Microsoft® Virtual Labs

Data Access using ADO.NET with C# - Part 1

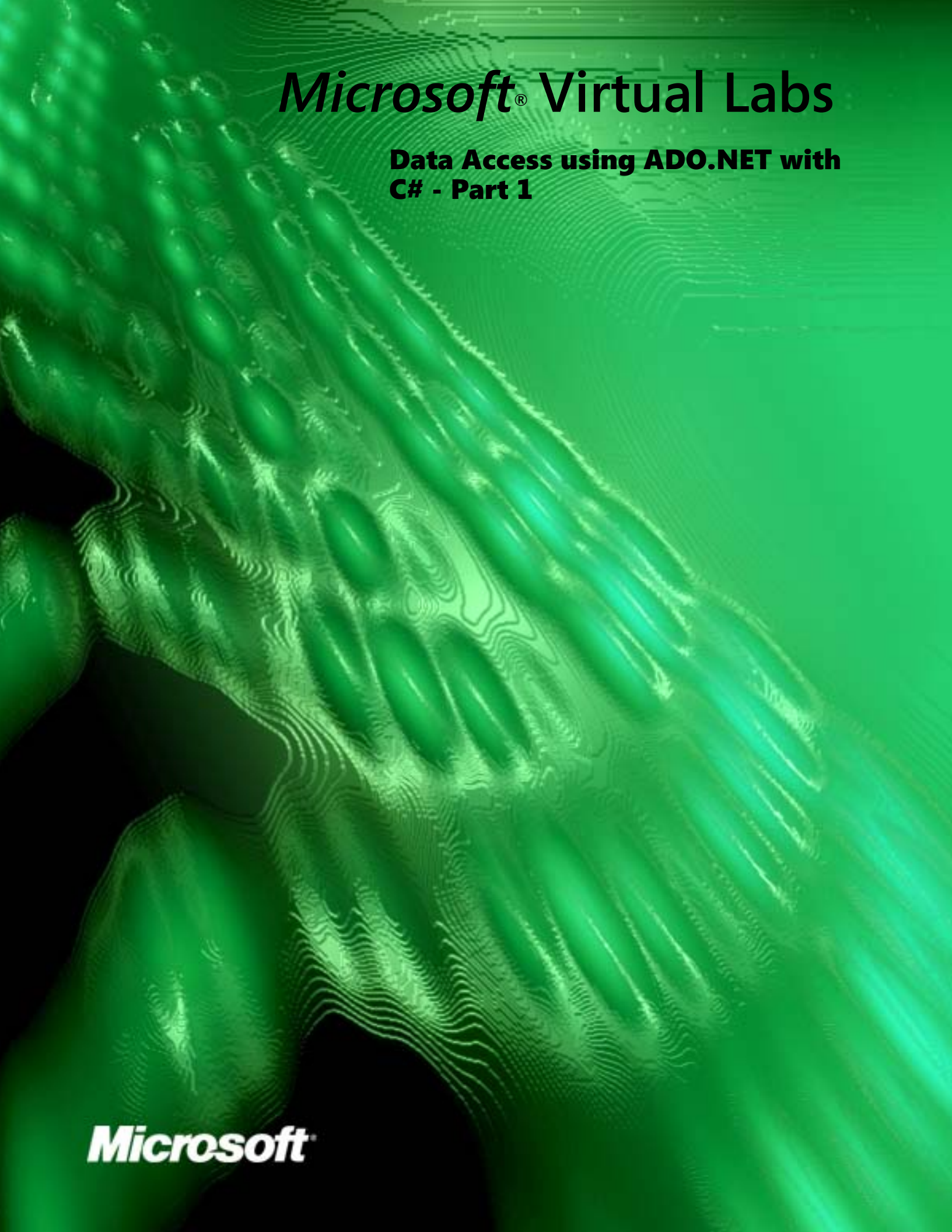

Table of Contents

Data Access using ADO.NET with C# - Part 1

Objectives

After completing this lab, you be able to:

- Work with the SqlCommand and SqlDataReader objects.
- Work with the DataSet and SqlDataAdapter objects.

Scenario

This lab provides you with an overview of ADO.NET. The first exercise provides a brief introduction to ADO.NET and subsequent exercises delve into details on how to perform different database operations such as insert, update and delete of database records, performing database transaction. The lab also discusses programmatically creating DataSet, DataAdapter, DataReader, Command objects and work with the DataGrid control.

This lab is the first part of two. The lab continues in **Data Access using ADO.NET with C# - Part 2**. It is strongly recommended that the labs be done in order.

Estimated time to complete this lab: 35 minutes

Exercise 1
Introduction to ADO.NET

Scenario

ADO.NET provides consistent access to data sources such as Microsoft SQL Server, as well as data sources like OLE DB and XML. Data-driven applications can use ADO.NET to connect to data sources, retrieve and manipulate data.

Overview of ADO.NET Data Providers

Data providers are a core component of the ADO.NET architecture, which enable communication between an application and the data source. A data provider allows you to connect to a data source, retrieve and manipulate data, and update the data source. The data provider in the .NET Framework serves as a bridge between an application and a data source.

The ADO.NET data provider is designed to be lightweight, creating a minimal layer between the data source and your code, increasing performance without sacrificing functionality.

The four core objects that make up an ADO.NET data provider are:

Connection – Establishes a connection to a specific data source

Command – Executes a command against a data source

DataReader – Provides a fast, forward-only, read-only access to data

DataAdapter – Populates a DataSet and resolves updates with the data source

Out-of-the-box, the .NET Framework includes 4 data providers.

1. **SQL Server data provider**

The SQL Server data provider uses the SQL Server-specific data transfer protocol (Tabular Data Stream) to communicate with SQL Server. The data provider is optimized to access SQL Server directly without adding an OLE DB or ODBC Connectivity layer. This provider is recommended for applications using Microsoft SQL Server 7.0 or later.

To use this data provider include the following “using” statement in your code:

```
using System.Data.SqlClient;
```

2. **OLEDB data provider**

The .NET Framework data provider for OLEDB uses native OLEDB through the COM interoperability layer to enable data access. The OLEDB data provider supports both manual and automatic transactions. This provider is recommended for middle-tier applications using Microsoft SQL Server 6.5 or earlier, or any OLE DB compliant provider.

To use this data provider include the following “using” statement in your code:

```
using System.Data.OleDb;
```

3. ODBC data provider (New to Framework 1.1)

The .NET Framework data provider for ODBC uses the native ODBC Driver Manager to enable access to ODBC-data sources. The ODBC data provider supports both local and distributed transactions. This provider is recommended for applications using ODBC data sources.

To use this data provider include the following "using" statement in your code:

```
using System.Data.Odbc;
```

4. Oracle data provider (New to Framework 1.1)

The .NET Framework data provider for Oracle enables data access to Oracle data sources using the Oracle client connectivity software. The data provider supports Oracle client software version 8.1.7 and later. This provider is recommended for applications using Oracle data sources.

Note: *To use this provider, you will need to import this namespace using the Project | Add Reference… menu command and select the* ***System.Data.OracleClient.dll*** *assembly.*

To use this data provider include the following "using" statement in your code:

```
using System.Data.OracleClient;
```

Connection String

The connection string in ADO.NET is similar to an OLEDB connection string. Applications need to specify a connection string for connecting to the underlying data source. The ConnectionString property of the Connection object can be set only when the connection is closed. The connection string is a series of key=value pairs delimited by semi-colons. To connect to a SQL Server running on the local machine, specify "(local)" for the server. The following elements constitute the ConnectionString property for a SqlConnection object:

Data Source-or- server	The name or network address of the instance of SQL Server to which to connect
Integrated Security -or- Trusted_Connection	When false, User ID and Password are specified in the connection. When true, the current windows account credentials are used for authentication
User ID	The SQL Server login account
Password -or- Pwd	The password for the SQL Server account logging on
Initial Catalog -or- Database	The name of the database

Exercise 2
Working with the SqlCommand and SqlDataReader objects

Scenario

In this exercise, you will accomplish common database tasks including connecting to a SQL server database using the SqlConnection class and inserting, updating and deleting data using the SqlCommand class. You will also use the SqlDataReader class to retrieve a read-only, forward-only stream of data from a database. Using the DataReader can increase application performance and reduce system overhead because only one row at a time is ever in memory. Finally you will understand how to perform database transactions.

Tasks	Detailed steps
1. Create a project then create an application configuration file, which will store the connection string.	**a.** Click **Start | All Programs | Microsoft Visual Studio. NET 2003 | Microsoft Visual Studio .NET 2003.** **b.** Select **File | New | Project** menu command. **c.** In the **Project Types** pane select **Visual C# Projects.** **d.** In the **Templates** list select **Windows Application.** **e.** Set the project name to **Lab8-CS**. **f.** Click **OK.** **g.** Select **Project | Add New Item**. **h.** In the **Add New Item** dialog box, select **Application configuration file** from the **Templates** list and click **Open.** **i.** Add the following XML content inside the <**configuration**> tag in **App.config** file (Snippet 1, click **Start | My Computer | C: | Microsoft Hands-on-Labs | Dev-HOL08 | Lab8-CS Snippets**): `<appSettings>` `<add key="ConnectionString"` `value="server=(local);Trusted_Connection=yes;` `database=northwind"/>` `</appSettings>`
2. Create a connection to **SQL Server** and then insert, update and delete data using the **SqlCommand** object.	**a.** Select **View | Solution Explorer**. **b.** Right-click **Form1.cs** in the **Solution Explorer** and select **View Code** **c.** Add the following highlighted code to the using block at the top of the code: `using System.Data.SqlClient;` **d.** Declare a **SqlConnection** variable in the general declaration section of **public class Form1**: `SqlConnection mySqlConnection;` **e.** Select **View | Designer**. **f.** Double-click in the design area of **Form1**. ***Visual Studio .NET 2003*** *will navigate to the source file of* ***Form1*** *and will create an event handler for the form load event (****Form1_Load****)* **g.** Assign a value to the **mySqlConnection** variable in **Form1_Load** event handler (Snippet 2): `mySqlConnection = new SqlConnection (`

```
System.Configuration.ConfigurationSettings.AppSettings.Get("ConnectionString"));
```

*The above line of code, reads the value of the "**ConnectionString**" key from the Application configuration file (**App.Config**)*

h. Select **View | Designer**.

i. Select **View | Toolbox**.

j. Drag a **Button** from the toolbox and drop it on to the form.

k. Right-click the button and select **Properties**.

l. Set the following properties of the button control using the **Properties** window:

- **Name - btnInsert**
- **Text - Insert Command**

m. Expand the width of the button control to fit the text.

n. Drag another **Button** from the toolbox and drop it on to the form.

o. Right-click the button and select the **Properties** menu command.

p. Set the following properties of the button control using the **Properties** window:

- **Name - btnUpdate**
- **Text - Update Command**

q. Expand the width of the button control to fit the text.

r. Drag another **Button** from the toolbox and drop it on to the form.

s. Right-click the button and select the **Properties** menu command.

t. Set the following properties of the button control using the **Properties** window:

- **Name - btnDelete**
- **Text - Delete Command**

u. Expand the width of the button control to fit the text.

v. Double-click the **Insert Command** button control.

***Visual Studio .NET** will navigate to the code view of **Form1 (Form1.cs)** and create an event handler for the **Click** event of the **Insert Command** button*

w. Add the following code (Snippet 3) in the **private void btnInsert_Click** method to insert data about a new customer using **SqlCommand** object:

```
    SqlCommand mySqlCommand = new SqlCommand();
    // Open the connection
mySqlConnection.Open();

//Assign the connection property.
    mySqlCommand.Connection=mySqlConnection;
    try
    {
          // Insert new record for XYZ customer
          mySqlCommand.CommandText = "INSERT INTO
Customers (CustomerId, CompanyName, ContactName,
ContactTitle, Address) Values ('XYZ','XYZ Company',
'John Smith', 'Owner','One My Way')";
```

```
            mySqlCommand.ExecuteNonQuery();
            MessageBox.Show("New Customer data
recorded!");
        }
        catch(Exception ex)
        {
            MessageBox.Show("Error occurred, customer
data could not be recorded: " + ex.Message);
        }
        finally
        {
        // Close the connection if it is open
        if
(mySqlConnection.State==ConnectionState.Open)
        {
            mySqlConnection.Close();
        }
        }
}
```

*The **State** property of the **SqlConnection** object indicates the state of the connection, whether it is open or closed.*

This property can be used to open the connection if it is closed to minimize the server resources consumed by the application

*The **ExecuteNonQuery** method of the **SqlCommand** object is used when the query being executed, **Insert** in this case, is not expected to return any results.*

x. Select **Debug | Start Without Debugging** or press **CTRL+F5**.

y. Click the **Insert Command** button to perform the insert operation.

*A message box appears saying **New Customer data recorded!***

z. Click **OK** to close the message box.

aa. Close **Form1.**

bb. To verify that the data has indeed been recorded successfully:

- Switch to **Visual Studio .NET**.
- Select **View | Server Explorer**.
- Right-click the **Data Connections** node and select **Add Connection** to add a new Data Connection**.**
- In the **Data Link Properties** dialog box, specify the server name as (**local**).
- Select the radio button **Use Windows NT Integrated Security**.
- In the **Select the database on the server** combo-box, select **Northwind**.
- Click **Test Connection**.

*You will see a **Microsoft Data Link** message box saying **Test Connection succeeded**.*

- Click **OK** to close the message box.
- Click **OK** to close the **Data Link Properties** dialog box.
- Select **View | Server Explorer** and navigate to the **Data Connections | CLIENT1.NorthWind.dbo | Tables | Customers** node.
- Double-click the **Customers** table to view all data in the table.

- Scroll to the bottom of the table to verify that a new customer with an ID of '**XYZ**' has been recorded.

cc. Select **View | Solution Explorer**.

dd. Double-click **Form1.cs** in the **Solution Explorer** to view **Form1** in **Design** mode.

ee. Double-click the **Update Command** button control.

*Visual Studio .NET will navigate to the code view of **Form1** (Form1.cs) and will create an event handler for the **Click** event of the **Update Command** button.*

ff. Add the following code (Snippet 4) in the **private void btnUpdate_Click** method to update the inserted data using **SqlCommand** object:

```
SqlCommand mySqlCommand = new SqlCommand();
    // Open the connection
mySqlConnection.Open();

//Assign the connection property.
    mySqlCommand.Connection=mySqlConnection;
    try
    {
          // Update XYZ customer
          mySqlCommand.CommandText = "UPDATE
Customers SET ContactName='Ian Smith' WHERE
CustomerId='XYZ'";
          mySqlCommand.ExecuteNonQuery();
          MessageBox.Show("Customer Data
successfully updated.");
    }
    catch(Exception ex)
    {
          MessageBox.Show("Error occurred, customer
data could not be updated: " + ex.Message);
    }
    finally
    {
    // Close the connection if it is open
    if
(mySqlConnection.State==ConnectionState.Open)
    {
          mySqlConnection.Close();
    }
    }
}
```

gg. Select **Debug | Start Without Debugging** or press **CTRL+F5**.

hh. Click the **Update Command** button to perform all update the inserted row using the **SqlCommand** object.

*A message box appears saying **Customer Data successfully updated**.*

ii. Click **OK** to close the message box.

jj. Close **Form1**.

kk. To verify that the data has indeed been updated successfully:

- Switch to **Visual Studio .NET**.
- Select **View | Server Explorer**.
- Navigate to the **Data Connections | CLIENT1.NorthWind.dbo | Tables | Customers** node.

- Double-click the **Customers** table to view all data in the table.
- Select **Query | Run** to refresh the data display.
- Scroll to the bottom of the table to verify that the **ContactName** of **XYZ** has changed to **Ian Smith**.

ll. Select **View | Solution Explorer**.

mm. Double-click **Form1.cs** in the **Solution Explorer** to view **Form1** in **Design** mode.

nn. Double-click the **Delete Command** button control.

***Visual Studio .NET** will navigate to the code view of **Form1 (Form1.cs)** and will create an event handler for the **Click** event of the **Delete Command** button.*

oo. Add the following code (Snippet 5) in the **private void btnDelete_Click** method to delete the inserted data using **SqlCommand** object:

```
        SqlCommand mySqlCommand = new SqlCommand();
        // Open the connection
mySqlConnection.Open();

//Assign the connection property.
        mySqlCommand.Connection=mySqlConnection;
        try
        {
                // Delete XYZ customer
                mySqlCommand.CommandText = "DELETE FROM
Customers WHERE CustomerId='XYZ'";
                mySqlCommand.ExecuteNonQuery();
                MessageBox.Show("Customer Data successfully
deleted.");
        }
        catch(Exception ex)
        {
                MessageBox.Show("Error occurred, data could not be
deleted: " + ex.Message);
        }
        finally
        {
        // Close the connection if it is open
        if (mySqlConnection.State==ConnectionState.Open)
        {
                mySqlConnection.Close();
        }
        }
}
```

pp. Select **Debug | Start Without Debugging** or press **CTRL+F5.**

qq. Click the **Delete Command** button to perform the Delete operation.

rr. A message box appears saying **Customer Data successfully deleted**.

ss. Click **OK** to close the message box.

tt. Close **Form1**.

uu. To verify that the data has indeed been updated successfully.

- Switch to **Visual Studio .NET**.
- Select **View | Server Explorer**.
- Navigate to the **Data Connections | CLIENT1.NorthWind.dbo |**

	Tables | Customers node. ▪ Double-click the **Customers** table to view all data in the table. ▪ Select **Query | Run** to refresh the data display. ▪ Scroll to the bottom of the table to verify that the record with customer ID '**XYZ**' is not present.
3. Work with the **SqlTransaction** object.	**a.** The following code sample represents the structure of a transacted database operation: `SqlTransaction myTrans;` `myTrans = mySqlConnection.BeginTransaction();` `SqlCommand.Transaction = myTrans;` `try` `{` `...` `...` `...` `myTrans.Commit();` `}` `catch(Exception ex)` `{` `myTrans.Rollback();` `}` **b.** Select **View | Solution Explorer**. **c.** Double-click **Form1.cs** in the **Solution Explorer** to view **Form1** in **Design** mode. **d.** Select **View | Toolbox**. **e.** Drag a **Button** from the toolbox and drop it on to the form. **f.** Right-click the button and select **Properties**. **g.** Set the following properties of the button control using the **Properties** window: ▪ **Name - btnTransaction** ▪ **Text - Transaction** **h.** Expand the width of the button control to fit the text. **i.** Double-click the **Transaction** button control. ***Visual Studio .NET** will navigate to the code view of **Form1 (Form1.cs)** and will create an event handler for the **Click** event of the **Transaction** button.* **j.** Add the following code (Snippet 6) in the **private void btnTransaction_Click** method to perform a transacted insert, update and delete operation using the **SqlCommand** object: `SqlTransaction myTrans;` `SqlCommand mySqlCommand = new SqlCommand();` `// Open the connection` `mySqlConnection.Open();` `//Assign the connection property.` `mySqlCommand.Connection=mySqlConnection;` `myTrans = mySqlConnection.BeginTransaction();` `mySqlCommand.Transaction = myTrans;` `try` `{` `// Insert new record for XYZ customer` `mySqlCommand.CommandText = "INSERT INTO`

```
Customers (CustomerId, CompanyName, ContactName,
ContactTitle, Address) Values ('XYZ','XYZ Company',
'John Smith', 'Owner','One My Way')";
          mySqlCommand.ExecuteNonQuery();

          // Update XYZ customer
          mySqlCommand.CommandText = "UPDATE
Customers SET ContactName='Ian Smith' WHERE
CustomerId='XYZ'";
          mySqlCommand.ExecuteNonQuery();

          // Delete XYZ customer
          mySqlCommand.CommandText = "DELETE FROM
Customers WHERE CustomerId='XYZ'";

          mySqlCommand.ExecuteNonQuery();
          MessageBox.Show("Committing
transaction");

          myTrans.Commit();
          MessageBox.Show("Data successfully
updated.");
    }

catch(Exception ex)
    {
          myTrans.Rollback();
          MessageBox.Show("Error occurred, data has
not been successfully updated: " + ex.Message);
    }

    finally
    {
    // Close the connection if it is open
    if
(mySqlConnection.State==ConnectionState.Open)
    {
          mySqlConnection.Close();
    }
    }
}
```

k. Select **Debug | Start Without Debugging** or press **CTRL+F5**.

l. Click the **Transaction** button to perform all the operations

Either all the operations occur or none of them occur.

m. A message box appears saying **Committing transaction**.

n. Click **OK** to close the message box.

o. A message box appears saying **Data successfully updated**.

p. Click **OK** to close the message box.

q. Close **Form1**

r. To verify that the transaction was successful:

- Switch to **Visual Studio .NET 2003**.
- Select **View | Server Explorer**.
- Navigate to the **Data Connections | CLIENT1.NorthWind.dbo | Tables | Customers** node.
- Double-click the **Customers** table to view all data in the table.

	▪ Select **Query | Run** to refresh the data display. ▪ Scroll to the bottom of the table to verify that the record with customer ID '**XYZ**' is not present.
4. Use the **CommandBehavior.CloseConnection** parameter of **DataReader** so that the connection gets closed automatically when the command is executed.	**a.** Select **View | Solution Explorer**. **b.** Double-click **Form1.cs** in the **Solution Explorer** to view **Form1** in **Design** mode. **c.** Select **View | Toolbox**. **d.** Drag another **Button** from the toolbox and drop it on to the form. **e.** Right-click the button and select **Properties**. **f.** Set the following properties of the button control using the **Properties** window: ▪ **Name - btnDataReader** ▪ **Text - Data Reader** **g.** Expand the width of the button control to fit the text. **h.** Drag a **ComboBox** from the toolbox and drop it on to the form. **i.** Right-click the combo box and select **Properties** **j.** Set the following properties of the combo box control using the **Properties** window: ▪ **DropDownStyle - DropDownList** (select from the drop down list) **k.** Double-click the **Data Reader** button control. ***Visual Studio .NET** will navigate to **Form1** code view and will create an event handler for the **Click** event of the **Data Reader** button* **l.** Open a database connection and read the data in **private void SqlDataReader** using **SqlCommand** object in the **btnDataReader_Click** method as shown below (Snippet 7): `SqlDataReader myReader = null;` `SqlCommand mySqlCommand = new SqlCommand("select ContactName from customers", mySqlConnection);` `try` `{` `mySqlConnection.Open();` `myReader = mySqlCommand.ExecuteReader(CommandBehavior.CloseConnection);` `while (myReader.Read())` `{` `comboBox1.Items.Add(myReader["ContactName"].ToString());` `}` `MessageBox.Show("All names loaded successfully!");` `}` `catch(Exception ex)` `{` `MessageBox.Show("Error occurred: " + ex.Message);` `}` `finally` `{`

```
            if (myReader != null)
                myReader.Close();
        }
}
```

m. To compile and run your application select **Debug | Start Without Debugging** menu command or press **CTRL+F5.**

n. Click the **Data Reader** button to load all customer contact names into the combo box.

o. A message box appears saying **All names loaded successfully!**

p. Click **OK** to close the message box.

The combo Box is populated with the data.

q. Expand the combo box to verify that the contact names are listed.

r. Close **Form1**.

*Note: Since you have used **CommandBehavior.CloseConnection** parameter in **ExecuteReade**r method of **SqlCommand** object, connection state is closed when the **DataReader** is closed, you need not close the connection explicitly.*

Exercise 3
Working with the DataSet object

Scenario

In this exercise, you will populate a DataSet with data using the SqlDataAdapter object to retrieve data into the presentation layer. The ADO.NET DataSet is a memory-resident representation of data that provides a consistent relational programming model independent of the data source. The DataSet object represents a complete set of data including tables, constraints, and relationships among the tables. Because the DataSet is independent of the data source, a DataSet can include data local to the application, as well as data from multiple data sources. Interaction with existing data sources is controlled through the DataAdapter.

In this exercise you will:

- Populate a DataSet with data using the SqlDataAdapter object.
- Persist DataSet to and from Xml.
- Create a DataRelation.

Tasks	Detailed steps
1. Retrieve data using a **SqlDataAdapter** object and **fill a DataSet** object and bind the **DataSet** to a **DataGrid** control.	**a.** Select **View | Solution Explorer**. **b.** Right-click project (**Lab8-CS**) in the **Solution Explorer** and select **Add | Add Windows Form**. *The **Add New Item** dialog box appears.* **c.** Click **Open**. ***Form2.cs** is added to the project and is visible in **Solution Explorer**.* **d.** Right-click **Form2.cs** in the **Solution Explorer** and select **View Code.** **e.** Add the following highlighted code to the using block at the top of the code: `using System.Data;` `using System.Data.SqlClient;` **f.** Declare variables for a **SqlConnection**, **DataSet** and **SqlDataAdapter** in the general declaration section of the **public class Form2**. The **SqlDataAdapter** will manage the updates to the **Customers** table. `SqlConnection mySqlConnection;` `DataSet myDataSet;` `SqlDataAdapter mySqlDataAdapter;` **g.** Select **View | Designer** to view **Form2** in **Design** mode. **h.** Double-click in design area of **Form2**. ***Visual Studio .NET 2003** will navigate to the source file of **Form2** and will create form load method (**Form2_Load**).* **i.** Set the **ConnectionString** property of the declared variable in **private void Form2_Load** event handler by adding the following code. (snippet2) `mySqlConnection = new SqlConnection (`

```
System.Configuration.ConfigurationSettings.AppSettings.Get("ConnectionString"));
}
```

j. Select **View | Designer** to view **Form2** in **Design** mode.

k. Select **View | Toolbox**.

l. Drag a **Button** from the toolbox and drop it on to the **Form2**.

m. Right-click the button and select **Properties**.

n. Set the following properties of the button control using the **Properties** window:

- **Name - btnGet**
- **Text - Get Data**

o. Select **View | Toolbox**.

p. Drag a **DataGrid** control from the toolbox and drop it on to the form.

*A **DataGrid** control is added with the name **dataGrid1.***

q. Expand the **DataGrid** control until it fills the width of the form.

r. Double-click the **Get Data** button control.

s. Add the following highlighted code in the **private void btnGet_Click** method (Snippet 8):

```
        mySqlDataAdapter = new SqlDataAdapter("select * from customers", mySqlConnection);
        try
        {
                myDataSet = new DataSet();
    mySqlDataAdapter.Fill(myDataSet,"Customers");
                dataGrid1.DataSource= myDataSet.Tables["Customers"].DefaultView;
        }
        catch(Exception ex)
        {
                MessageBox.Show("Unable to retrieve Customer data: " + ex.Message);
        }
}
```

t. To load **Form2** during Application startup:

- Select **View | Solution Explorer**.
- Right-click **Form1.cs** in the **Solution Explorer** and select **View Code.**
- In the source file of **Form1**, locate the **Main()** method.
- To load the **Form2** when the application is executed, change the parameter of **Application.Run** method to **Form2()**.

```
static void Main()
{
     Application.Run(new Form2());
}
```

u. Select **Debug | Start Without Debugging** or press **CTRL+F5.**

v. To load data into the **DataGrid** control, click **Get Data** button.

w. Close **Form2**.

2. Create a relationship between **Customers** and **Orders**, name the

a. Select **View | Solution Explorer**.

b. Right-click **Form2.cs** in the **Solution Explorer** and select **View Code.**

relationship **CustOrders**, and present the relational data in a **DataGrid** control.

c. Declare a **SqlDataAdapter** variable in the general declaration section of **Form2** class for managing updates to the **Orders** table. These adapters will be used to managing updates to the back-end **SQL** database tables:

```
SqlDataAdapter mySqlDataAdapter2;
```

d. Add the following highlighted code in the btnGet_**Click** method:

```
private void btnGet_Click(object sender,
System.EventArgs e)
{
mySqlDataAdapter1 = new SqlDataAdapter("select *
from customers", mySqlConnection);
    mySqlDataAdapter2 = new SqlDataAdapter("select
* from orders", mySqlConnection);
    try
    {
          myDataSet = new DataSet();

    mySqlDataAdapter1.Fill(myDataSet,"Customers");

    mySqlDataAdapter2.Fill(myDataSet,"Orders");

// ADD RELATION
    myDataSet.Relations.Add("CustOrders",myDataSet.
Tables["Customers"].Columns["CustomerId"],myDataSet
.Tables["Orders"].Columns["CustomerId"]);

          dataGrid1.DataSource =
myDataSet.Tables["Customers"].DefaultView;
    }
    catch(Exception ex)
    {
          MessageBox.Show("Unable to retrieve
Customer data: " + ex.Message);
    }
}
```

*This code loads data from the **Orders** table also in the **Dataset** and creates a DataRelation between the **Customers** and **Orders** tables.*

e. Select **Debug | Start Without Debugging** or press **CTRL+F5.**

f. Click **Get Data**.

*Observe the relational data in the **DataGrid** control, binding the **DataGrid** to the **Customers** table, automatically retrieves the related data from the **Orders** table using the **DataRelation** between the two tables.*

g. Close **Form2**.

3. Save a **DataSet** as an **XML** document and load **XML** data into the **DataSet**.

a. Select **View | Solution Explorer**.

b. Double-click **Form2.cs** in the **Solution Explorer** to open **Form2** in **Design** mode.

c. Select **View | Toolbox**.

d. Drag a **Button** from the toolbox and drop it on to the Form2.

e. Right-click the button and select **Properties**.

f. Set the following properties of the button control using the **Properties** window:

- **Name - btnSave**
- **Text - Save Xml**

g. Drag a **Button** from the toolbox and drop it on to the Form2.

h. Right-click the button and select **Properties**.

i. Set the following properties of the button control using the **Properties** window:

- **Name - btnLoad**
- **Text - Load Xml**

j. Select **View | Code**.

k. Declare a **String** variable in the general declaration section of **Form2** class. This string will hold the path of the Xml file where the **Dataset** is persisted:

```
public class Form2 : System.Windows.Forms.Form
{
SqlConnection mySqlConnection;
DataSet myDataSet;
SqlDataAdapter mySqlDataAdapter;
SqlDataAdapter mySqlDataAdapter2;
string xmlFilename=@"C:\Customers.xml";
...
...
```

l. Select **View | Designer**.

m. Double-click the **Save Xml** button control.

n. Add the following highlighted code in **private void btnSave_Click** method (Snippet 9).

```
    myDataSet.WriteXml(xmlFilename);
    MessageBox.Show("Customer details stored in: "
+ xmlFilename);
    myDataSet.Clear(); //this will clear the grid
also
}
```

o. Select **View | Designer**.

p. Double-click the **Load Xml** button control.

q. Add the following highlighted code in **private void btnLoad_Click** method (Snippet 10):

```
    MessageBox.Show("Reading data from XML file...
");
    myDataSet.ReadXml(xmlFilename);
dataGrid1.DataSource=
myDataSet.Tables["Customers"].DefaultView;
}
```

r. Select **Debug | Start Without Debugging** or press **CTRL+F5.**

s. Click **Get Data**.

*The application will load **Customer** and **Order** data from the **Northwind** database.*

t. Click **Save Xml** button to write the *Customer-Order* data into the Xml file **C:\Customers.xml**.

u. Click **OK** to close the **Customer details stored in C:\Customers.xml** message box.

v. Click **Load Xml** button to read the **Customer-Order** data from the **Xml** file.

w. Click **OK** to close the **Reading data from XML File** message box.

x. The application will read the data from **C:\Customers.xml** file and

	will fill the data in the **DataGrid** control using the **DataSet** object. **y.** Close **Form2**. **z.** Open **Internet Explorer**, type **C:\Customers.xml** in the **Address** field, and click **Go**. **aa.** Notice how the entire dataset data is persisted in this file. **bb.** Close **Internet Explorer**. **cc.** Return to **Visual Studio .NET 2003**. **dd.** Select **File | Exit** to close **Visual Studio**.