[1. Experiment: Using ADO.NET Classes in View Record Function 2](#_Toc372191528)

[*1.1* *How It Works (Functionality I)* 5](#_Toc372191529)

[*1.2* *How It Works (Functionality II)* 7](#_Toc372191530)

[2. Experiment: Using ADO.NET Classes in Update Record Function 8](#_Toc372191531)

[*2.1* *How It Works* 12](#_Toc372191532)

[3. Experiment: Delete Record Function Using ADO.NET Classes 15](#_Toc372191533)

[*3.1* *How It Works* 18](#_Toc372191534)

[4. SQL to Populate 20](#_Toc372191535)

[5. Additional Notes 21](#_Toc372191536)

**Data Access Topic (Part 3)**

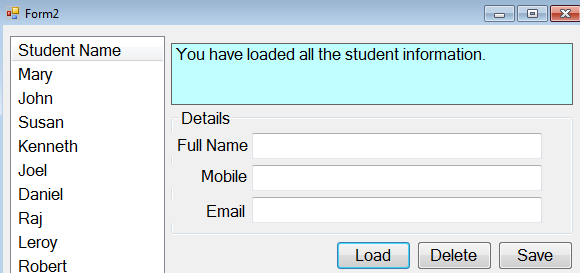
The topic in Data Access Topic (Part 2) allows you to appreciate the DataTable behavior. Recall that the DataTable object's content in previous experiments were not obtained from the database. The DataTable object was manually provided by using code (Listing 1, line 29 to 31 of Data Access Topic Part 2). In this section, you will experiment retrieving records from the database and use these records to populate the DataTable object.

# 1. Experiment: Using ADO.NET Classes in View Record Function

Begin creating Form2 in the same project, ExperimentAdoNetObjects. Note that this project was created earlier in Data Access Topic (Part 1). Provide the interface described in Fig. 1. In this section, you will experiment by providing some code for the Form2 so that it has the following 2 functionalities:

Functionality I

When the user clicks the btnLoad Button control, the code in the btnLoad\_Click event handler will execute to read student records from the database and display the students' name in the ListView control, lvStudent.



btnDelete

btnLoad

lblMessage

Fig. 1

txtFullName

txtMobile

txtEmail

lvStudent

Functionality II

btnSave

When the user clicks a name among the list in the ListView control, lvStudent, the lblMessage Label control will display the record ID of the respective selection.

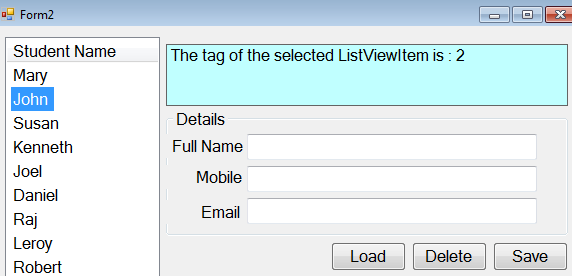


Fig. 2

To achieve the functionality (I), there are several blocks of code needed for the Form2\_Load event handler.

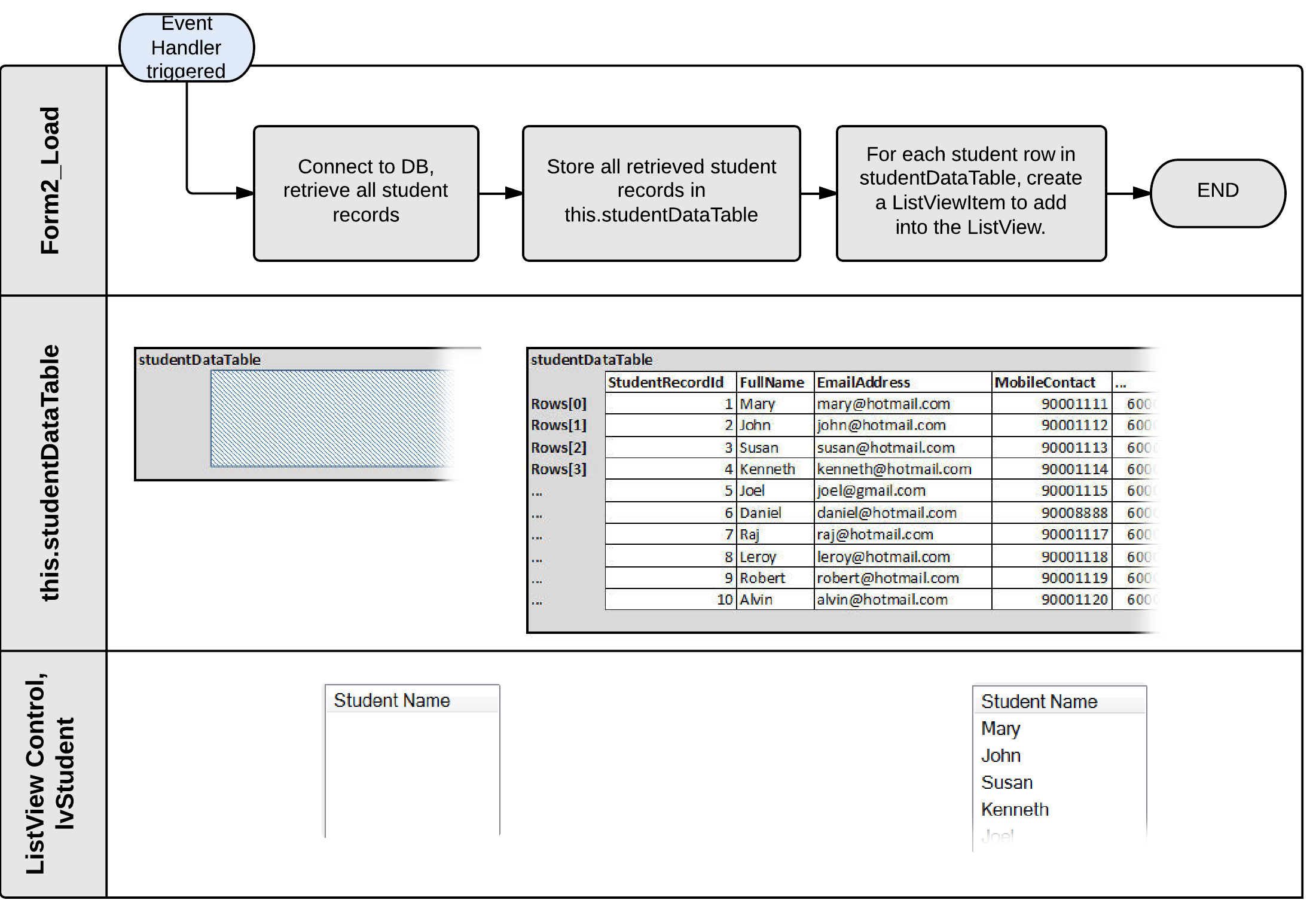


Fig. 3

To achieve the functionality (II), there are several blocks of code needed for the lvStudent\_SelectedIndexChanged event handler.

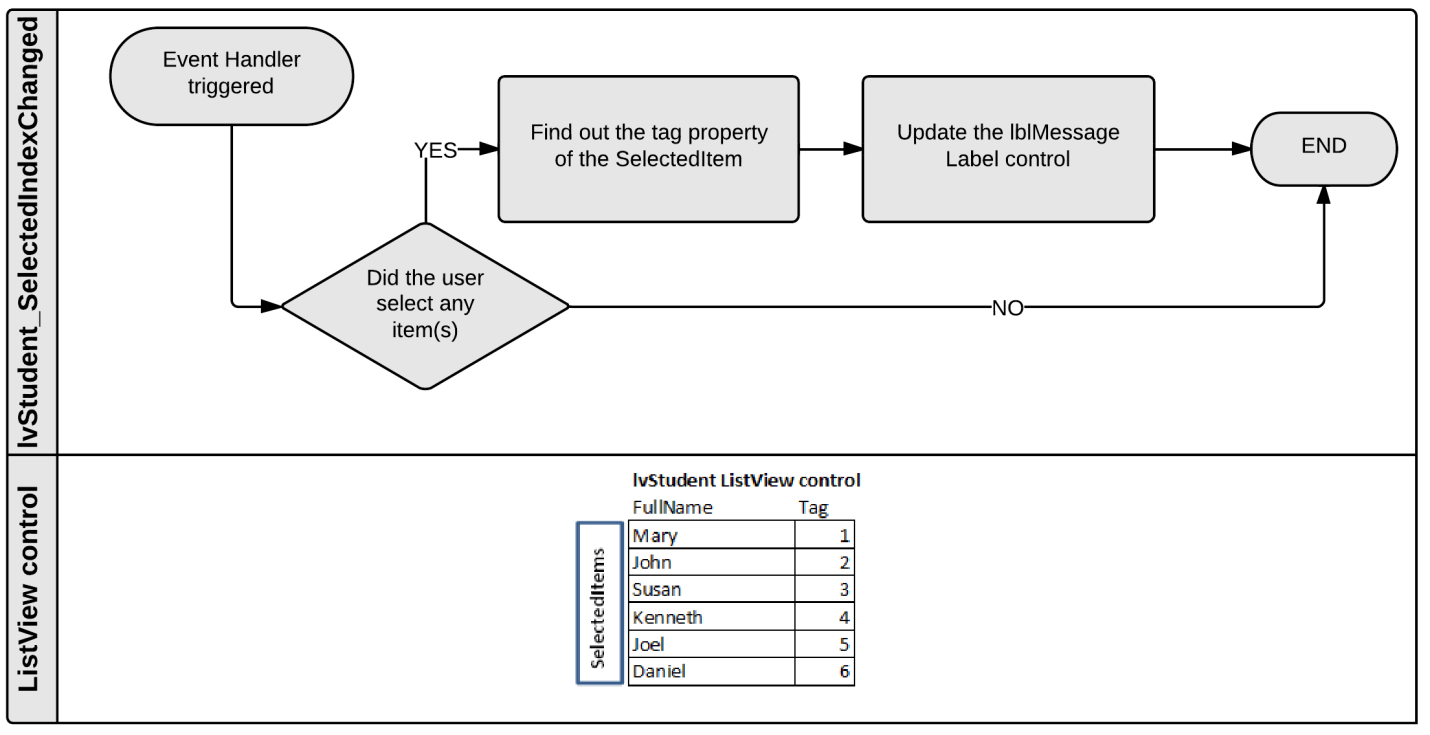


Fig. 4

To allow the 2 sets of code which supports functionality part (I) and functionality part (II) to work together, they need two important private member variables (studentDataTable and currentStudentRecordId) to share information. These variables are:

This is an additional private member to hold ConnectionString info which is frequently needed by many subroutines.

private DataTable studentDataTable = new DataTable();

private string currentStudentRecordId = "";

private const string ConnectionString = "*[Your connection string]*";

The code to support the functionality part (II) is shown in Listing 1.

| Listing 1 The code for btnLoad\_Click and updateListView method to achieve functionality (I). |
| --- |
| 1. private void btnLoad\_Click(object sender, EventArgs e) 2. **{** string userMessage = ""; 3. using (SqlConnection conn = new SqlConnection()) 4. **{** 5. using (SqlCommand cmd = new SqlCommand()) 6. **{** 7. using (SqlDataAdapter da = new SqlDataAdapter()) 8. **{** 9. try 10. **{** 11. conn.ConnectionString = ConnectionString; 12. conn.Open(); 13. cmd.Connection = conn; 14. cmd.CommandText = "SELECT \* FROM Student"; 15. da.SelectCommand = cmd; 16. da.Fill(this.studentDataTable); 17. *//calling the updateListView method to refresh and populate the ListView* 18. *//lvStudent control.* 19. updateListView(); 20. userMessage = "You have loaded all the student information."; 21. **}** 22. catch (Exception ex) 23. **{** 24. userMessage = "Something wrong has occurred."; 25. **}** 26. finally 27. **{** 28. conn.Close(); 29. Console.WriteLine("Closed the connection");*//for debugging purpose only* 30. **}***//end of try..catch..finally* 31. **}***//end of using da* 32. **}***//end of using cmd* 33. **}***//end of using conn* 34. lblMessage.Text = userMessage; *//display the status to the user* 35. **}***//btnLoad\_Click* 36. private void updateListView() 37. **{** lvStudent.Items.Clear(); 38. *//Loop through each student row inside the DataTable, studentDataTable's Rows collection* 39. *//to create ListViewItem objects which can be added into the Items collection of* 40. *//the ListView control, lvStudent.* 41. foreach (DataRow dr in this.studentDataTable.Rows) 42. **{** 43. ListViewItem lvItem = new ListViewItem(dr["FullName"].ToString()); 44. lvItem.Tag = dr["StudentRecordId"].ToString(); 45. lvStudent.Items.Add(lvItem); 46. **}** 47. **}***//updateListView* |

## *1.1 How It Works (Functionality I)*

When the btnLoad\_Click event handler is triggered, the code at line 7, using (SqlDataAdapter da = new SqlDataAdapter()) will create an SqlDataAdapter class object, da. This SqlDataAdapter is an interesting class. You can create a SqlDataAdapter class object to do the following:

* Tell the SqlDataAdapter object, da to use the SqlCommand object, cmd. (line 15)
* Tell the SqlCommand object, cmd to send the SQL command to the database. (line 16)
* When the database server processes the SQL and returns results, the SqlDataAdapter object, da, can write the results into a DataTable, this.studentDataTable. (line 16)

Reference of SqlDataAdapter can be found at: <http://www.dotnetperls.com/sqldataadapter>

Before the command da.Fill(this.studentDataTable); is executed, the studentDataTable (which is a private member variable of the Form2 class) is an empty DataTable instance. After the execution of the command, the SqlDataAdapter object, da, will write all the SQL results into the studentDataTable. As a result, this DataTable object will contain rows of student data and also the field structure such as StudentRecordId, FullName, EmailAddress etc. (Fig. 5)

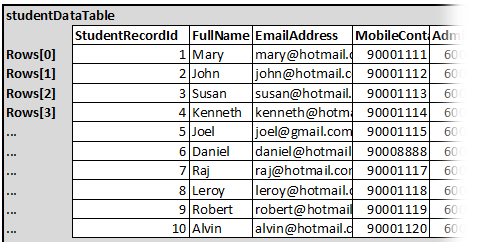


Fig. 5

Each DataRow will store one student data.

**Rows Collection**

The updateListView() subroutine is called at line 19 so that the logic in the method will access the studentDataTable by using the foreach loop technique.

The statement at line 41, foreach (DataRow dr in this.studentDataTable.Rows) will get the DataRow object, dr to represent all the DataRows in the studentDataTable one by one. During each pass of the loop, the ListViewItem object is created and added into the ListView control, lvStudent.

During the 1st pass of the loop, the runtime system will treat the DataRow dr as:



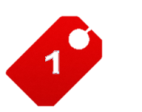
Therefore, when the runtime system processes line 43, ListViewItem lvItem = new ListViewItem(dr["FullName"].ToString()); The runtime system will treat it as ListViewItem lvItem = new ListViewItem("Mary"); This will create a new ListViewItem, lvItem, which has a Text information, Mary. The ListViewItem, lvItem, will be tagged by using the StudentRecordId field information which belongs to the DataRow, dr. This is achieved by having the runtime system execute the code at line 44, lvItem.Tag = dr["StudentRecordId"].ToString(); which is treated as lvItem.Tag = "1";

Fig. 6

Finally, the code at line 45 will insert the ListViewItem, lvItem, into the ListView control, lvStudent. The lvStudent control will have:

During the 1st pass of the loop, the lvStudent control will have one ListViewItem which displays "Mary" and tagged with a value of 1.





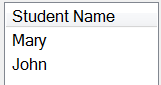
During the 2nd pass of the loop, the runtime system will get the DataRow, dr, to represent the 2nd DataRow inside the studentDataTable object. As a result, the following diagram describes the content and structure of the dr during the 2nd pass of the loop.

Fig. 7



At line 43, ListViewItem lvItem = new ListViewItem(dr["FullName"].ToString()); will be executed as ListViewItem lvItem = new ListViewItem("John"); to create a new ListViewItem, lvItem, which has a Text information, John. The ListViewItem, lvItem will be tagged by using the StudentRecordId field information which belongs to the DataRow, dr. Therefore, lvItem.Tag = dr["StudentRecordId"].ToString(); will be treated as lvItem.Tag = "2";

Finally, the code at line 45 will insert the ListViewItem, lvItem, into the ListView control, lvStudent. The lvStudent control will have:



During the 2nd pass of the loop, the lvStudent control will have one ListViewItem which displays "John" and tagged with a value of 2.



Assume that the SQL result has 10 student records, the DataTable object, studentDataTable will have 10 DataRows. Therefore, the foreach looping will occur 10 times to create 10 ListViewItems inside the lvStudent for you to click and select later.

The tagging helps you to track which ListViewItem was clicked for the next functionality (II) which is discussed later.

The next section discusses the code for functionality (II). The code will be using the SelectedIndexChange event which is supported by the ListView control, to create an lvStudent\_SelectedIndexChange event handler. You will build the necessary logic to display which ListViewItem was selected (by displaying the tag number inside the Label control, lblMessage.

## *1.2 How It Works (Functionality II)*

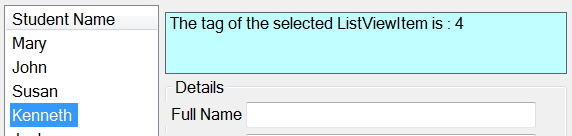
| Listing 2 The code for lvStudent\_SelectedIndexChanged event handler to support functionality (II). |
| --- |
| 1. private void lvStudent\_SelectedIndexChanged(object sender, EventArgs e) 2. **{** 3. //This conditional statement is important to avoid unnecessary indexing error 4. if (lvStudent.SelectedItems.Count > 0) 5. **{** 6. //The code below is used for testing purpose to illustrate how to obtain 7. //the student record id info. 8. this.currentStudentRecordId = lvStudent.SelectedItems[0].Tag.ToString(); 9. lblMessage.Text = "The tag of the selected ListViewItem is : " + this.currentStudentRecordId; 10. **}** 11. **}** |

If you click on Kenneth ListViewItem in the ListView control, the statement at line 8:

currentStudentRecordId = lvStudent.SelectedItems[0].Tag.ToString(); will be treated by the runtime system as currentStudentRecordId = "4";

As a result, the statement at line 9 will be treated by the runtime system as lblMessage.Text = "The tag of the selected ListViewItem is : 4";

Fig. 8



If you click John in the ListView control, the statement at line 8: currentStudentRecordId = lvStudent.SelectedItems[0].Tag.ToString(); will be treated by the runtime system as currentStudentRecordId = "2"; As a result, the statement at line 9 will be treated by the runtime system as lblMessage.Text = "The tag of the selected ListViewItem is : 2"

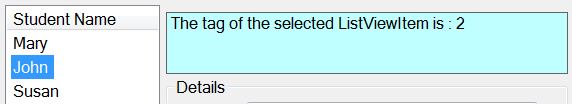


Fig. 9

# 2. Experiment: Using ADO.NET Classes in Update Record Function

You will need to create a Form3 Windows Form in the ExperimentAdoNetObjects project, so that Form2 Windows Form can be used for revision purposes.

Let's revisit the code in the lvStudent\_SelectedIndexChanged event handler in the Form2.cs by referring to the code listing below.

|  |
| --- |
| Listing 3 |
| 1. private void lvStudent\_SelectedIndexChanged(object sender, EventArgs e) 2. **{** 3. *//This conditional statement is important to avoid unnecessary indexing runtime error* 4. if (lvStudent.SelectedItems.Count > 0)**{** 5. *//The code below is used for testing purpose to illustrate how to obtain* 6. *//the student record id info by reading the Tag property of the SelectedItems[0].* 7. this.currentStudentRecordId = lvStudent.SelectedItems[0].Tag.ToString(); 8. lblMessage.Text = "The tag of the selected ListViewItem is : " + this.currentStudentRecordId; *//Display the student record id in the lblMessage* 9. **}**  1. **}** |

Functionality I

The purpose of having the code at line 8 is just to have an assurance that you can retrieve or track what the user has selected in the ListView control, lvStudent. In this section, you are focusing on the update functionality. As such, you need to modify the code in the lvStudent\_SelectedIndexChanged event handler so that the details of the selected student is displayed in the respective Textbox controls.

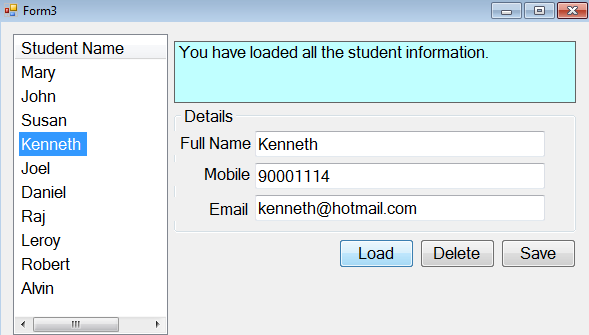


Fig. 10

When you click the ListViewItem which displays Kenneth in the ListView control, lvStudent, your code should find out the record id of this Kenneth, so that it can query the studentDataTable to grab the record details and display them in the Textbox controls.

Functionality II

Refer to Fig. 11. Assume the user changes the name and email address of the Kenneth information in the Windows Form. When the user clicks the btnUpdate Button control, the code in the btnUpdate\_Click event handler will ➀read the new information from the TextBox controls, ➁prepare an Update SQL statement and ➂update the correct student record in the Student table which is residing in the database server. ➃The ListView control, lvStudent will refresh to reflect the changes done.

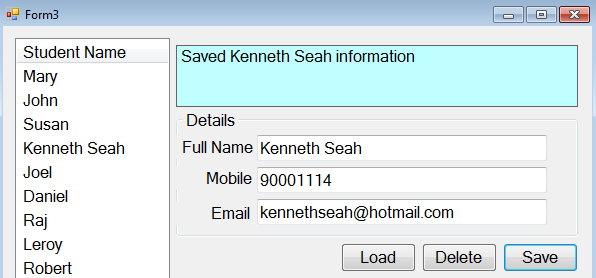
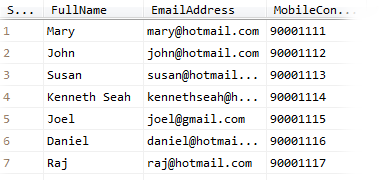


Fig. 11



***Step 1 Modify the lvStudent\_SelectedIndexChanged event handler to display record details***

|  |
| --- |
| Listing 4 lvStudent\_SelectedIndexChanged event handler C# logic |
| 1. private void lvStudent\_SelectedIndexChanged(object sender, EventArgs e) 2. **{** 3. *//This conditional statement is important to avoid unncessary indexing error* 4. if (lvStudent.SelectedItems.Count > 0) 5. **{** 6. *//Query the studentDataTable to find the correct student row.* 7. *//Use the currentRow to represent the student row* 8. this.currentStudentRecordId = lvStudent.SelectedItems[0].Tag.ToString(); 9. DataRow[] currentRow = this.studentDataTable.Select("StudentRecordId=" + currentStudentRecordId); 10. *//Use the currentRow to update the Text property of the respective controls* 11. txtFullName.Text = currentRow[0]["FullName"].ToString(); 12. txtMobile.Text = currentRow[0]["MobileContact"].ToString(); 13. txtEmail.Text = currentRow[0]["EmailAddress"].ToString(); 14. **}**   You will find it useful to place MessageBox.Show(lvStudent.SelectedItems[0].Tag.ToString()) command somewhere between line 7 and line 8 to have a quick feeling how the Tag property contributes to the logic.   1. **}** |

***Step 2 Provide the logic for the btnSave\_Click event handler to update the database server's table***

The code for the btnSave\_Click event handler requires several building blocks illustrated in the basic program flow below.

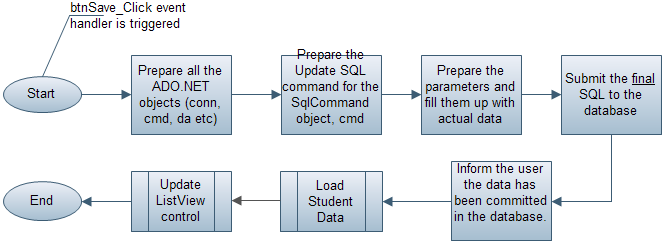


Fig. 12

Refer to the code listing (Listing 5), the C# logic for the btnSave\_Click will retrieve all the information from the Textbox controls and translate these information to a valid UPDATE Sql statement which is finally sent to the database.

| Listing 5 btnSave\_Click event handler C# logic |
| --- |
| 1. private void btnSave\_Click(object sender, EventArgs e) 2. **{** 3. string userMessage = ""; 4. using (SqlConnection conn = new SqlConnection()) 5. **{** 6. using (SqlCommand cmd = new SqlCommand()) 7. **{** 8. using (SqlDataAdapter da = new SqlDataAdapter()) 9. **{** 10. try 11. **{** 12. conn.ConnectionString = ConnectionString; 13. conn.Open(); 14. da.SelectCommand = cmd; 15. cmd.Connection = conn; 16. cmd.CommandText = "UPDATE Student SET FullName=@inFullName," + 17. "MobileContact=@inMobileContact,EmailAddress=@inEmailAddress " + 18. "WHERE StudentRecordId=@inStudentRecordId"; 19. cmd.Parameters.Add("@inFullName",SqlDbType.VarChar,100); 20. cmd.Parameters.Add("@inMobileContact", SqlDbType.VarChar, 8); 21. cmd.Parameters.Add("@inEmailAddress", SqlDbType.VarChar, 50); 22. cmd.Parameters.Add("@inStudentRecordId", SqlDbType.Int); 23. cmd.Parameters["@inFullName"].Value = txtFullName.Text.ToString(); 24. cmd.Parameters["@inMobileContact"].Value = txtMobile.Text.ToString(); 25. cmd.Parameters["@inEmailAddress"].Value = txtEmail.Text.ToString(); 26. cmd.Parameters["@inStudentRecordId"].Value = currentStudentRecordId; 27. cmd.ExecuteNonQuery();  1. userMessage = "Saved " + txtFullName.Text.ToString() + " information"; 2. **}** 3. catch (Exception ex) 4. **{** 5. Console.WriteLine(ex.Message); 6. userMessage = "Something wrong has occurred."; 7. **}** 8. finally 9. **{** 10. conn.Close(); 11. Console.WriteLine("Closed the connection"); 12. **}** 13. **}**//end of using da 14. **}**//end of using cmd 15. **}**//end of using conn 16. lblMessage.Text = userMessage; 17. loadStudentData(); 18. updateListView(); 19. **}**//end of btnSave\_Click 20. //jason chee was here 22. private void loadStudentData() 23. { 24. this.studentDataTable.Clear();//empty the DataTable object, studentDataTable. 25. using (SqlConnection conn = new SqlConnection()) 26. { 27. using (SqlCommand cmd = new SqlCommand()) 28. { 29. using (SqlDataAdapter da = new SqlDataAdapter()) 30. { 31. try 32. { 33. conn.ConnectionString = ConnectionString; 34. conn.Open(); 35. da.SelectCommand = cmd; 36. cmd.Connection = conn; 37. cmd.CommandText = "SELECT \* FROM Student"; 38. da.Fill(this.studentDataTable); 39. } 40. catch (Exception ex) 41. { 42. Console.WriteLine(ex.Message); 43. } 44. finally 45. { 46. conn.Close(); 47. } 48. }//end of using da 49. }//end of using cmd 50. }//end of using conn 51. }//loadStudentData() |

## *2.1 How It Works*

To begin understand the logic, you need to have some assumptions first. Let's examine the Student table, there is a record describing Kenneth. The record's StudentRecordId field value is 4. Imagine that the user has chosen this student record in the Windows Form, and changed the full name information from Kenneth to Kenneth Seah, and email address from kenneth@hotmail.com to kennethseah@hotmail.com (Fig. 11), the code (line 16 to 26) inside the btnSave\_Click event handler has to prepare a valid UPDATE SQL statement to the database server as shown below:

UPDATE Student SET FullName='Kenneth Seah', MobileContact='90001114', EmailAddress='kennethseach@hotmail.com' WHERE StudentRecordId=4;

The command, cmd.ExecuteNonQuery() at line 27 will send the above SQL statement to the database server side to make changes on the actual Kenneth record in the Student table.

The code at line 16 to 26 which prepares of the SQL statement is based on the following concept:

i) Create an SQL template in the SqlCommand object, cmd, and mark those that are going to be dynamic values with a meaningful parameter name (which begins with a prefix @)

Let's have a look at the code at line 16, 17 and 18:

cmd.CommandText = "UPDATE Student SET FullName=@inFullName," +

"MobileContact=@inMobileContact,EmailAddress=@inEmailAddress " +

"WHERE StudentRecordId=@inStudentRecordId";

The code is assigning a string of characters into the CommandText property which belongs to the SqlCommand object, cmd. You can visually recognize from the code, and notice that @inFullName, @inMobileContact, @inEmailAddress and @inStudentRecordId are parameters waiting to be filled by the Textbox control's content. But, the SqlCommand object, cmd does not know these parameters yet until you apply the statements from line 19 to 26.

ii) Inform SqlCommand object, cmd, about the parameters and at the same time describe their respective datatypes.

cmd.Parameters.Add("@inFullName",SqlDbType.VarChar,100);

cmd.Parameters.Add("@inMobileContact", SqlDbType.VarChar, 8);

cmd.Parameters.Add("@inEmailAddress", SqlDbType.VarChar, 50);

cmd.Parameters.Add("@inStudentRecordId", SqlDbType.Int);

The 4 lines of code (line 19 to 22) are totally focused on creating 4 SqlParameter objects. You can find out more about this SqlParameter in this reference: <http://www.dotnetperls.com/sqlparameter>. There are 4 parameters identified in the Update SQL statement in this example. Therefore, you need to create 4 SqlParameter objects. The datatype should match the field design of the Student table.

iii) Copy the input controls' information into the Value property of the parameter objects.

cmd.Parameters["@inFullName"].Value = txtFullName.Text.ToString();

cmd.Parameters["@inMobileContact"].Value = txtMobile.Text.ToString();

cmd.Parameters["@inEmailAddress"].Value = txtEmail.Text.ToString();

cmd.Parameters["@inStudentRecordId"].Value = currentStudentRecordId;

The runtime system will read the above code as:

cmd.Parameters["@inFullName"].Value = "Kenneth Seah";

cmd.Parameters["@inMobileContact"].Value = "90001114";

cmd.Parameters["@inEmailAddress"].Value = "kennethseah@hotmail.com";

cmd.Parameters["@inStudentRecordId"].Value = 4;

The SqlCommand object, cmd, will eventually build the following SQL statement which is ready for submission to the database.

UPDATE Student SET FullName='Kenneth Seah', MobileContact='90001114', EmailAddress='kennethseach@hotmail.com' WHERE StudentRecordId=4;

After the update operation, there are 2 items which are not up-to-date. The DataTable object, studentDataTable still contains the outdated Kenneth DataRow. The ListView control, lvStudent is still showing outdated Kenneth. Therefore, the subroutine, loadStudentData() is called (line 44) to make a query to the database to obtain a fresh set of student records into the DataTable object, studentDataTable. The updateListView() method (line 45) is called to refresh the display of the ListView control, lvStudent based on the studentDataTable's new content.

# 3. Experiment: Delete Record Function Using ADO.NET Classes

The Form3 has a btnDelete Button control. The button helps the user to delete the record he has selected. Fig. 13 below shows that the user has selected Kenneth and viewing the Kenneth record details in the Details Panel. When the user clicks the btnDelete Button control, the code in the btnDelete\_Click event handler will execute to delete the correct student record in the Student table. Fig. 14 shows that the ListView control, lvStudent will recreate its ListViewItems so that it can reflect the record status in the Student table to the user.

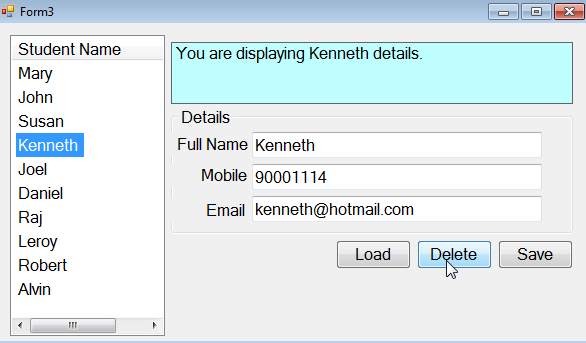


Fig. 13

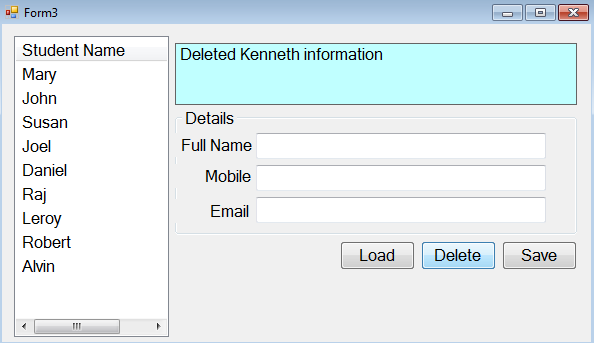
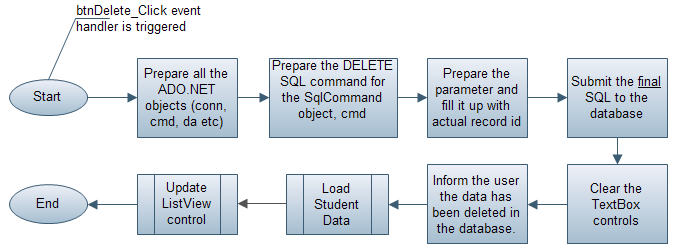


Fig. 14

➀The Textbox controls are cleared. ➁The Label control displays a meaningful message indicating that the record has been deleted. ➂ The ListView control will refresh to reflect the changes done to the Student table in the database.

Refer to Fig. 15, the building blocks for the C# logic in the btnDelete\_Click event handler is very similar to the btnSave\_Click event handler. The logic in the btnSave\_Click event handler uses the SQL Update statement. The logic in the btnDelete\_Click event handler uses the SQL Delete statement.

Fig. 15



To achieve the delete functionality objective, you can use the code provided in the code listing (Listing 6).

| Listing 6 btnDelete\_Click event handler C# logic |
| --- |
| 1. private void btnDelete\_Click(object sender, EventArgs e) 2. **{** 3. string userMessage = ""; 4. using (SqlConnection conn = new SqlConnection()) 5. **{** 6. using (SqlCommand cmd = new SqlCommand()) 7. **{** 8. using (SqlDataAdapter da = new SqlDataAdapter()) 9. **{** 10. try 11. **{** 12. string studentFullName = txtFullName.Text.ToString(); 13. conn.ConnectionString = ConnectionString; 14. conn.Open(); 15. da.SelectCommand = cmd; 16. cmd.Connection = conn; 17. cmd.CommandText = "DELETE Student " + 18. "WHERE StudentRecordId=@inStudentRecordId"; 19. cmd.Parameters.Add("@inStudentRecordId", SqlDbType.Int); 20. cmd.Parameters["@inStudentRecordId"].Value = this.currentStudentRecordId; 21. cmd.ExecuteNonQuery(); 22. *//Clear the Textbox controls' content* 23. txtFullName.Text = ""; 24. txtEma il.Text = ""; 25. txtMobile.Text = ""; 26. *//-------------------------Start----------------------------------* 27. *//This section tries to ensure that the ListView's display* 28. *//is consistent with the database Student table's information* 29. *//----------------------------------------------------------------* 30. *//Call loadStudentData() to update the studentDataTable's content* 31. loadStudentData(); 32. *//Update the ListView control's display* 33. updateListView(); 34. *//----------------------------------------------------------------* 35. *//-------------------------End----------------------------------* 36. *//----------------------------------------------------------------* 37. *//Clear the currentStudentRecordId member variable's content* 38. this.currentStudentRecordId = ""; 39. userMessage = "Deleted " + studentFullName + " information"; 40. **}** 41. catch (Exception ex) 42. **{** 43. Console.WriteLine(ex.Message); 44. userMessage = "Something wrong has occurred."; 45. **}** 46. finally 47. **{** 48. conn.Close(); 49. Console.WriteLine("Closed the connection"); 50. **}** 51. **}**//end of using da 52. **}**//end of using cmd 53. **}**//end of using conn 54. lblMessage.Text = userMessage; 55. **}** |

## *3.1 How It Works*

If you want to delete the Kenneth record from the Student table by using the SQL Server Management Studio's Query Editor Window, you will need the following SQL:

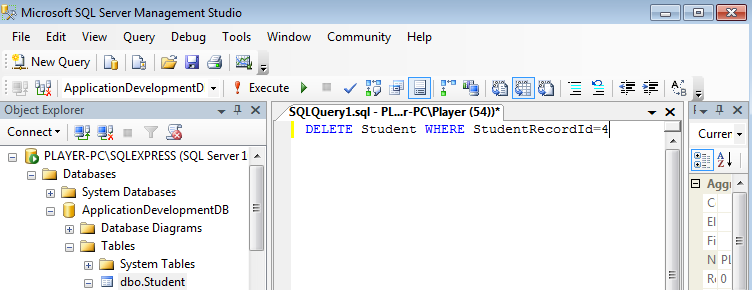


Fig. 16

Deleting the Alvin record from the Student table will require another SQL in the Query Editor window too:

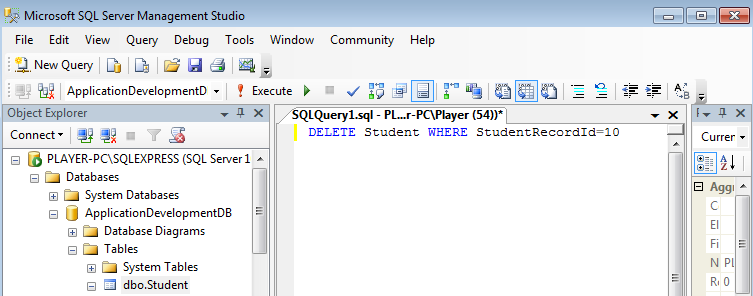


Fig. 17

By analyzing the SQL statement, you will notice that if you need to use an SQL Delete statement in your C# code, you must mark the section in the DELETE Student WHERE StudentRecordId= ; as a dynamic parameter. This is done by preparing an SQL template for the SqlCommand object, cmd, by using the following code (line 17 - 18):

cmd.CommandText = "DELETE Student " +

"WHERE StudentRecordId=@inStudentRecordId";

With the code below (line 19), the SqlCommand object, cmd will know that @inStudentRecordId is a Parameter. The parameter will be waiting to be filled by an actual value at line 20.

cmd.Parameters.Add("@inStudentRecordId", SqlDbType.Int);

The value (record id) which you need to supply to the SqlParameter object can be obtained from the currentStudentRecordId member variable. If the user selects Kenneth, this variable will store a value of 4. If the user selects Alvin, this member variable will store a value of 10. The code at line 20 focus on achieving the objective.

cmd.Parameters["@inStudentRecordId"].Value = this.currentStudentRecordId;

It this user selects Kenneth in the ListView control, this variable should hold the value of 4. As a result, the runtime system will read this line of code as: cmd.Parameters["@inStudentRecordId"].Value = 4;

The command at line 21, cmd.ExecuteNonQuery(); will tell the SqlCommand object, cmd, to begin sending the SQL to the database.

With the combined effort of all the code from line 17 to line 20, assuming the user has selected Kenneth to delete, the SqlCommand object, cmd, will send the following SQL to the database to delete the Kenneth record from the Student table.

DELETE Student WHERE StudentRecordId = 4;

Assuming the user has selected Alvin to delete, the SqlCommand object, cmd, will send the following SQL to the database to delete the Alvin record from the Student table:

DELETE Student WHERE StudentRecordId = 10;

End

# 4. SQL to Populate

INSERT INTO Student VALUES ('Mary', '6000111', 'BLK 309A Ang Mo Kio Street 31 #24-349 Singapore', '562309', 'mary@hotmail.com', '90001111')

INSERT INTO Student VALUES ('John', '6000112', 'BLK 387 Bukit Batok West Avenue 5 #24-378', '650387', 'john@hotmail.com', '90001112')

INSERT INTO Student VALUES ('Susan', '6000113', 'BLK 264 Bukit Batok East Avenue 4 #10-211', '650264', 'susan@hotmail.com', '90001113')

INSERT INTO Student VALUES ('Kenneth', '6000114', 'BLK 398 Kallang #21-01 Singapore', '339098', 'kenneth@hotmail.com', '90001114')

INSERT INTO Student VALUES ('Joel', '6000115', 'BLK 502 Pasir Ris Street 52 #11-227 Singapore', '510502', 'joel@gmail.com', '90001115')

INSERT INTO Student VALUES ('Daniel', '6000116', 'BLK 327 Clementi Avenue 5 #08-197 Singapore', '120327', 'daniel@hotmail.com', '90001116')

INSERT INTO Student VALUES ('Raj', '6000117', 'BLK 52 New Upper Changi Road #02-1490 Singapore', '461052', 'raj@hotmail.com', '90001117')

INSERT INTO Student VALUES ('Leroy', '6000118', '27 Holland Hill #01-03 Holland Peak Singapore', '278741', 'leroy@hotmail.com', '90001118')

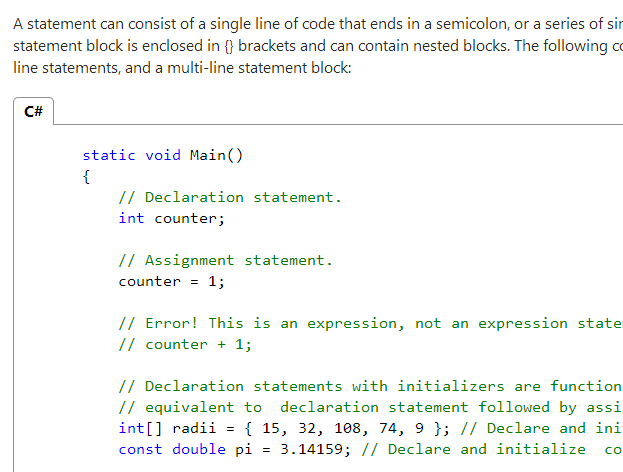
INSERT INTO Student VALUES ('Robert', '6000119', 'BLK 336 Ubi Avenue 1 #04-835 Singapore', '400336', 'robert@hotmail.com', '90001119')

INSERT INTO Student VALUES ('Alvin', '6000120', 'BLK 477 Pasir Ris Drive 6 #05-512 Singapore', '510477', 'alvin@hotmail.com', '90001120')

# 5. Additional Notes

Terminologies are important especially for beginners in C# programming. You must use the right terms when communicating with others during projects and internship. The following reference is very helpful for you to visit and strengthen your programming literacy from time to time.

<http://msdn.microsoft.com/en-us/library/ms173143.aspx>



Many developers are usually confused on the term, "throw exception" and what it means. If you find that you are in the same club with them, this reference might help: <http://social.msdn.microsoft.com/Forums/en-US/57e2c281-eaf4-48fa-8f5e-c13240c1281d/what-does-throw-mean>

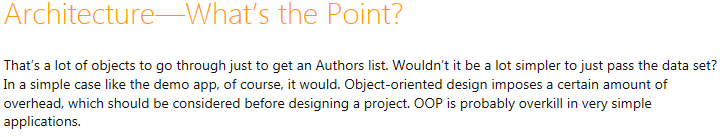
**Programmers Community's View on ADO.NET and Object Oriented Programming**

You should revisit this article "ADO.NET for the Object-Oriented Programmer" from time to time when you level up in programming skills. The article is divided to 2 pages (Part 1 and Part 2). The content is written from the Object Oriented Programmers' point of view. The content in Part 1 focuses in using OO coding and Ado.Net is used for shifting data back and forth only. In Part 2, the author shows the advantages of Ado.Net and gives the reader the feeling that OO is an overkill in some situations. You should read the introduction and conclusion section of each article.

<http://www.codeproject.com/Articles/12669/ADO-NET-for-the-Object-Oriented-Programmer-Part-On>

<http://www.codeproject.com/Articles/12767/ADO-NET-for-the-Object-Oriented-Programmer-Part-Tw>

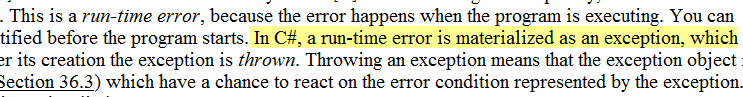
There is also an interesting paragraph as illustrated in the following figure:



It is not wise for a programmer insists on Object Oriented Programming in any development scenarios. It is also not wise for a programmer to insists on using purely ADO.NET in any coding situations. Just like those marketing professionals trying to forecast demand and sales, you just have to keep accumulating experience to make an informed judgement (using the ADO.NET way or the OOP way) in designing a solution for clients based on how frequent the requirement changes and the timeline of the project.

**Programmers Community's View on Runtime Error and Exception**

If you are confused the meaning of an exception which is used through out the internet community, you are not alone. Another reference can be found at http://people.cs.aau.dk/~normark/oop-csharp/html/notes/exceptions\_themes-cs-error-handling-sect.html. Runtime error is something bad that happens when the application is running. An exception is an object that is created by the .NET runtime system to describe that error so that you can use the Catch block to handle the runtime error gracefully.



**Other Miscellaneous Discussion on ListView and the "this" Keyword**

ListView control is rather confusing from programming point of view. To make the column header text visible, the View property should be View.Details

This link <http://msdn.microsoft.com/en-us/library/dk1507sz.aspx> explains the purpose and usage of the **this** keyword. As a result, variables declared inside a Form class scope does not need a naming convention such as \_FullName, mFullName etc.

End