Chapter 2: Data Commands & Data Reader

Understanding Data Commands & Data Readers

- Essentially, an ADO.NET data command is simply a SQL command or a reference to a stored procedure that is executed against a Connection object. In addition to retrieving and updating data, the Data Command can be used to execute certain types of queries on the data source that do not return a result set and to execute data definition (DDL) commands that change the structure of the data source.

- When a Data Command does return a result set, a DataReader is used to retrieve the data. The DataReader object returns a read-only, forward-only stream of data from a Data Command. Because only a single row of data is in memory at a time (unlike a DataSet, which, as we'll see which stores the entire result set), a DataReader requires very little overhead. The *Read* method of the DataReader is used to retrieve a row, and the *GetType* methods (where *Type* is a system data type, such as *GetString* to return a data string) return each column within the current row.

- As part of the Data Provider, Data Commands and DataReaders are specific to a data source. Each of the .NET Framework Data Providers implements a Command and a DataReader object: OleDbCommand and OleDbDataReader in the System.Data.OleDb namespace; and SqlCommand and SqlDataReader in the System.Data.SqlClient namespace.

- The Data Command supports four versions of its constructor, as shown in Table 3-1. The New () version sets all the properties to their default values, while the other versions allow you to set properties of the Command object during creation. Whichever version you choose, of course, you can set or change property values after the Command is created.

Table 3-1: Command Constructors

Method	Description
New()	Creates a new, default instance of the Data Command
New(Command)	Creates a new Data Command with the Command Text set to the string specified in Command
New(Command, Connection)	Creates a new Data Command with the Command Text set to the string specified in Command and the Connection property set to the SqlConnection specified in Connection
New(Command, Connection, Transaction)	Creates a new Data Command with the Command Text set to the string specified in Command, the Connection property set to the Connection specified in Connection, and the Transaction property set to the transaction Specified in Transactions.

- ***Using Command Properties***

The properties exposed by the Data Command object are shown in Table 3-2. These properties will only be checked for syntax errors when they are set. Final validation occurs only when the Command is executed by a data Source.

Table 3-2: Command Properties

Property	**Meaning**
CommandText	Property indicate query as a string that command have to execute
Command Type	Indicates how the Command Text property is to be interpreted, defaults to Text
Connection	The Connection object on which the Data Command is to be executed
Parameters	The Parameters Collection
Transaction	The Transaction in which the command will execute
UpdateRowSource	Determines how results are applied to a DataRow when the Command is used by the Update method of a DataAdapter

- ***Command Methods***

Table 3-3: Command Methods

Method	**Description**
Cancel	Cancels execution of a Data Command
CreateParameter	Creates a new parameter
ExecuteNonQuery	Executes a command against the Connection and returns the number of rows affected
ExecuteReader	Sends the CommandText to the Connection and builds a DataReader

ExecuteScalar	Executes the query and returns the first column of the first row of the result set
ExecuteXmlReader	Sends the CommandText to the Connection and builds an XMLReader
Prepare	Creates a prepared (compiled) version of the command on the data source
ResetCommandTimeout	Resets the CommandTimeout property to its default value

- ***Data Reader:-***

The DataReader's properties are shown in Table 3-4. The Item property supports two versions: Item(Name), which takes a string specifying the name of the column as a parameter, and Item(Index), which takes an Int32 as an index into the columns collection. (As with all collections in the .NET Framework, the collection index is zerobased.)

Table 3-4: DataReader Properties

Properties	**Description**
Depth	The depth of nesting for the current row in hierarchical result sets. SQL Server always returns zero.
FieldCount	The number of columns in the current row.
IsClosed	Indicates whether the DataReader is closed.
Item	The value of a column.
RecordsAffected	The number of rows changed, inserted, or deleted.

- ***Data Reader Methods***

Table 3-3: Data Reader Methods

Method	Description
Close	Closes the DataReader
GetType	Gets the value of the specified column as the specified type
GetDataTypeName	Gets the name of the data source type
GetFieldType	Returns the system type of the specified Column
GetName	Gets the name of the specified column
GetOrdinal	Gets the ordinal position of the column Specified
GetSchemaTable	Returns a DataTable that describes the structure of the DataReader
GetValue	Gets the value of the specified column as its native type
GetValues	Gets all the columns in the current row
IsDbNull	Indicates whether the column contains a Nonexistent value
GetResult	Advances the DataReader to the next Result
Read	Advances the DataReader to the next row

- ***Data Reader Application***

The given application illustrates use of DataReader. The Form having a combo box that will bound to student.mdb rollno fields and text box display the name of selected item of combo box. The code is given below:

```
using System;
using System.Data;
using System.Windows.Forms;

namespace ADO_DataReader
{
    public partial class Form1 : Form
    {
        System.Data.OleDb.OleDbConnection cn;
        System.Data.OleDb.OleDbCommand cmd;
        System.Data.OleDb.OleDbDataReader dr;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

          cn = new
          System.Data.OleDb.OleDbConnection("Provider=Micro
          soft.Jet.OLEDB.4.0;Data Source=D:\\Student.mdb");
          cn.Open();
          cmd = new System.Data.OleDb.OleDbCommand("select
          Roll_No from stu", cn);
          dr = cmd.ExecuteReader();
          while (dr.Read())
          {
                comboBox1.Items.Add(dr[0]);
          }
          cn.Close();
        }

        private void comboBox1_SelectedIndexChanged(object
        sender, EventArgs e)
        {
            cn.Open();
            cmd = new
          System.Data.OleDb.OleDbCommand("select
          Roll_No,name from stu where Roll_no=" +
          comboBox1.SelectedItem + "", cn);
            dr = cmd.ExecuteReader();
            dr.Read();
            textBox1.Text = Convert.ToString(dr[1]);
            cn.Close();
         }
    }
}
```

• *Data Manipulation Application:*

Insert/Update/Delete Query that will manipulate data in database file using Command object. This Application having student as database file which is having two fields roll no and name. The code is given below:

```
namespace Query
{
    public partial class Form1 : Form
    {
        System.Data.OleDb.OleDbConnection cn;
        System.Data.OleDb.OleDbCommand cmd;
        System.Data.OleDb.OleDbCommand cmd1;


        public Form1()
        {
          cn = new
          System.Data.OleDb.OleDbConnection("Provider=Micro
          soft.Jet.OLEDB.4.0;Data Source=D:\\Student.mdb");

            InitializeComponent();
        }

        private void BtnInsert_Click(object sender,
        EventArgs e)
        {
            cn.Open();
            cmd1 = new
          System.Data.OleDb.OleDbCommand("select count(*)
          from stu where rollno=" + textBox1.Text + "",
          cn);
            int n=Convert.ToInt32( cmd1.ExecuteScalar());
            if (n > 0)
            {
               MessageBox.Show("u cant insert its already
               in database" + n);
            }
            else
            {
               cmd = new
               System.Data.OleDb.OleDbCommand("insert into
               stu values(" + textBox1.Text + ",'" +
               textBox2.Text + "')", cn);
                cmd.ExecuteNonQuery();
                MessageBox.Show("data has been inerted");
            }
            textBox1.Text = "";
            textBox2.Text = "";
            cn.Close();
        }

        private void BtnUpdate_Click(object sender,
        EventArgs e)
        {
            cn.Open();

            cmd = new
          System.Data.OleDb.OleDbCommand("update stu set
          name='"+ textBox2.Text + "'where rollno=" +
          textBox1.Text + "", cn);
            cmd.ExecuteNonQuery();
            MessageBox.Show("data has been updated");
            textBox1.Text = "";
            textBox2.Text = "";
            cn.Close();

        }

        private void BtnDelete_Click(object sender,
        EventArgs e)
        {

            cn.Open();
            cmd1 = new
          System.Data.OleDb.OleDbCommand("select count(*)
          from stu where rollno=" + textBox1.Text + "",
          cn);
            int n = Convert.ToInt32(cmd1.ExecuteScalar());
            if (n == 0)
            {
               MessageBox.Show("No data found for such
               entry");
            }
            else
            {
               cmd = new
               System.Data.OleDb.OleDbCommand("delete from
               stu where rollno=" + textBox1.Text + "",
               cn);
                cmd.ExecuteNonQuery();
                MessageBox.Show("data has been Deleted");
            }
```

```
            textBox1.Text = "";
            textBox2.Text = "";
            cn.Close();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
    }
}
```