successfully");

}

}

}

**DataReader**

There are two ways to read and store data: one is DataSet and the other is DataReader. A data reader provides an easy way for the programmer to read data from a database as if it were coming from a stream. The DataReader is the solution for forward streaming data through ADO.NET. The data reader is also called a firehose cursor or forward read-only cursor because it moves forward through the data. The data reader not only allows you to move forward through each record of database, but it also enables you to parse the data from each column. The DataReader class represents a data reader in ADO.NET.  
  
Similar to other ADO.NET objects, each data provider has a data reader class for example; OleDbDataReader is the data reader class for OleDb data providers. Similarly, SqlDataReader and ODBC DataReader are data reader classes for Sql and ODBC data providers, respectively.

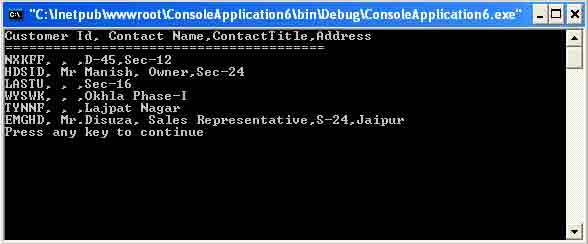
**Initializing DataReader**  
As you've seen in the previous examples, you call the ExecuteReader method of the Command object, which returns an instance of the DataReader. For example, use the following line of code:  
  
            SqlCommand cmd = new SqlCommand(SQL, conn);  
            // Call ExecuteReader to return a DataReader  
            SqlDataReader reader = cmd.ExecuteReader();  
  
Once you're done with the data reader, call the Close method to close a data reader:  
  
            reader.Close(); **DataReader Properties and Methods**  
  
 The 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 |

DataReader methods

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| Close | Closes a DataRaeder object. |
| Read | Reads next record in the data reader. |
| NextResult | Advances the data reader to the next result during batch transactions. |
| Getxxx | There are dozens of Getxxx methods. These methods read a specific data type value from a column. For example. GetChar will return a column value as a character and GetString as a string. |

**DataReader reads data from a SQL server database**  
  
using System;  
using System.Collections.Generic;  
using System.Text;  
using System.Data.SqlClient;  
  
namespace CommandTypeEnumeration  
{  
    class Program  
   {  
        static void Main(string[]args)  
        {  
  
            //Create a connection string  
            string ConnectionString = "Integrated Security = SSPI; " +  
            "Initial Catalog= Northwind; " + " Data source = localhost; ";  
            string SQL = "SELECT \* FROM Customers";  
  
            // create a connection object  
            SqlConnection conn = new SqlConnection(ConnectionString);  
  
            // Create a command object

            SqlCommand cmd = new SqlCommand(SQL, conn);  
            conn.Open();  
  
            // Call ExecuteReader to return a DataReader  
            SqlDataReader reader = cmd.ExecuteReader();  
            Console.WriteLine("customer ID, Contact Name, " + "Contact Title, Address ");  
            Console.WriteLine("=============================");  
  
            while (reader.Read())  
            {  
                Console.Write(reader["CustomerID"].ToString() + ", ");  
                Console.Write(reader["ContactName"].ToString() + ", ");  
                Console.Write(reader["ContactTitle"].ToString() + ", ");  
                Console.WriteLine(reader["Address"].ToString() + ", ");  
            }  
  
            //Release resources  
            reader.Close();  
            conn.Close();  
        }  
    }  
}  
  
  
  
Top of Form

Data Binding

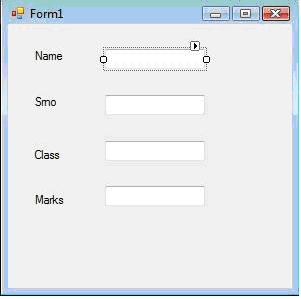
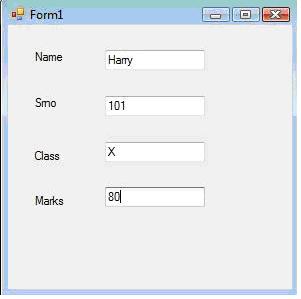
Data Binding Mechanism in ADO.NET. ADO.NET gives you the means to connect to a **database** and represent its data as **tables** that can be bound to data-aware controls. The System.Data.DataSet is a major component of ADO.NET. It represents a collection of **tables** that you can relate to each other.

The user can bind values to the respective controls in ADO.NET. Depending on the type of binding offered, they are distinguished as follows:

* 1. Simple Data Binding
  2. Complex Data Binding

### 1. Simple Data Binding

The Simple Data Binding is the process of binding the control with the single value in the dataset. The controls like text box, label can be bound to the control through the control properties.   
  
Consider an example to display the result of the students in an examination. The details are added in the following format.

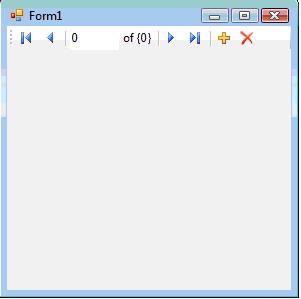
* 1. Create a Windows Form Application in Visual Studio .NET. The following customized format is created for user.  
     
  2. Once the design of the form is created, select the View option from the menu bar. Click on the Properties window.
  3. Select the first text box and the properties for it appear in the window.
  4. Expand the DataBindings property
  5. Select the Text property for enabling the drop down list.
  6. Click the Add Project Data Source from the drop down list
  7. Make a connection with the CurrentInfo database and select the Student table
  8. Select the Other Data Sources, Project Data Sources, CurrentInfoDataSet, Student table.
  9. Select the Name column and bind it with the textbox.
  10. Bind all the other text boxes with the database values.
  11. Press F5 and execute the Windows Form.
  12. The following output is displayed to the user.  
      

### 2. Complex Data Binding

The Complex Data Binding is the process of binding the component with the Database. The controls can be GridView, Dropdown list, or combo box. Multiple values can be displayed from the dataset through the binding.   
  
The controls that can be used for binding the multiple values from the database to the Windows Form are listed below.

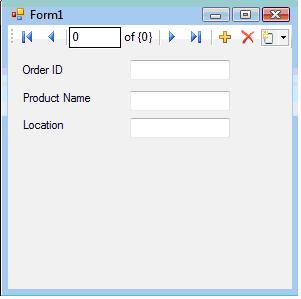
* 1. DataGridView: It is used to display the multiple records and columns. The DataSource property of the DataGridView control is used for binding the specific data element.
  2. ComboBox: The control contains a text box for entering the data and drop down list for displaying the values. The DataSource property is useful for binding the control. The element specific information can be bind through the DisplayMember property
  3. ListBox: It is used for displaying the data for the column from several records of datasets. The DataSource property is used for binding the control to the data source.
  4. The DisplayMember property is used for binding the control to the specific data element.

### Navigating Records in ADO.NET

A BindingNavigator control is used for handling the binding to the data source through the pointer to the current item in the list of records.   
  
The navigator control is used with the BindingSource control for enabling the users to navigate the data records on a form. It provides a layer between the controls and windows form of the data source. Users can navigate and modify the records in the Windows form.  
  
The following figure displays the BindingNavigator control and the BindingSource control in the Windows Form.  
  
  
  
The Binding Navigator control has many controls for modifying the data source. The list of controls and their functions are mentioned below:

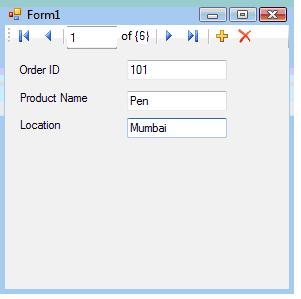
* 1. bindingNavigatorAddNewItem Button: The + sign indicates that the new row can be added to the data source.
  2. bindingNavigatorDeleteItem Button: The X sign indicates that the current row can be deleted from the data source.
  3. bindingNavigatorMoveFirstItem Button: The button indicates that the user can move to the first item in the data source.
  4. bindingNavigatorMoveLastItem Button: The button indicates that the user can move to the last item in the data source
  5. bindingNavigatorMoveNextItem Button: The button indicates that the user can move to the next item in the data source
  6. bindingNavigatorMovePreviousItem Button: The button indicates that the user can move to the previous item in the data source
  7. bindingNavigatorPositionItem textbox: The returns current position in the data source
  8. bindingNavigatorCountItemText box: The is used to return the total number of items in the data source.

Consider the Order details table containing the data about the orders to be added. The data is organized into a format as shown below:

* 1. Open Visual studio application and add Windows Forms Application from the template pane.
  2. Add the labels and a binding Navigator control, and textbox controls to the form
  3. Click OK button
  4. Click View, Properties Window, and open the Properties Window.
  5. Add the appropriate names to the controls present in the web form.   
     
  6. Open the Data Source Configuration Wizard. The Database icon must be selected. Click Next Button
  7. Click New Connection Button. Add Connection dialog box is shown.
  8. Add the Server Name, select Use [SQL Server](http://www.go4expert.com/articles/introduction-sql-server-t34535/) Authentication option from the Log on the server section
  9. Add the User name as sa and password as abcd1234
  10. Select the Order Details database and click Test Connection button
  11. Click OK and close the Add Connection dialog box
  12. Click Next button. In the Choose Your Database Objects dialog box, expand Tables node.
  13. Select the Orderdata table and click Finish button.

For binding the data to the control in the Windows Form, the following steps are executed.

* 1. Select the textbox1, and expand the DataBindings property.
  2. Select the Text property and click on the drop down list.
  3. Expand the Other Data Sources, Project Data, Sources, Orderdataset, Orderdata nodes.
  4. Select the OrderID column and bind it with textbox1.
  5. Perform the similar operations for all the textboxes.

Press F5 or click Debug -> Start Debugging option from the menu. The Order Details form is displayed as shown below:  
  
  
  
User can navigate through the other records using the navigator button of the control.  
Close the form and exit the Visual Studio application.

### 

### Filtering Data

There are requirements when user wants to display only limited data to the client. The filtering of data is possible for displaying the desired results. The data can be sorted in ascending or descending order.   
  
There are two ways by which the data can be filtered. They are as mentioned below:

* 1. Parameterized Queries
  2. Filtering data using controls in Windows Form.

**1. Parameterized Queries**  
  
The stored procedures are always useful for accessing data from the database. By using the stored procedures users can precompiled execution, use of code, less network traffic, and security is high for the data stored in the system.  
  
The parameterized queries are useful for filtering the data based on the conditions defined by the user at runtime. They are useful when user wants to execute the data based on the situation.   
  
The following query is useful for selecting the student with the ID specified.

Code:

SELECT StudName FROM Student

WHERE StudID = @StudID

In the above code, the @StudID is the parameter in the query. The value is passed at the runtime by the user.   
  
In ADO.NET, for populating the @StudID parameter, the SqlParameter object in the SqlParameter class for the command. The SqlParameter object is used for assigning the parameterized values to the queries.   
  
Consider the example for extracting the details of the student. The following code shows the execution parameterized query.

Code:

SqlConnection con = new SqlConnection();

con.ConnectionString = “DataSource = SQLSERVER02; Initial Catalog=StudData; User ID = sa; Password=abcd1234”;

con.Open();

string studid;

studid = textbox1.Text;

String query = “ SELECT \* from StudData where StudID = @StudID”;

SqlCommand cmd = new SqlCommand( query, con);

cmd.Parameters.Add( new SqlParameter ( “@StudID”, StudID ) );

SqlDataReader dr = cmd.ExecuteReader();

**2. Filtering data using controls in Windows Form**  
  
Once the data is retrieved from the data source by binding the data to the control of the Windows form, the data filter is used for displaying the selective records.   
  
Consider the example of a company where user wants to view the information about the employee data. The information is displayed in the form of a grid view. The connection is established with the database and the data binding through the DataGridView control.  
  
If user wants the data only of the specific employees in the organization, they can be filtered using the following steps.

* 1. Select the DataGridView control in the form and open the DataGridViewTasks window.
  2. Click on the Add Query option. The SearchCriteria Builder dialog box will open.
  3. Add the following query in the query text box.

Code:

SELECT EmpID, EmpName, EmpRole FROM Employee WHERE ( EmpID = 101 )

* 1. Click on the Query Builder button
  2. Click on the Execute Query button. The output generated by the query can be checked
  3. Click OK button and close the Search Criteria Builder dialog box
  4. Press F5 and execute the Windows Form.

In the above example, TableAdapter queries are the SQL statements or procedures that are executed. The FillBy method is useful for executing the TableAdapter queries.   
  
The following code can be added to the Click event of the FillByToolStrip control.

Code:

private void fillByToolStripButton\_Click( object sender, EventArgs e )

{

try

{

this.internalCandidateTableAdapter.FillBy( this.EmpDataSet.Employee);

}

catch( System.Exception ex )

{

System.Windows.Forms.MessageBox.Show( ex.Message );

}

}

In the above code, the FillBy method is used for adding data from the Employee table of the EmpDataSet.

## **XML in .NET**

XML is a software- and hardware-independent tool for storing and transporting data.

* XML stands for eXtensible Markup Language
* XML is a markup language much like HTML
* XML was designed to store and transport data
* XML was designed to be self-descriptive

Although many programming languages and environments have provided XML support as an add-on, .NET’s support is integrated into the framework more tightly than most. The .NET development team decided to use XML extensively within the framework in order to meet its design goals. Accordingly, they built in XML support from the beginning.

The .NET Framework contains five main assemblies that implement the core XML standards. Following table lists the five assemblies, along with a description of the functionality contained in each.

| **Assembly** | **Description** |
| --- | --- |
| **System.Xml** | **Basic XML input and output with XmlReader and XmlWriter many XML utility classes** |
| **System.Xml.Schema** | **Constraint of XML via XML Schema with XmlSchemaObject and its subclasses** |
| **System.Xml.Serialization** | **Serialization to plain XML and SOAP** (Simple Object Access Protocol is a messaging protocol that allows programs that run on disparate operating systems (such as Windows and Linux) to communicate using HTT and **XML**)  **with XmlSerializer** |
| **System.Xml.XPath** | **Navigation of XML via XPath with XPathDocument, XPathExpression, and XPathNavigator** |
| **System.Xml.Xsl** | **Transformation of XML documents via XSLT with XslTransform** |

In addition, the System.Web.Services and System.Data assemblies contain classes that interact with the XML assemblies

**For example, the System.Data assembly handles database operations. Its DataSet class provides a mechanism to transmit database changes using XML.**

There are two approaches to work with XML and ADO.

1. First, you can use ADO.NET to access XML documents.
2. Second, you can use XML and ADO.NET to access XML. Additionally, you can access a relational database using ADO.NET and XML.NET.

**Reading XML using Data Set**

In ADO.NET, you can access the data using the DataSet class. The DataSet class implements methods and properties to work with XML documents. The following sections discuss methods that read XML data.

**The Read xml Method**

ReadXml is an overloaded method; you can use it to read a data stream, TextReader, XmlReader, or an XML file and to store into a DataSet object, which can later be used to display the data in a tabular format. The ReadXml method has eight overloaded forms. It can read a text, string, stream, TextReader, XmlReader, and their combination formats. In the following example, create a new DataSet object.

In the following example, create a new DataSet object and call the DataSet. ReadXml method to load the books.xml file in a DataSet object:

**//Create a DataSet object**

**DataSet ds = new DataSet();**

**// Fill with the data**

**ds.ReadXml("books.xml");**

Once you've a DataSet object, you know how powerful it is. Make sure you provide the correct path of books.xml.

**The ReadXmlSchema method**

The ReadXMLSchema method reads an XML schema in a DataSet object. It has four overloaded forms. You can use a Text Reader, string, stream, and XmlReader. The following example shows how to use a file as direct input and call the ReadXmlSchema method to read the file:

**DataSet ds = new DataSet();**

**ds.ReadSchema(@"c:\books.xml");**

**The following example reads the file XmlReader and uses XmlTextReader as the input of ReadXmlSchema:**

**//Create a dataset object**

**DataSet ds = new DataSet("New DataSet");**

**// Read xsl in an XmlTextReader**

**XmlTextReader myXmlReader = new XmlTextReader(@"c:\books.Xml");**

**// Call Read xml schema**

**ds.ReadXmlSchema(myXmlReader);**

**myXmlReader.Close();**

**Writing XML using Data Set**

Not only reading, the DataSet class contains methods to write XML file from a DataSet object and fill the data to the file.

**The Writexml Method**

The WriteXml method writes the current data (the schema and data) of a DataSet object to an XML file. This is overloaded method. By using this method, you can write data to a file, stream, TextWriter, or XmlWriter. This example creates a DataSet, fills the data for the DataSet, and writes the data to an XML file.

Write xml Method

using System;

using System.IO;

using System.Xml;

using System.Data;

namespace XmlAndDataSetsampB2

{

class XmlAndDataSetSampCls

{

public static void Main()

{

try

{

// Create a DataSet, namespace and Student table

// with Name and Address columns

DataSet ds = new DataSet("DS");

ds.Namespace = "StdNamespace";

DataTable stdTable = new DataTable("Student");

DataColumn col1 = new DataColumn("Name");

DataColumn col2 = new DataColumn("Address");

stdTable.Columns.Add(col1);

stdTable.Columns.Add(col2);

ds.Tables.Add(stdTable);

//Add student Data to the table

DataRow newRow; newRow = stdTable.NewRow();

newRow["Name"] = "Mahesh Chand";

newRow["Address"] = "Meadowlake Dr, Dtown";

stdTable.Rows.Add(newRow);

newRow = stdTable.NewRow();

newRow["Name"] = "Mike Gold";

newRow["Address"] = "NewYork";

stdTable.Rows.Add(newRow);

newRow = stdTable.NewRow();

newRow["Name"] = "Mike Gold";

newRow["Address"] = "New York";

stdTable.Rows.Add(newRow);

ds.AcceptChanges();

// Create a new StreamWriter

// I'll save data in stdData.Xml file

System.IO.StreamWriter myStreamWriter = new

System.IO.StreamWriter(@"c:\stdData.xml");

// Writer data to DataSet which actually creates the file

ds.WriteXml(myStreamWriter);

myStreamWriter.Close();

}

catch (Exception e)

{

Console.WriteLine("Exception: {0}", e.ToString());

}

return;

}

}

}

You wouldn't believe the WriteXml method does for you. If you see the output stdData.xml file, it generates a standard XML file that looks like listing 6-27.

WriteXml method output

<?xml version="1.0" ?>

<DS xmlns="StdNamespace">

<Student>

<Name>Mahesh Chand</Name>

<Address>Meadowlake Dr, Dtown</Address>

</Student>

<Student>

<Name>Mike Gold</Name>

<Address>NewYork</Address>

</Student>

<Student>

<Name>Mike Gold</Name>

<Address>New York</Address>

</Student>

</DS>

The Write xml schema method

This method writes DataSet structure to an XML schema. WriteXmlSchema has four overloaded methods. You can write the data to a stream, text, TextWriter, or Xmlwriter. Listing 6-28 uses XmlWriter for the output.

Listing 6-28. write xml schema sample

using System;

using System.IO;

using System.Xml;

using System.Data;

namespace XmlAndDataSetsampB2

{

class XmlAndDataSetSampCls

{

public static void Main()

{

DataSet ds = new DataSet("DS");

ds.Namespace = "StdNamespace";

DataTable stdTable = new DataTable("Students");

DataColumn col1 = new DataColumn("Name");

DataColumn col2 = new DataColumn("Address");

stdTable.Columns.Add(col1);

stdTable.Columns.Add(col2);

ds.Tables.Add(stdTable);

// Add student Data to the table

DataRow newRow; newRow = stdTable.NewRow();

newRow["Name"] = "Mahesh chand";

newRow["Address"] = "Meadowlake Dr, Dtown";

stdTable.Rows.Add(newRow);

newRow = stdTable.NewRow();

newRow["Name"] = "Mike Gold";

newRow["Address"] = "NewYork";

stdTable.Rows.Add(newRow);

ds.AcceptChanges();

XmlTextWriter writer = new XmlTextWriter(Console.Out);

ds.WriteXmlSchema(writer);

Console.ReadLine();

Console.ReadLine();

return;

}

}

}