Version *1.0*

*10/11/2019*

**Application Overview**

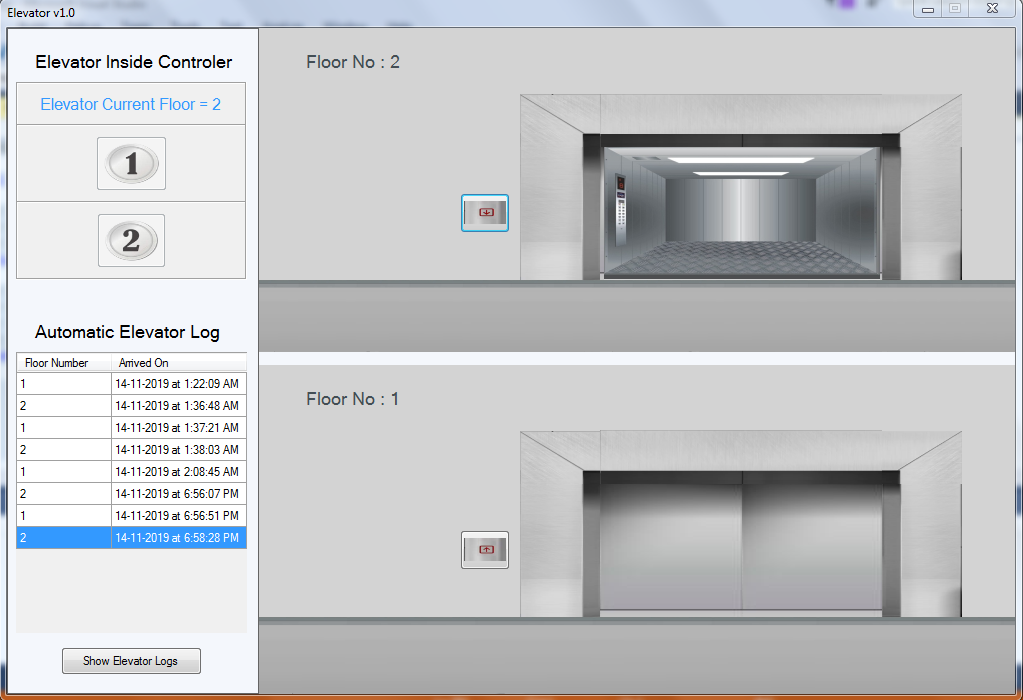
‘Automatic Elevator’ is an application created for specific company with the object oriented approach in c#. This application contains two different floors with the names floor 1 and floor 2 and also contains one elevator which can be controlled by the floor button or from the elevator panel. When elevator arrives on the floor the doors opens and closes automatically, and also saves the current floor state with the updated floor. Inside of the elevator its contain one panel which have 2 buttons for moving the elevator from up floor to the down floor or from down floor to the up floor, also in this panel there is label which changes its text according to the functions by default it show the current state of the elevator. And if the elevator is on floor 1 and call came from floor 2 then without opening the door elevator will come to the floor 2 and then will open door etc after this can be controlled by the calls button or from the inside elevator control panel.

**List of the functionality that were checked**

Below is the list of the tasks which have been implemented in the application with the references of code and screenshots.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task Number** | **Sub-tasks** | **Possible Marks** | **Self-assessment (completed Yes/No)** | **Reference to your testing report** |
| **Task 1** | Complete GUI for Task 1 | 20 | yes | Page 3 index(1) |
| Skeleton of event handlers in place for all buttons | 10 | yes | Page 4 index(2) |
| **Task 2** | All event handlers are functional | 10 | yes | Page 4 Index(3) |
| **Task 3** | Database (DB) is designed and can be connected | 5 | yes | Page 5 Index(4) |
| Log Information can be retrieved from DB and displayed in the GUI | 5 | yes | Page 6 Index(5) |
| When the log button is pressed, log information is sent to and stored in the DB | 5 | yes | Page 7 Index(6) |
| Data source is updated via DataAdapters Update() method instead of ExecuteNonQuery() method | 5 | No, used updating with ExecuteNonQuery() method. | Page 7 Index(6) |
| **Task 4** | Events described in Task 2 animated using delegation and timer | 10 | yes | Page 8 Index(7) |
| **Task 5** | Using relative path instead of absolute path | 5 | yes | Page 9 Index(8) |
| Avoiding any duplication among the event handlers over the database related functions | 5 | yes | Page 12 Index(11) |
| Eliminating logical errors and handling exceptions with try and catch | 5 | yes | Page 10 Index(9) |
| Optimise the efficiency of GUI by implementing multiple tasks concurrently via BackgroundWorker | 5 | no |  |
| Use state patterns instead of if-else statementsto accommodate future changes of the requirement | 10 | Yes | Page 11 Index(10) |
| **Total** |  | 100 |  |  |

1. **Complete GUI for Task 1:**

Below is the screenshot of the ‘Automatic Elevator’ GUI which represents all the panels, buttons, floor, and database tables etc.

1. **Skeleton of event handlers in place for all buttons:**

Below in the code of the GUI buttons, all the code is placed inside of the buttons event handlers with the different classes and object according to the button.

1. // button floor 1
2. **private** **void** elevator\_floor1\_btn\_up\_Click(object sender, EventArgs e)
3. {
4. floor1.Open\_Close\_Gates();
5. elevator.MoveToFloor(floor2);
6. }
7. // button floor 2
8. **private** **void** elevator\_floor2\_btn\_down\_Click(object sender, EventArgs e)
9. {
10. floor2.Open\_Close\_Gates();
11. elevator.MoveToFloor(floor1);
12. }
14. // button for displaying database log in to the dataview grid
15. **private** **void** Display\_db\_Log\_btn\_Click(object sender, EventArgs e)
16. {
17. db.GetTablesData();
18. Display\_db\_Log\_btn.Text = "Reload Elevator Logs";
19. }
20. **All event handlers are functional:**

In this code snippet inside of the buttons there are functions which are used from the different classes to perform the different functionalities.

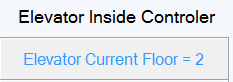
1. // button for displaying database log in to the dataview grid
2. **private** **void** Display\_db\_Log\_btn\_Click(object sender, EventArgs e)
3. {
4. db.GetTablesData();
5. Display\_db\_Log\_btn.Text = "Reload Elevator Logs";
6. }
8. // button 1 of the elevator control panel
9. **private** **void** button\_floor\_1\_Click(object sender, EventArgs e)
10. {
11. **if** (Int32.Parse(db.GetCurrentFloor()) == 2)
12. {
13. move\_elevatorBox\_from\_floor2\_to\_floor1.Start();
14. }
15. **else**
16. {
17. control\_label.Text = "Lift is already on Floor 1";
18. }
19. }
21. // button floor 1
22. **private** **void** elevator\_floor1\_btn\_up\_Click(object sender, EventArgs e)
23. {
24. floor1.Open\_Close\_Gates();
25. elevator.MoveToFloor(floor2);
26. }
28. // button floor 2
29. **private** **void** elevator\_floor2\_btn\_down\_Click(object sender, EventArgs e)
30. {
31. floor2.Open\_Close\_Gates();
32. elevator.MoveToFloor(floor1);
33. }
34. **Database (DB) is designed and can be connected:**

Below snippet from the database class there is a method which connect to the database and bring the current floor from the database, in the debug folder there is the access database file, so with the help of OleDbConnection created new connection with the application path and database name and then created new query in which last record of the table retrieved(basically get the last floor number from database specific table), then with the help of OleDbDataReader  command is executed after executing this sql query got the value from the database and then assign it to the Floor variable which will be returned by this function. If connection is failed then it will show error.

1. // Function to retun current floor of the Elevator from Database
2. internal string GetCurrentFloor() {
3. string Floor = "No Current Floor Records Available";
4. var DBPath = Application.StartupPath + "\\Elevator\_DB\_2000.mdb";
5. OleDbConnection conn = **new** OleDbConnection("provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=" + DBPath);
6. **try**
7. {
8. // Get Current Floor Number From Database
9. string queryString = "SELECT Floor\_number FROM Floors WHERE Floor\_id=(SELECT Floor\_id FROM Current\_Floor WHERE Current\_Floor\_id=(SELECT MAX(Current\_Floor\_id) FROM Current\_Floor))";
11. OleDbCommand command = **new** OleDbCommand(queryString, conn);
12. conn.Open();
13. OleDbDataReader reader = command.ExecuteReader();
15. **while** (reader.Read())
16. {
17. Floor = reader.GetInt32(0).ToString();
18. }
20. reader.Close();
21. conn.Close();
22. }
23. **catch** (Exception ex)
24. {
25. MessageBox.Show("Error!!! Something is wrong with Database"+ ex);
26. }
28. **return** Floor;
29. }

Proof of this code snippet is which executes and get the floor number and displayed to the elevator panel:

Created the object of the class with the name db and then running the function of this object.

1. // set the label with the current floor number
2. control\_label.Text = "Elevator Current Floor = " + db.GetCurrentFloor();

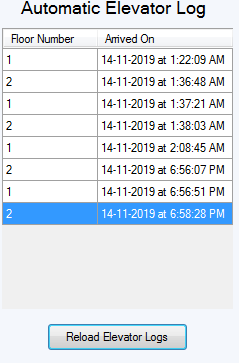
Result:

1. **Log Information can be retrieved from DB and displayed in the GUI:**

Below in the code snippet method of the class, first it will connect to the database and then, added query which will get the all data of the table from the database and then with the help of DataTable and with the DbAdapter got the all data of the table and then in the application I have inserted all the record of the database table to the data grid view component which I have in the GUI of this application.

1. // Function to showing tables of database in the form gridview
2. internal **void** GetTablesData()
3. {
4. var DBPath = Application.StartupPath + "\\Elevator\_DB\_2000.mdb";
5. OleDbConnection conn = **new** OleDbConnection("provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=" + DBPath);
7. **try**
8. {
9. conn.Open();
10. // Get Tables Data from Database
11. string queryString = "SELECT \* FROM Current\_Floor";
12. OleDbDataAdapter DbAdapter = **new** OleDbDataAdapter(queryString, conn);
13. //OleDbCommand command = new OleDbCommand(queryString, conn);
14. DataTable DbTable = **new** DataTable();
15. DbAdapter.Fill(DbTable);
16. // block the un necessary columns from the table
17. Main\_Form.Self.database\_grid\_view.AutoGenerateColumns = **false**;
18. // add the column of the above sql query to out form data view grid table
19. Main\_Form.Self.database\_grid\_view.DataSource = DbTable;
21. conn.Close();
22. }
23. **catch** (Exception ex)
24. {
25. MessageBox.Show("Error!!! Something is wrong Loading data from the database" + ex);
26. }
27. }

Result of the application after calling the above method of the database class:

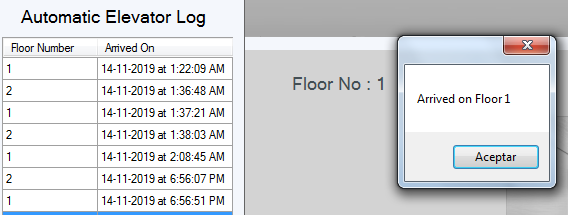


1. **When the log button is pressed, log information is sent to and stored in the DB:**

For storing the data in to the database I have created function in the database class the code snipped of this function is below, in this snippet when this method is called from the object this will take one argument which will be the floor number we want to save into the database, after connecting with the database then built in function is executed which store the current date and timer into the string variable and then it executes the Sql query which will add the current floor number and the data and time in to the database table in to the different columns after running the query it will send the message box saying floor is updated, and this function will run after elevator will arrive to the new floor.

1. // Function to Set the Current Floor of the Elevator in the database
2. **private** **void** SetFloorToDatabase(**int** State)
3. {
4. var DBPath = Application.StartupPath + "\\Elevator\_DB\_2000.mdb";
5. OleDbConnection conn = **new** OleDbConnection("provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=" + DBPath);
7. **try**
8. {
9. // Set Floor State to the Database with current date and time
10. string date\_time = DateTime.Now.ToString("dd-MM-yyyy 'at' h:mm:ss tt");
11. string queryString = "INSERT INTO Current\_Floor(Floor\_id, Date\_Time) SELECT Floors.Floor\_id, '"+ date\_time + "' FROM Floors WHERE Floors.Floor\_number=?";
13. OleDbCommand command = **new** OleDbCommand(queryString, conn);
14. command.Parameters.AddWithValue("@a",State);
16. conn.Open();
17. command.ExecuteNonQuery();
18. //MessageBox.Show("Arrived on Floor "+State);
19. conn.Close();
20. }
21. **catch** (Exception ex)
22. {
23. MessageBox.Show("Error!!! Something is wrong with Database" + ex);
24. }
25. }

Result of adding floor and the date time to the database after executing this function, below after arriving to the floor 1 this function is executed automatically and also log is updated:

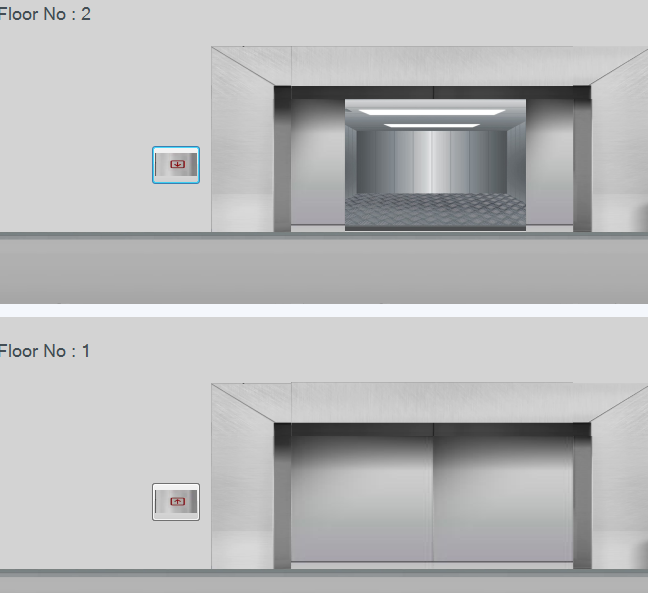


1. **Events described in Task 2 animated using delegation and timer:**

For the moving things/picturebox in the application like open doors, close doors of the floor or for moving the elevator box from floor 1 to floor 2 etc, timers are used for this purpose, for demonstration below are the code snippet of the timers Tick which is for moving elevator from floor 1 to 2 and from floor 2 to 1. Inside the timer tick function after closing the doors of the floor this timer will move the picturebox of the elevator to the up or down according to the timer, when it will reach on the floor in which this picturebox/elevator was moving then there is a condition to confirm that it reaches on the correct floor and on the correct position, then it will stop and then start the other timer immediately which is for opening the door and other functions like button disables or enables etc.

1. // Timer for moving the elevator from floor 2 to floor 1
2. **private** **void** move\_elevatorBox\_from\_floor2\_to\_floor1\_Tick(object sender, EventArgs e)
3. {
4. Buttons\_On\_Off("off");
5. control\_label.Text = "Going to Floor 1";
6. **if** (floor2\_door\_open\_close\_timer.Enabled == **false**)
7. {
8. elevator\_box\_picturebox.Top += 2;
9. **if** (elevator\_box\_picturebox.Location.Y >= floor1.GetFloorYLocation()) {
10. move\_elevatorBox\_from\_floor2\_to\_floor1.Stop();
11. floor1.Open\_Close\_Gates();
12. db.SetCurrentFloor(1);
13. db.GetTablesData();
14. Buttons\_On\_Off("on");
15. Btn\_ColorLabel\_reset();
16. control\_label.Text = "Elevator Current Floor = " + db.GetCurrentFloor();
17. }
18. }
19. }
21. // Timer for moving the elevator from floor 1 to floor 2
22. **private** **void** move\_elevatorBox\_from\_floor1\_to\_floor2\_Tick(object sender, EventArgs e)
23. {
24. Buttons\_On\_Off("off");
25. control\_label.Text = "Going to Floor 2";
26. **if** (floor1\_door\_open\_close\_timer.Enabled == **false**)
27. {
28. elevator\_box\_picturebox.Top -= 2;
29. **if** (elevator\_box\_picturebox.Location.Y <= floor2.GetFloorYLocation()){
30. move\_elevatorBox\_from\_floor1\_to\_floor2.Stop();
31. floor2.Open\_Close\_Gates();
32. db.SetCurrentFloor(2);
33. db.GetTablesData();
34. Buttons\_On\_Off("on");
35. Btn\_ColorLabel\_reset();
36. control\_label.Text = "Elevator Current Floor = "+db.GetCurrentFloor();
37. }
38. }
39. }

Result of this code snippet are below where elevator is moving to the floor 2 using the timer and then i will opens the doors also with the timers.



1. **Using relative path instead of absolute path:**

The below code snippet is used in the database class where it connect with the database folder called [Elevator\_DB\_2000.mdb] which is save into the debug folder, so the path I used to connect 2 different string is relative path which show below I have application.staruppath plus the name of the database file, programme will find the file automatically in the debug folder, I just needed to specify the file name, with this approach program will find automatically it will not create any problem on changing the application folders or location.

1. var DBPath = Application.StartupPath + "\\Elevator\_DB\_2000.mdb";
2. OleDbConnection conn = **new** OleDbConnection("provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=" + DBPath);
3. **try**
4. {
5. // Get Current Floor Number From Database
6. string queryString = "SELECT Floor\_number FROM Floors WHERE Floor\_id=(SELECT Floor\_id FROM Current\_Floor WHERE Current\_Floor\_id=(SELECT MAX(Current\_Floor\_id) FROM Current\_Floor))";
8. OleDbCommand command = **new** OleDbCommand(queryString, conn);
9. conn.Open();
10. **Eliminating logical errors and handling exceptions with try and catch:**

For having less logical errors and for the handling exceptions I have used try and catch in this application, below in the code snippet I have showed some of the error handling exceptions with try and catch .

The below code is from one of the method of application where it will execute on starting of the program and will check the current state of the elevator and then it will adjust the elevator box according to the database result, so I have added try and catch in this method just in case if the database is empty or could not connect then it will add the floor number 1 by default to the database, so the program will keep running smoothly even without having empty database.

1. // function to set the elevatorBox on program run.
2. **private** **void** SetElevator() {
3. **try**
4. {
5. **if** (Int32.Parse(db.GetCurrentFloor()) == 1)
6. {
7. elevator\_box\_picturebox.Location = **new** System.Drawing.Point(0, floor1.GetFloorYLocation());
8. }
9. **else** **if** (Int32.Parse(db.GetCurrentFloor()) == 2)
10. {
11. elevator\_box\_picturebox.Location = **new** System.Drawing.Point(0, floor2.GetFloorYLocation());
12. }
13. }
14. **catch** {
15. // by default add the elevator to floor 0
16. elevator\_box\_picturebox.Location = **new** System.Drawing.Point(0, floor1.GetFloorYLocation());
17. // and add to the db for other functions to work correctly
18. db.SetCurrentFloor(1);
19. }
20. }

Another code snippet is from the database class where it generates error if the connection is unable to connect; in this first program will try first line of the code in side of the try brackets and if any line generates error then it will move to the catch and throw a message showing error.

1. var DBPath = Application.StartupPath + "\\Elevator\_DB\_2000.mdb";
2. OleDbConnection conn = **new** OleDbConnection("provider=Microsoft.Jet.OLEDB.4.0;" + "Data Source=" + DBPath);
4. **try**
5. {
6. // Set Floor State to the Database with current date and time
7. string date\_time = DateTime.Now.ToString("dd-MM-yyyy 'at' h:mm:ss tt");
8. string queryString = "INSERT INTO Current\_Floor(Floor\_id, Date\_Time) SELECT Floors.Floor\_id, '"+ date\_time + "' FROM Floors WHERE Floors.Floor\_number=?";
10. OleDbCommand command = **new** OleDbCommand(queryString, conn);
11. command.Parameters.AddWithValue("@a",State);
13. conn.Open();
14. command.ExecuteNonQuery();
15. //MessageBox.Show("Arrived on Floor "+State);
16. conn.Close();
17. }
18. **catch** (Exception ex)
19. {
20. MessageBox.Show("Error!!! Something is wrong with Database" + ex);
21. }
22. **Use state patterns instead of if-else statements to accommodate future changes of the requirement:**

To prevent a lot of the if else statement i have use some conditions inside of the classes so whenever it creates object and then use method of the class it will return or do specific thing according to the object. Which will minimize the if else conditions in side of the main functions. Below there is a class of the floor in the ‘open or close door’ method there are conditions which will run according to the object value, so this way we can change in the future more efficiently and easily instead of adding statements in side of the main functions:

1. **class** Floors
2. {
4. **int** FloorNumber;
5. **int** FloorStartLocationY;
7. // constuctor for setting floor number and location of the floor
8. **public** Floors(**int** nmbr, **int** location) {
9. FloorNumber = nmbr;
10. FloorStartLocationY = location;
11. }
13. // get the location where the elevator start
14. internal **int** GetFloorYLocation() {
15. **return** FloorStartLocationY;
16. }
17. // get floor number of this floor
18. internal **int** GetFloorNumber()
19. {
20. **return** FloorNumber;
21. }
22. // open and close button for the specific floor
23. internal **void** Open\_Close\_Gates() {
24. **if** (FloorNumber == 1) {
25. // timer of floor 1 for open and close gate.
26. Main\_Form.Self.floor1\_door\_open\_close\_timer.Start();
27. }
28. **else** **if** (FloorNumber == 2)
29. {
30. // start floor 2 close open gate timer.
31. Main\_Form.Self.floor2\_door\_open\_close\_timer.Start();
33. }
34. }
35. **Avoiding any duplication among the event handlers over the database related functions:**

To prevent the duplications of the event I have created functions which will block the click event from the buttons, so I can use them while doing specific thing for an example while elevator is moving to the up floor and i want to stop the buttons to work so below in the code snippet I have added some button in which I want to disable clicks while elevator is moving so in the elevator timer I had added this function with the argument off so they will not be enable while the timer is finish and when timer is finished I used the same function with the ‘on’ argument so this will bring the buttons to their default positions so with this I can prevent a lot of the event also while connecting or loading data from the database:

1. // function which will desable or enable buttons of the form while elevator is moving or doors are closing or opening
2. **private** **void** Buttons\_On\_Off(string btn) {
3. **if** (btn == "on") {
4. elevator\_floor1\_btn\_up.Click += elevator\_floor1\_btn\_up\_Click;
5. elevator\_floor2\_btn\_down.Click += elevator\_floor2\_btn\_down\_Click;
6. }
7. **else** **if** (btn == "off")
8. {
9. elevator\_floor1\_btn\_up.Click -= elevator\_floor1\_btn\_up\_Click;
10. elevator\_floor2\_btn\_down.Click -= elevator\_floor2\_btn\_down\_Click;
11. }
12. }