

# מדריך קוד לניהול מסמכים ו-Log System

## TABLE OF CONTENTS

### GENERAL

PROGRAM.....	2
DB_CLIENT.....	3

### XML HANDLER

REQUEST XML .....	4
RESPONSE XML.....	4
NOTIFICATION XML.....	4
HANDLER INTERFACE.....	5
XML REQUEST HANDLER.....	5

### DOCUMENT FLOW

USER.....	6
DOCUMENT PROCESS EVENT ARGUMENTS .....	6
PIPE CONTROL.....	7
NOTIFICATION CONTROL.....	7

### GENERAL - CONTINUE

FUNCTIONS CONTAINER .....	8
LOGGER .....	8
SERVER .....	9
GLOBAL.....	10
DECLERE EXAMPLE .....	10
HOW TO ADD NEW FUNCTIONS TO THE CODE .....	11
UNSUBSCRIBE EXAMPLE .....	11
RESETDOCPIPE EXAMPLE .....	11
<b>THE COURSE OF THE PROGRAM.....</b>	<b>13</b>
THE STEPS IN THE PROGRAM .....	13
EXPLAIN WHERE EACH STEP OCCURS.....	14

## Program.cs

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            Globals.Init();

            String link = ConfigurationManager.AppSettings.Get("IP");
            int port = int.Parse(ConfigurationManager.AppSettings.Get("Port"));
            Server s = new Server(link, port);
            Globals.Log.WriteLog($"start server in link:{link} and port {port}");
        } catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Globals.Log.WriteLog(e.Message);
        }
        Console.ReadLine();
    }
}
```

מחלקה זו היא המחלקה הראשית של הפרוייקט.

תחילה היא יוצרת את המשתנים הגלובליים (יוסבר בהמשך) בעזרת הפקודה `Globals.Init()`.

`link` הוא משתנה אשר בתוכו יש את כתובת ה-url/ip שהשרת מאזין בו. ו-`port` מכיל את הפורט

שהשרת מאזין לו. משתנים אלה מוגדרים בAppConfig של התוכנית.

לאחר הגדרת ה-`link` וה-`port` ניצור את השרת (יוסבר בהמשך) ונכתוב לconsole את פרטי השרת.

במקרה של כישלון יכתב ל-console מה הייתה השגיאה, ובנוסף נכתוב לקובץ log מה הייתה השגיאה.

ה-`ReadLine` בסוף הוא בשביל לראות מה קורה עם ה-console מבלי שהתוכנית תיסגר.

## DBClient.cs

מחלקה זו מתארת client שיתחבר וישלח פקודה ל-Database שלנו.

```
class DBClient
{
    private string connectionString;
    private SqlConnection cnn;

    public DBClient ()
    {
        //this.connectionString = @"Data Source=WIN-50GP30FG75; Initial Catalog=Demodb; User ID=sa;Pasword=demo123";
        this.connectionString = ConfigurationManager.ConnectionStrings["LogManager"].ConnectionString;
        this.cnn = new SqlConnection(connectionString);
    }

    public SqlConnection GetConnection()
    {
        return this.cnn;
    }
}
```

המשתנה **connectionString**

הוא החיבור לשרת ה-SQLServer

של התוכנית. אנו מקבלים אותו מה-

AppConfig של התוכנית.

המשתנה **cnn** (מסוג

**SqlConnection**) הוא החיבור עצמו ל-Database.

```
public bool DoCommand(SqlCommand command)
{
    try
    {
        this.cnn.Open();
        int res = command.ExecuteNonQuery();
        bool sucsees = !(res < 0); // if (res < 0) it is fail
        command.Dispose();
        this.cnn.Close();
        Console.WriteLine($"Status: {sucsees}, with res: {res}. of the qurry: {command.ToString()}");
        return sucsees;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        this.cnn.Close();
        return false; // fail
    }
}
```

הפונקציה **DoCommand** מבצעת את

פקודת ה-SQL שנשלחה אליה ומחזירה

האם הפעולה הצליחה או לא (אם

הפעולה כשלה אז זה אומר שהערך

המוחזר קטן מ-0). נשלח לפונקציה זו

פקודות SQL אשר לא מצפות לקבל

ערך החזרה (כגון Update, Insert,

Delete).

```
public DataTable GetCommandValues(SqlCommand command)
{
    DataTable dataTable = new DataTable();
    string connString = ConfigurationManager.ConnectionStrings["LogManager"].ConnectionString;
    string query = "select * from programCodes";

    this.cnn.Open();

    // create data adapter
    SqlDataAdapter da = new SqlDataAdapter(command);
    // this will query your database and return the result to your datatable
    da.Fill(dataTable);
    this.cnn.Close();
    da.Dispose();

    return dataTable;
}
```

הפונקציה **GetCommandValues**

מבצעת את פקודת ה-SQL שנשלחה

אליה ומחזירה את הטבלה שנוצרה

לאחר הפעלת שאילתת ה-SQL. נשלח

לפונקציה זו פקודות SQL אשר מצפות

לקבל ערך החזרה (טבלה) כגון

Select.

## xmlHandler

### RequestXML.cs

מחלקה זו ממירה את ה-XML שנשלח לשרת על ידי ה-client למחלקה.

- **ProgramID** זה ב-ID של התוכנית שמבקשת את הבקשה.
  - **SubProgramID** זה ב-ID של התת תוכנית שמבקשת את הבקשה.
  - **Function** זה הפונקציה שצריך להפעיל, שלה צריך להגדיר את ה-ID **Function** ואת המשתנים ששולחים לה (**Params**).
- דוגמאות ל-XML וההגדרה המדויקת שלו נמצאות בקובץ התוכנית.

### ResponseXML.cs

מחלקה זו ממירה את תוצאות הפונקציה ל-XML אשר ישלח בחזרה לשולח ה-XML.

- **Status** האם הפעולה כשלה או לא.
- **ProgramID** זה ב-ID של התוכנית שמבקשת את הבקשה.
- **SubProgramID** זה ב-ID של התת תוכנית שמבקשת את הבקשה.
- **Value** מה ערך ההחזרה, במקרה שאין ערך החזרה יוחזר 0.

### NotificationXML.cs

מחלקה זו יוצרת התראה, שתישלח אל ה-subscribers, בצורת XML.

- **Status** האם הפעולה כשלה או לא.
- **Count** כמה מסמכים יש ב-queue.
- **Process** זה ב-ID של התהליך שהמסמכים ב-queue שלו.
- **Pipe** זה ב-pipe שבו המסמך עובר.

## IHandler.cs

```
interface IHandler
{
    /// <summary>
    /// handlers
    /// </summary>
    /// <returns>State of the handler (falied, success and ect.)</returns>
    string Handle();
}
```

Interface זה מגדיר את פונקציית

Handle שאותה, כל client

שמתחבר לשרת, צריך להפעיל.

## XMLRequestHandler.cs

מחלקה זו ממשת את **IHandler** והיא מטפלת בבקשות XML המוגדרות על ידי

**.RequestXML**

במחלקה יש 2 משתנים: **request** שזה הוא הבקשה שנשלחה, ו-**dbClient** שזה

ה-client ל-DataBase.

```
public string Handle()
{
    List<string> parameters = new List<string>();
    parameters.Add($"{request.ProgramID}");
    parameters.Add($"{request.SubProgramID}");
    parameters.AddRange(request.Func.Params);
    string status = Globals.functions[request.Func.id](parameters, dbClient);
    return status;
}
```

המימוש של **Handle**, ניצור

רשימה של פרמטרים אשר קיבלנו

אותם בבקשה, כך שהראשון הוא

ה-**ProgramID**, השני הוא ה-

**SubProgramID** וכל שאר

המשתנים הם הפרמטרים לפונקציה שביקשו להפעיל. לאחר מכן ניקח את ה-ID של

הפונקציה נלך לטבלת הפונקציות שהוגדרו ב-**Global** (יוסבר בהמשך), בעזרת ה-ID

של הפונקציה נפעיל את הפונקציה הרלוונטית עבור הבקשה ונשלח אליה את ה-

**dbClient** שלנו ביחד עם רשימת הפרמטרים שיצרנו.

## Document Flow

### User.cs

```
public void SendNotification(object sender, EventArgs e)
{
    DocProcessEventArgs args = (DocProcessEventArgs)e;
    string msg = new NotificationXML(1, args.Count, args.Process, args.Pipe).ToXML();

    try
    {
        TcpClient client = new TcpClient(IP, Port);

        // Translate the passed message into ASCII and store it as a Byte array.
        Byte[] data = System.Text.Encoding.ASCII.GetBytes(msg);

        // Get a client stream for reading and writing.
        NetworkStream stream = client.GetStream();

        // Send the message to the connected TcpServer.
        stream.Write(data, 0, data.Length);

        stream.Close();
        client.Close();
    }
    catch (ArgumentNullException ex)
    {
        Console.WriteLine("ArgumentNullException: {0}", ex);
    }
    catch (SocketException ex)
    {
        Console.WriteLine("SocketException: {0}", ex);
    }
}
```

מחלקה זו מתארת משתמש אשר שולח הודעות, נרשם או מוחק את ההרשה מהמערכת. למשתמש יש **IP**, **Port** שזה פרטי השרת אליו צריך לשלוח את ההתראות. ו-**Process** שזה התהליך אליו הוא מקשיב.

לכל משתמש יש פעולה שנקראת **SendNotification** אשר תופעל ברגע שיהיה התראה. והיא תשלח אל ה-XML User של **NotificationXML**. האופרטורים שממומשים במחלקה זה אופרטרי השוואה כדי שלא יהיה כפילויות של Users.

חשוב לשים לב שכאשר משתמשים ב-**SendNotification** בצד השני צריך להיות שרת TCP אשר יקבל את ההתראה כ-XML.

### DocProcessEventArgs.cs

מחלקה זו יורשת מ-EventArgs ומגדירה את הארגומנטים של queue של המסמכים.

- **Count** כמה מסמכים יש ב-queue.
- **Process** זה ה-ID של התהליך שהמסמכים ב-queue שלו.
- **Pipe** זה ה-pipe שבו המסמך עובר.

## PipeControl.cs

מחלקה זו מכילה את המידע לגבי ה-pipes שבהם מסמכים צריכים לעבור, מידע זה נמצא במשתנה `pipesMap` שהוא מסוג `Dictionary` והוא `readonly`. את המידע של סדר ה-pipes היא לוקחת מה-Database וכך עבור כל ID של pipe ניקח את רשימת התהליכים שלו ונמיר אותם ל-`List`.

pipe מוגדר כרצף סיריאלי של תהליכים שכל מסמך צריך לעבור. לצורך הפרויקט ניתן לקרוא לזה SubPipe שכן בעתיד יהיו pipes למסמך ואז את ה-pipe יש לפרק ל-SubPipes סיריאליים כמו שהוגדר מקודם. כך בעצם ניתן לבצע עבודה במקביל על מסמך כלשהו, כך שלא יהיה תלות בין SubPipes שונים.

## NotificationControl.cs

מחלקה זו אחראית על שליחת ההתראות למשתמשים השונים.

מחלקה זו מגדירה `Dictionary` שבו עבור כל תהליך יש `EventHandler` שישלח התראות לכל הרשומים ל-queue של תהליך זה עם ה-`DocProcessEventArgs` בשם `notificationMap`.

את המידע על התהליכים השונים מוציאה המחלקה מה-Database. כמו כן, המחלקה תומכת בהוספת של User כרשום עבור תהליך ספציפי (`AddSubscribe`), במחיקה של User מרשימה (`RemoveSubscribe`), בספירה של כמות המסמכים ב-queue ובהתראה ל-Users השונים (`Notify`). בכל פעם שיש מישהו חדש שנרשם נוסף ל-`EventHandler` של התהליך הזה את הפונקציה `SendNotification` של ה-`User`.

## FunctionsContainer.cs

מחלקה זו אחראית להחזקה של הפונקציות שונות שמוגדרות ב-Database. היא עושה זאת על ידי החזקה של **Dictionary** שמקבל את ה-ID של הפונקציה ומחזיר את הפונקציה שהוגדרה עליו.

## Logger.cs

```
public class Logger
{
    string path;
    Mutex m;

    public Logger(string path)
    {
        m = new Mutex();
        this.path = path;
    }

    public void WriteLog(string msg)
    {
        m.WaitOne();
        DateTime d = DateTime.Now;
        using (StreamWriter sw = File.AppendText(path))
        {
            sw.WriteLine($"[{d.ToString()}] {msg}");
        }
        m.ReleaseMutex();
    }
}
```

מחלקה זו יוצרת קובץ log וכותבת לתוכה logs חשובים לגבי מצב התוכנית בקריסה.

כאשר קוראים לכתיבה של ה-log אז יוצר בשורה חדשה log אשר יהיה מורכב מהתאריך בוא התרחשה הכתיבה ומתוכן ההודעה שנשלחה.

יש לשים לב, שאין בעיה לקרוא לכתיבה ממקומות שונים ומתהליכונים שונים כיוון שיש Mutex שומר מפני דריסה של כתיבות.



## Server.cs

מחלקה זו היא בעצם השרת עצמו איתו אנו עובדים.

היא מקבלת כפרמטר **link** ו-**port** איתם היא פותחת שרת TCP שמאזין ל-**ip** זה-  
port שהיא קיבלה.

```
public void HandleClient(Object obj)
{
    TcpClient client = (TcpClient)obj;
    var stream = client.GetStream();
    string imei = String.Empty;
    Byte[] bytes = new Byte[4096];
    try
    {
        var input = stream.Read(bytes, 0, bytes.Length);
        if (input == 0)
        {
            client.Close();
            return;
        }

        // convert to string
        string str = Encoding.ASCII.GetString(bytes, 0, input);
        Console.WriteLine($"received: {str}");
        // the answer from the handler
        string state = new XMLRequestHandler(str).Handle();
        Console.WriteLine($"return: {state}");

        // replay
        Byte[] reply = System.Text.Encoding.ASCII.GetBytes(state);
        stream.Write(reply, 0, reply.Length);
    }
    catch (Exception e)
    {
        Console.WriteLine("Exception: {0}", e.ToString());
    }
    finally
    {
        client.Close();
    }
}
```

הפונקציה **StartListener** מחקה ש-

client יתחבר, ועבור כל client כזה

היא פותחת Thread שירוך עם הפונקציה

**HandleClient**.

בפונקציה **HandleClient** מחכה לקבלת

הבקשה של ה-client בצורת XML של

**RequestXML** כמחרוזת ואז יוצרת

**XMLRequestHandler** איתו ומפעילה את

פונקציית ה-**Handle** שלו.

לאחר קבלת ה-**ResponseXML** מפונקציית

ה-**Handle** היא תשלח את ה-response

(ה-XML של התוצאה) בחזרה אל ה-

client.

לאחר מכן השרת יסגור את החיבור עם ה-

client ויסגור את ה-Thread.

## Global.cs

מחלקה זו היא מחלקה static אשר מגדירה את כל המשתנים הגלובליים של התוכנית:

- **functions** הפונקציות של התוכנית שהוגדרו ב-Database.
- **notifications** שאחראי על ניהול ההתראות ל-Users והשונים.
- **pipes** שמכיל מידע על ה-pipes של המסמכים.
- **Log** שעושה את הכתיבה של קובץ ה-log.

הפעולה **Init** מאתחלת את המשתנים השונים. ועבור **functions** היא קוראת לפעולה **Init\_functions**.

הפעולה **Init\_functions** יוצרת תכילה **FunctionsContainer** ריק ולאחר מכן מאתחלת את הפונקציות השונות שלה באופן ידני, כן שעבור ID של פונקציה כלשהי ב-Database היא תיצור עבור ה-ID שלה את הפונקציה שלה.

לדוגמה, עבור הפונקציה **Declere** ב-Database שיש לה את ID = 1 נכתוב את המימוש כך:

```
// string Declere(string description)
functions[1] = ((List<string> parameters, DBClient dbClient) =>
{
    if (parameters.Count != 3)
    {
        return new ResponseXML(0, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
    }
    //string qurry = $"INSERT INTO logs.eventTable (event_date, event_type, program_id, sub_program_id, de:
    string qurry = ConfigurationManager.AppSettings.Get("InsertLog");

    SqlCommand command = new SqlCommand(qurry, dbClient.GetConnection());
    command.Parameters.Add("@event_date", SqlDbType.DateTime).Value = DateTime.Now;
    command.Parameters.AddWithValue("@event_type", 1);
    command.Parameters.AddWithValue("@program_id", parameters[0]);
    command.Parameters.AddWithValue("@sub_program_id", parameters[1]);
    command.Parameters.AddWithValue("@description", parameters[2]);

    if (dbClient.DoCommand(command))
    {
        Console.WriteLine(new ResponseXML(1, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML());
        return new ResponseXML(1, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
    }
    Console.WriteLine(new ResponseXML(0, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML());
    return new ResponseXML(0, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
});
```

נשים לב שכל פונקציה מקבלת את הפרמטרים (רשימה של מחרוזות) של הפונקציה ו-  
`dbClient` שיעשה את החיבור ל-Database.

כעת, אם נרצה להוסיף פונקציות חדשות כל מה שצריך לעשות זה לקחת ID שעדיין  
לא קיים עבור הפונקציה ולכתוב לה את המימוש בצורה שנכתבה לעיל (רק לשים לב  
שמוסיפים את הפונקציה ב-Database עם ID תואם).

כמו כן, נשים לב שהפונקציה צריכה להחזיר XML כמחרוזת (בפונקציות שכעת קיימות  
בפרוייקט הם מחזירות `ResponseXML`).

לא כל פונקציה חייבת לעשות שימוש ב-Database (הפונקציה `Declere` כן עשתה  
שימוש ב-`dbClient` כשרצתה להוסיף log לטבלה עם Insert). לדוגמה הפונקציה  
`UnSubscribe` עם `ID = 3`.

```
// string Unsubscribe(string ip, int port, int process)
functions[3] = ((List<string> parameters, DBClient dbClient) =>
{
    if (parameters.Count != 5)
    {
        return new ResponseXML(0, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
    }
    User u = new User(parameters[2], int.Parse(parameters[3]), int.Parse(parameters[0]));
    notification.RemoveSubscribe(u, int.Parse(parameters[4]));

    return new ResponseXML(1, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
});
```

ישנם גם פונקציות שצריכות להודיע כי יש שינוי ב-queue, לדוגמה איתחול ה-`pipe`  
של המסמך, הפונקציה `ResetDocPipe` עם `ID = 6`. פונקציות מסוג זה, בדרך כלל  
יוצאו תחילה מידע על ה-queue או המסמך הרלוונטי (קריאת SQL ראשונה). ולאחר  
מכן יבצעו את השינוי שהם רצו לגבי המסמך או המסמכים הרלוונטיים (קריאת SQL  
שנייה). ובמידת הצורך הם יעשו התראה לכל ה-`Users` שנרשמו ל-queue.

```
// string ResetDocPipe(int doc_id, int sub_pipe)
functions[6] = ((List<string> parameters, DBClient dbClient) =>
{
    if (parameters.Count != 4)
    {
        return new ResponseXML(0, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
    }
    int doc_id = int.Parse(parameters[2]);
    int pipe_id = int.Parse(parameters[3]);

    string query = $"SELECT * FROM flowTable WHERE doc_id = {doc_id} AND pipe_id = {pipe_id}";

    SqlCommand command = new SqlCommand(query, dbClient.GetConnection());

    try
    {
        DataTable dataTable = dbClient.GetCommandValues(command);

        if (dataTable.Rows.Count <= 0)
        {
            // not found
            return new ResponseXML(0, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
        }

        var row = dataTable.Rows[0];

        DataColumn available = dataTable.Columns[5];
        // if available
        if ((int)row[available] > 0)
        {
            // reset the pipe
            query = $"UPDATE flowTable SET available = 0, start_time = GETDATE(), pipe_index = 0, cprocess = {pipes[pipe_id][0]}, round = {(int)row[dataTable.Columns[6]] + 1} WHERE doc_id = {doc_id} AND pipe_id = {pipe_id}";

            SqlCommand update_command = new SqlCommand(query, dbClient.GetConnection());
            if (dbClient.DoCommand(update_command))
            {
                // remove from the current documents process count
                notification.RemoveProcess((int)row[dataTable.Columns[2]]);
                // notify of the current process in pipe
                notification.Notify(new DocProcessEventArgs(notification.ProcessQueueSize((int)row[dataTable.Columns[2]], (int)row[dataTable.Columns[2]], pipe_id));

                // add to the first documents process count of the pipe
                notification.AddProcess(pipes[pipe_id][0]);
                // notify of the first process in pipe
                notification.Notify(new DocProcessEventArgs(notification.ProcessQueueSize(pipes[pipe_id][0], pipes[pipe_id][0], pipe_id));
                return new ResponseXML(1, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
            }
        }
        return new ResponseXML(0, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        dbClient.GetConnection().Close();
        return new ResponseXML(0, int.Parse(parameters[0]), int.Parse(parameters[1])).ToXML();
    }
});
```

בדוגמה הנ"ל:

- בכחול - שימוש בפקודות ה-SQL עם קבלת ערך (הטבלה).
- בכתום – שימוש בפקודות ה-SQL בלי קבלת ערך (רק הצליח או נכשל).
- בצהוב – דוגמה לשימוש בהתראות שם שולחים התראה לגבי Pipe שהשתנה.

## מהלך התוכנית

1. תחילה מאתלים את המשתנים ב-Global.
2. פותחים את השרת.
3. עבור כל client פותחים Thread בו אנו מחכים לבקשה של ה-client (נשלח כמחרוזת של XML).
4. מבצעים את הפונקציה שה-client שלח בבקשה עם הפרמטרים שהוא נתן (קריאה לפונקציה Handle).
5. הפונקציה Handle מבצעת את הקריאה לפונקציה המתאימה ב-Global ומפעילה אותה.
6. בהינתן שצריך שצריך להשתמש בבקשת SQL נבצע זאת (בין אם זה בקשה אחת או כמה, ובין אם זה עם ערך החזרה או לא).
7. בהינתן וצריך להתריע על שינויים ב-queue של המסמכים נעשה זאת לכל המשתמשים הרשומים לתהליכים שעודכנו (נעשה בעזרת Notify).
8. נחזיר מהפונקציה XML כמחרוזת (ResponseXML).
9. הפונקציה Handle מחזירה את המחרוזת שהפונקציה החזירה.
10. השרת יחזיר ל-client את ה-XML שהוחזר מהפונקציה.
11. השרת יסגור את החיבור על ה-client.

- הקריאה לאתחול ב-1 מתבצע במחלקה Program, והאתחול עצמו נעשה במחלקה Global.
- פתיחת השרת ב-2 מתבצעת במחלקה Program, והתחלת הרצת השרת במחלקה Server.
- פתיחת ה-Threads, קליטת ה-client ב-3 מתבצעות במחלקה Server.
- פעולה 4 מתבצעת במחלקה Server בטיפול ב-client.
- פעולה 5 מתבצעת במחלקה XMLRequastHandler.
- פעולות 6,7,8 מתבצעות בפונקציה שה-client ביקש, המימוש של פונקציות אלה נמצא במחלקה Global.
- פעולה 9 מתבצעת במחלקה XMLRequastHandler.
- פעולות 10,11 מתבצעות במחלקה Server בטיפול ב-client.