9-2019

EXTRACT TRANSFORM AND LOADING TOOL FOR EMAIL

Amit Rajiv Lawanghare
California State University – San Bernardino, amit.lawanghare@gmail.com

EXTRACT TRANSFORM AND LOADING TOOL FOR EMAIL

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Amit Lawanghare

September 2019

EXTRACT TRANSFORM AND LOADING TOOL FOR EMAIL

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Amit Lawanghare

September 2019

Approved by:

Dr. Owen Murphy, Advisor, Computer Science and Engineering

Dr. Kristen Voigt, Committee Member

Dr. Yunfei Hou, Committee Member

ABSTRACT

This project focuses on applying Extract, Transform and Load (ETL) operations on the relational data exchanged via emails. An Email is an important form of communication by both personal and corporate means as it enables reliable and quick exchange. Many useful files are shared as a form of attachments which contains transactional/ relational data. This tool allows user to write the filter conditions and lookup conditions on attachments; define the attribute map for attachments to database table. The Data Cleansing for each attribute can be performed writing rules and their matching state. A user can add custom functions for the data transformation. The aggregation of the data is done in form of reports after the operation of data loading into the database is complete. The tool needs one-time setup per file template and its automated from that point.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Owen Murphy for his continuous support and strong encouragement throughout this project. He steered me in the right the direction whenever he thought I needed it. I am also grateful to the members of my committee Dr. Kristen Voigt and Dr. Yunfei Hou for their support.

Finally, I would like to thank my family: my parents and my siblings for supporting me spiritually throughout this project and my life in general.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE: INTRODUCTION

Background

There exists a manual data entry extraction in accounts payable and invoice processing, contracts based in forms, purchase orders, transactional data in financial organizations/small business, personal credit/debit card statements, etc. The manual workflow of corrections and copying data can be replaced with automation. Automated Recurring correction and extraction of important/secure data would improve time, cost and performance.

Purpose

Email is the most preferred and reliable way of sharing important data in most of the organizations. On numerous occasions, a user wants to make use of the full or filtered data periodically. The purpose of this tool is to automate the time consuming and repetitive tasks such as manual transaction entries in which relational data is retrieved from emails.

The need for data segregation is important, to achieve that users can create multiple accounts and add templates under one account. Data filters are defined on each template. To manipulate the relational data, rules can be defined. Multiple rules and conditions can be applied at the template level. Rules get triggered when a specific condition is satisfied. Finally, the aggregation of this data can be performed. For each aggregated dataset custom reports can be generated or loaded in custom files e.g.xml, txt, etc. or in a remote database.

CHAPTER TWO: PROJECT OVERVIEW

Proposed System

This tool has a background job which polls an Email box using Exchange Web Server API [5] (EWS API) to check new emails. It includes a Graphical User Interface (GUI) for setting up new accounts and templates. This interface shows all activity of extraction and transformation of the data as well as failures/errors occurred in the process. A User can generate datasets or reports from loaded data using the GUI.

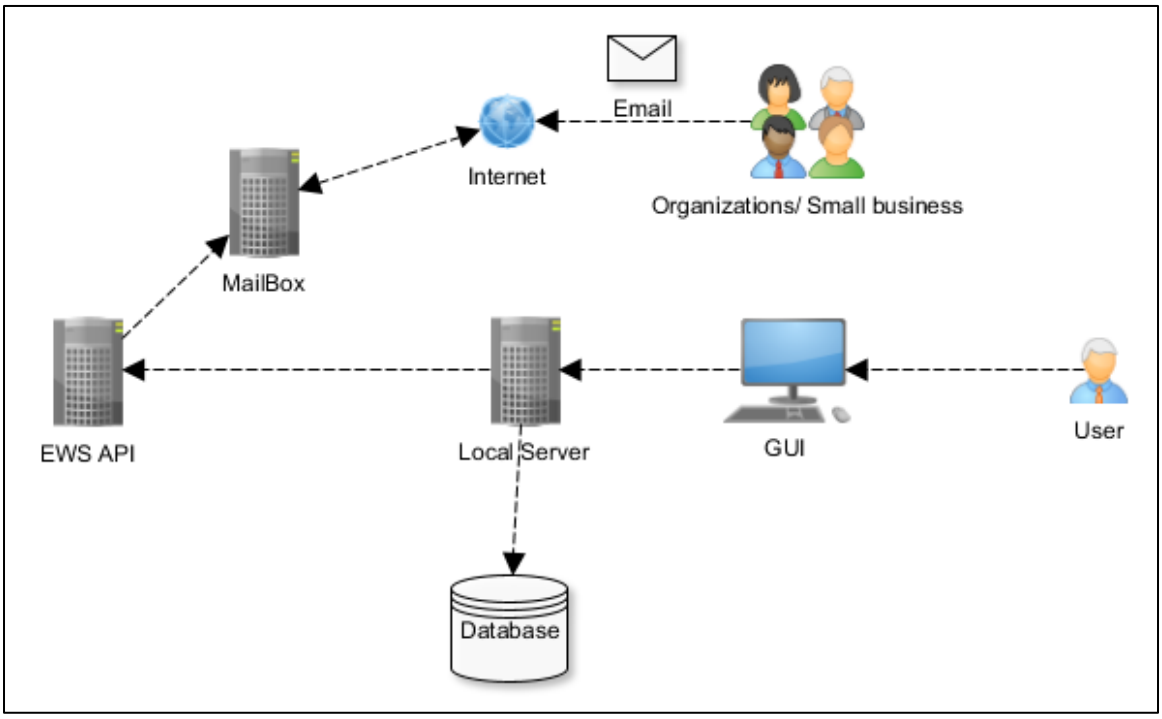

Figure 1. High Level Architecture

Technologies

Exchange Web Server Managed API

In this tool Exchange Web Server managed API provides an object-oriented interface to connect to Exchange Server mailboxes. As stated by David (2014) “Exchange Server provides the underlying infrastructure necessary to run a messaging system. Exchange Server provides the database to store email data, the transport infrastructure to move the email data from one place to another, and the access points to access email data via several different clients” (p. 5) [4].

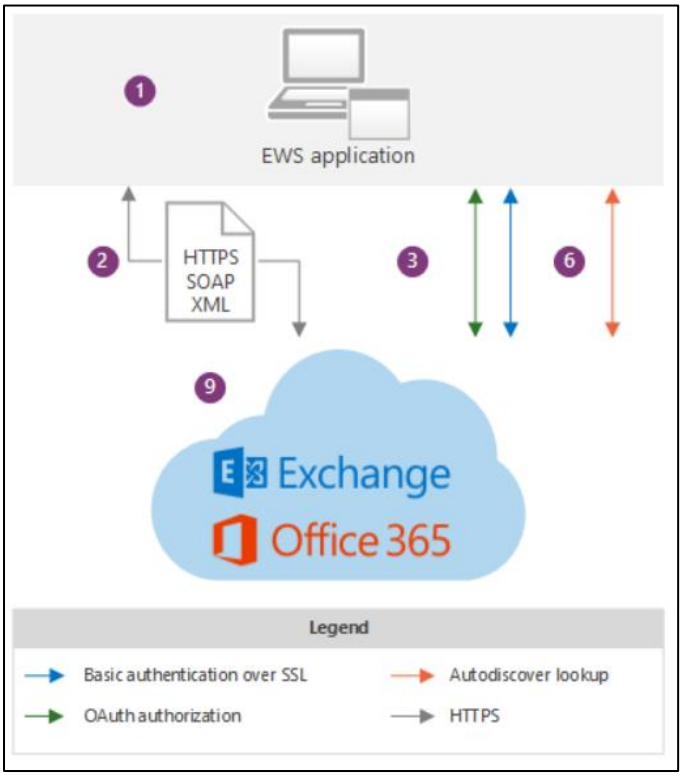

Figure 2. Exchange Web Service App and Exchange Online Architecture

Figure 2 obtained from Caputo Linda's article [2] shows the basic architecture of an Exchange Web Service App connected to an Exchange server which uses Simple Object Access Protocol (SOAP) protocol.

AngularJS

AngularJS [1] is a JavaScript framework to build dynamic web applications which enable to extend HTML syntax. AngularJS mainly provides dependency injections and data binding which can reduce lines of code. It provides features of routing, custom controllers, services and expressions abstracted to a high-level making HTML Document Object Model manipulation easy.

Web API

The term API stands for an Application programming interface. It is a framework under .Net Framework to build Representational State Transfer services over HTTP. Figure 3 obtained from book Pro ASP.NET Web API Security: Securing ASP.NET Web API [4] shows typical deployment of an ASP.net Web API application. Web API provides great flexibility to build services based on HTTP and the developer has easy control over headers and contents sent and requested via HTTP.

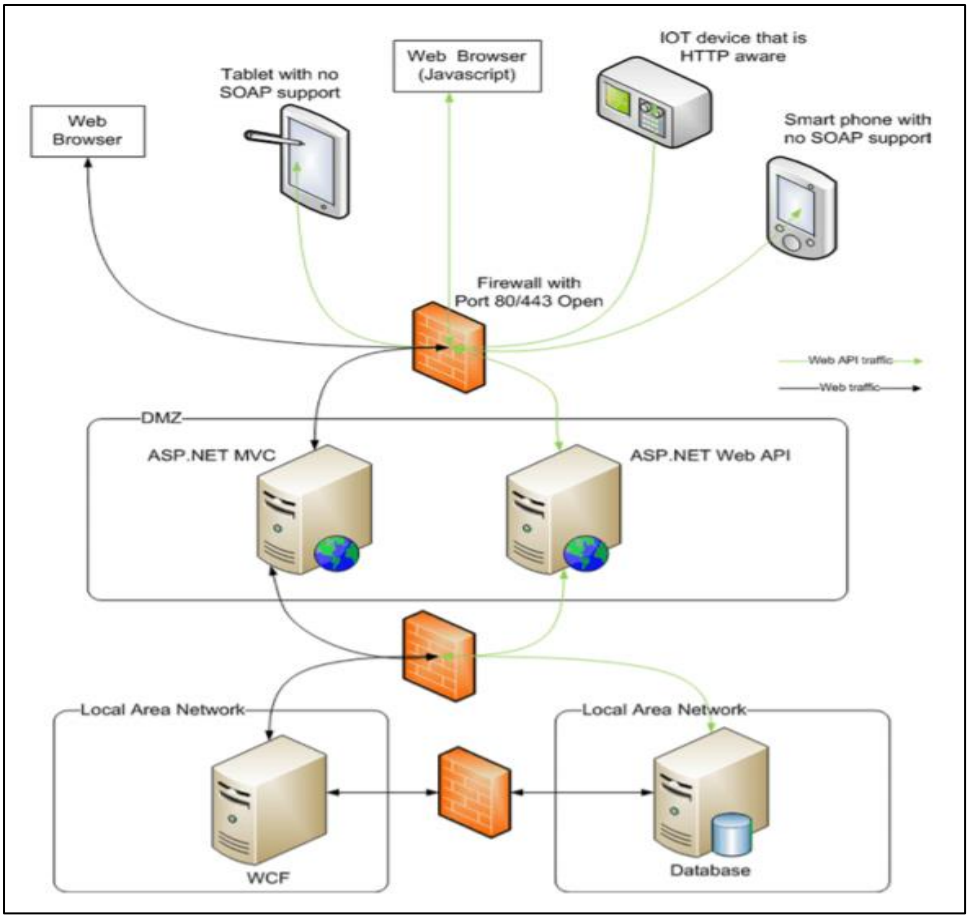

Figure 3. Generic Deployment Diagram with Web API

Database Microsoft SQL Server 2017

This tool uses the Microsoft SQL Server 2017 [11] database. Two major tasks of bulk loading the data and transformation of the data are handled in this tier. Functions in the class SQLBulkCopy [12] are used to load the data in the database efficiently.

CHAPTER THREE: USER INTERFACE

Creating an Account

An account can be considered as a container/repository for all files received from one organization/business. A user can create multiple accounts for a single organization/business to ease the data separation. By default, users can see accounts created by themselves. An account can be shared among other users to make operations easier. Figure 4 shows a form to create a new account. Users can subscribe to email notifications. Email ID field for each account will get email notifications regarding the progress of all operations on a file.

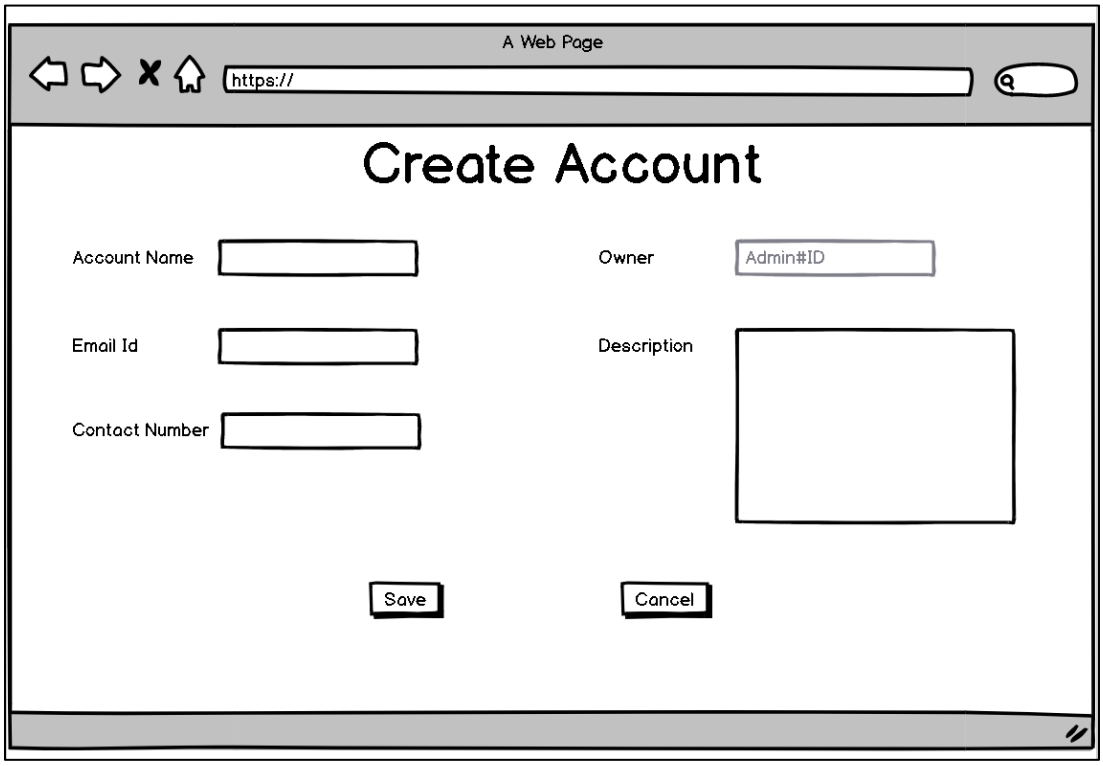

Figure 4. Add Account Form

Setting up File Template

A File Template represents a structure of the file under an account. This step of setting a File Template to an Account should be performed only once unless the file structure changes. Figure 5 shows the form to set up a file template under an account. Fields in the File Template setup form are described below.

- **Template Name** – Any name to identify a specific file template.
- **File name Regular expression [10]** – A regular expression that matches filenames from a particular organization/business e.g. an insurance firm sending home insurance data of a state on weekly basis with file names HSInsurance_CA_01-01-19.xlsx, HSInsurance_CA_01-08-19.xlsx, HSInsurance_CA_01-16-19.xlsx can have regular expression such as “HSInsurance_*” or .”HSInsurance_CA_*” if file template is for the state of California. This field is important in matching the correct file template with the input file. Input files not matching with any regular expression of any account file templates will be rejected from the further processing.
- **File Type** – The file type of input file should be selected from a list; options available are comma-separated values (.csv), Excel spreadsheets (.xlsx, .xls) and text files (supporting any extension).

- **Custom Assembly** – User has given the option to write custom logic for transforming the dataset. Assembly must implement a given interface called ITransformationProvider. This custom assembly will be executed for that specific file template which matches with the input file. More details on custom assembly are covered in Chapter 4.
- **File Template** – This field requires a sample input file for that account.

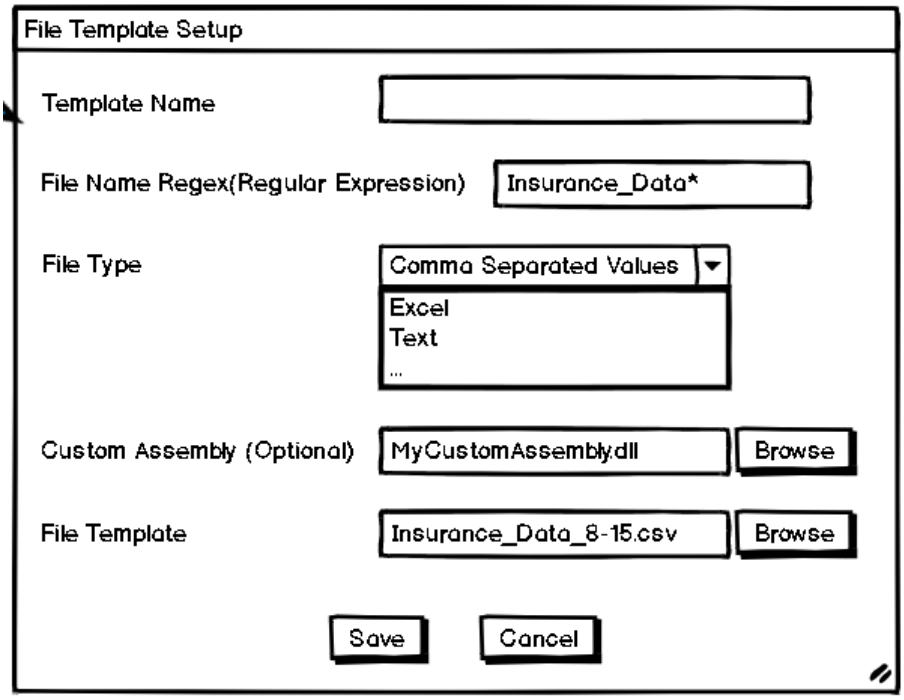

Figure 5.File Template Setup form

Attribute Association

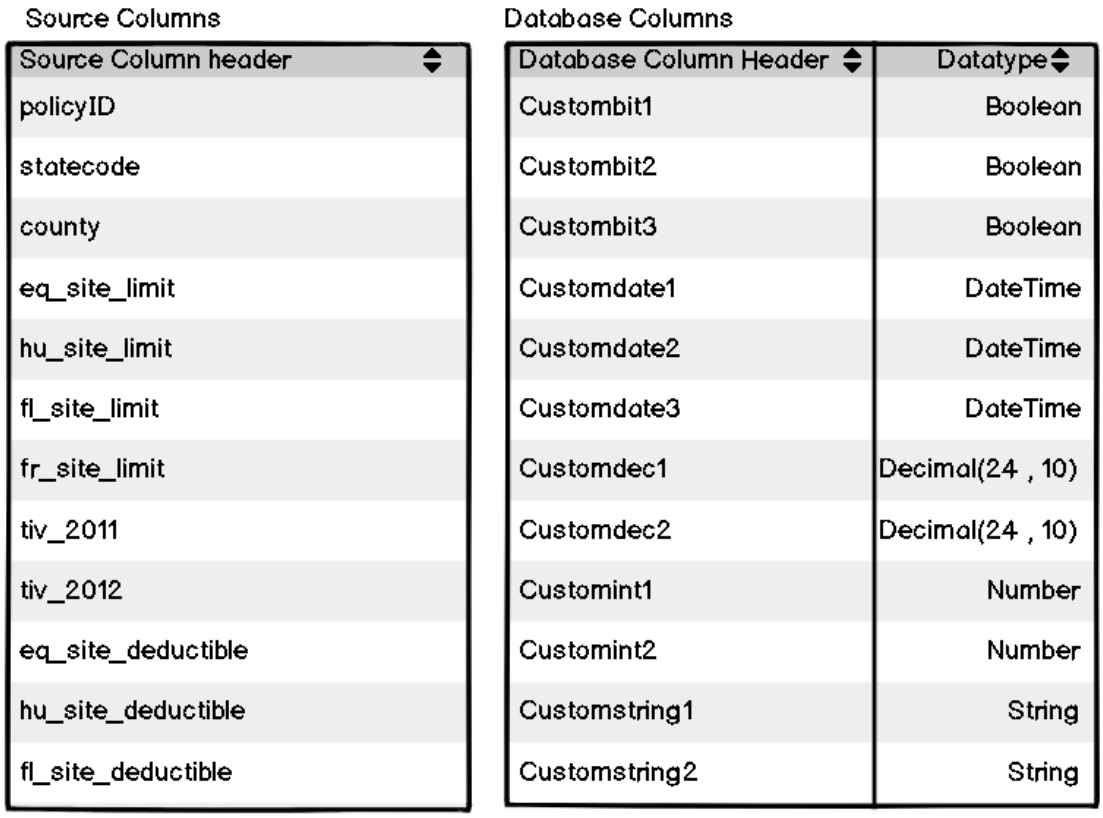

Figure 6. Source and Database Column Grid from Attribute Association Page

In this phase source-column headers from input file template are shown in a grid as shown in Figure 6. Database columns show lists of custom columns which supports five datatypes –

1. Boolean
2. DateTime
3. Decimal (24,10), Length of 24 and can store up to 10 decimal places.
4. Number
5. String

Different data types in the database column allow a user to map the matching column properly. Further operations on columns are performed based on the data type of the column e.g. on a column with datatype number/decimal all mathematical operations can be performed etc.

Source Column header	Database Column	Datatype	Uniqu	Action
policyID	Custombit1	Boolean	☑	(o)Delete
statecode	Custombit2	Boolean	☐	(o) Delete
county	Custombit3	Boolean	☐	(o) Delete
eq_site_limit	Customdate1	DateTime	☐	(o) Delete
hu_site_limit	Customdate2	DateTime	☐	(o) Delete
fl_site_limit	Customdate3	DateTime	☐	(o) Delete
fr_site_limit	Customdec1	Decimal(24 ,	☐	(o) Delete
tiv_2011	Customdec2	Decimal(24 ,	☐	(o) Delete
tiv_2012	Customint1	Number	☐	(o) Delete
eq_site_deductible	Customint2	Number	☐	(o) Delete
hu_site_deductible	Customstring1	String	☐	(o) Delete
fl_site_deductible	Customstring2	String	☐	(o) Delete

Figure 7. Mapped Attribute Grid

Once a source column and its matching database column is mapped that pair is shown in the grid shown in Figure 7. Columns with unique values can be selected from this grid by checking the checkbox. Mapping can be deleted using the delete option.

Setup Transformation Operations

This phase has two parts operation and condition. Operation is a set of transformation functions that can be applied to a column. The condition consists of constraints to a value of column before applying any operation e.g. Figure 8 shows the operation of replacing a string which equals FL.

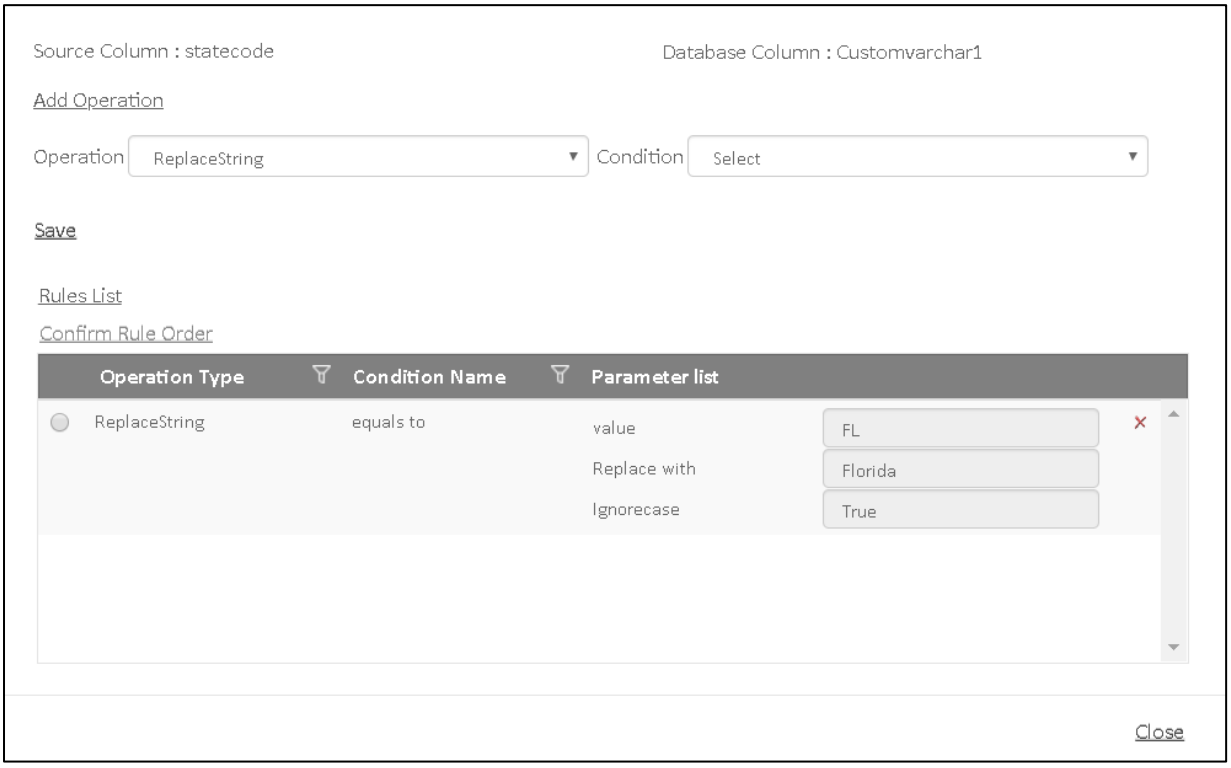

Figure 8. Transformation Setup Screen

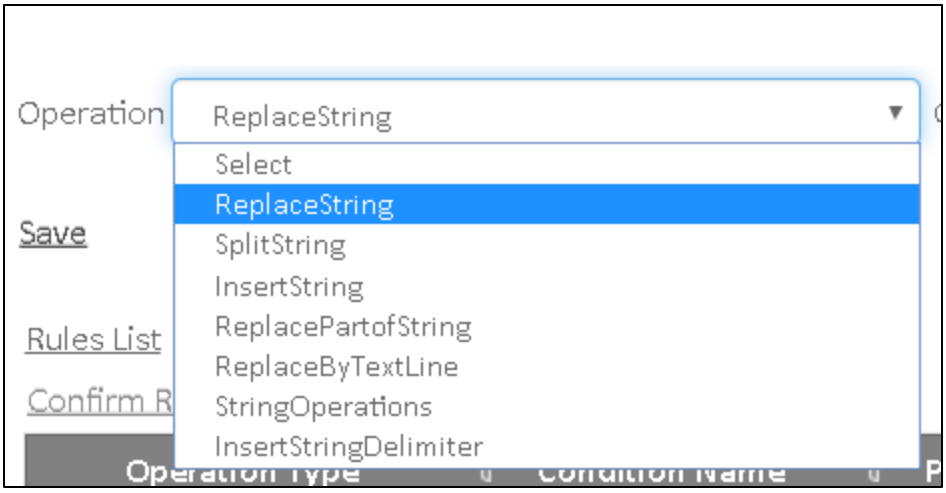

Figure 9. String Operation List

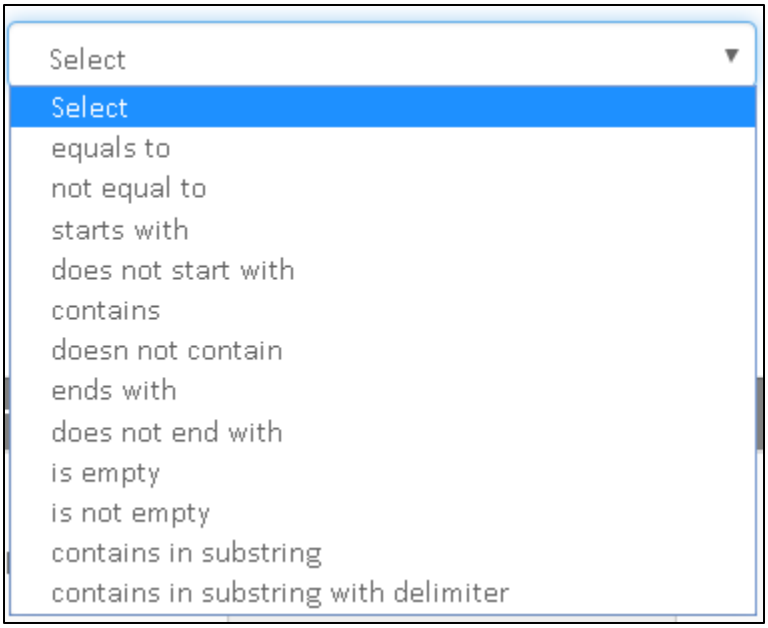

Figure 10. String Condition List

Figure 9 shows a list of operations on a column with a string datatype and Figure 10 shows a list of conditions on a string datatype column. Selection of one operation and one condition completes one transformation cycle. There can be

multiple cycles applied in a single column. Figure 11 shows a sample transformation on a string column that replaces a string equal to ‘FL’ with ‘Florida’ ignoring the case of letters.

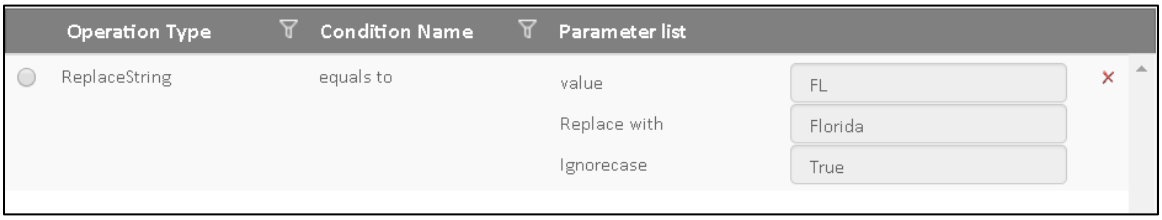

Figure 11. One Transformation on String Type Column

Data Grid

Once data load and transformation are successful, the data grid shows the dataset which is present in the database. Figure 12 shows the data grid for account ‘Insurance Data’ and template ‘FM 1’. This is a read-only grid.

Account Name: Insurance Data | File Template: FM 1 | View Data

Account ID	File Name	File Name Pattern	Customvarchar1	Customvarchar2	Customvarchar3	Customvarchar4	Customvarchar5	Customvarchar
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.265605	-82.16215	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.27125	-82.160875	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	29.869762	-82.202431	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	29.86261	-82.135475	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	29.864408	-82.135536	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	30.043814	-82.075211	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	30.045353	-82.073212	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.28364	-82.12209	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.283901	-82.122543	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.285202	-82.105835	Residential	Masonry
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.285202	-82.105835	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.285202	-82.105835	Residential	Masonry

Figure 12. Data Grid

Logging

In this automated tool, actions performed by the process are more important. Logging gives more idea on the flow of the application and to diagnose any error. This tool logs two main sections.

1. File import logging – This part logs the progress of importing a file with timestamp until the file is imported successfully or rejected.
2. Manual activity logging – This part logs all user activity on user interface e.g. adding/updating/deleting attribute mapping, adding/updating/deleting transformation cycle.

CHAPTER FOUR: SYSTEM DESIGN

Deployment Diagram

As shown in Figure 13, this tool needs one web server and a database server to operate. Users can use a browser on any device on the Internet and connect to the webserver using TCP/IP connection or URL.

Web server consists of a windows service called EmailPollingService. EmailPollingService is responsible to continuously poll on Email Server, fetch and process the files received on an email box. This service uses Exchange Web Server (EWS) managed API [5] to connect to the email box.

Web API component exposes UI services so that the browser can connect to the Web server. The reporting Service component is responsible to generate the report from the processed dataset and export as defined by the user.

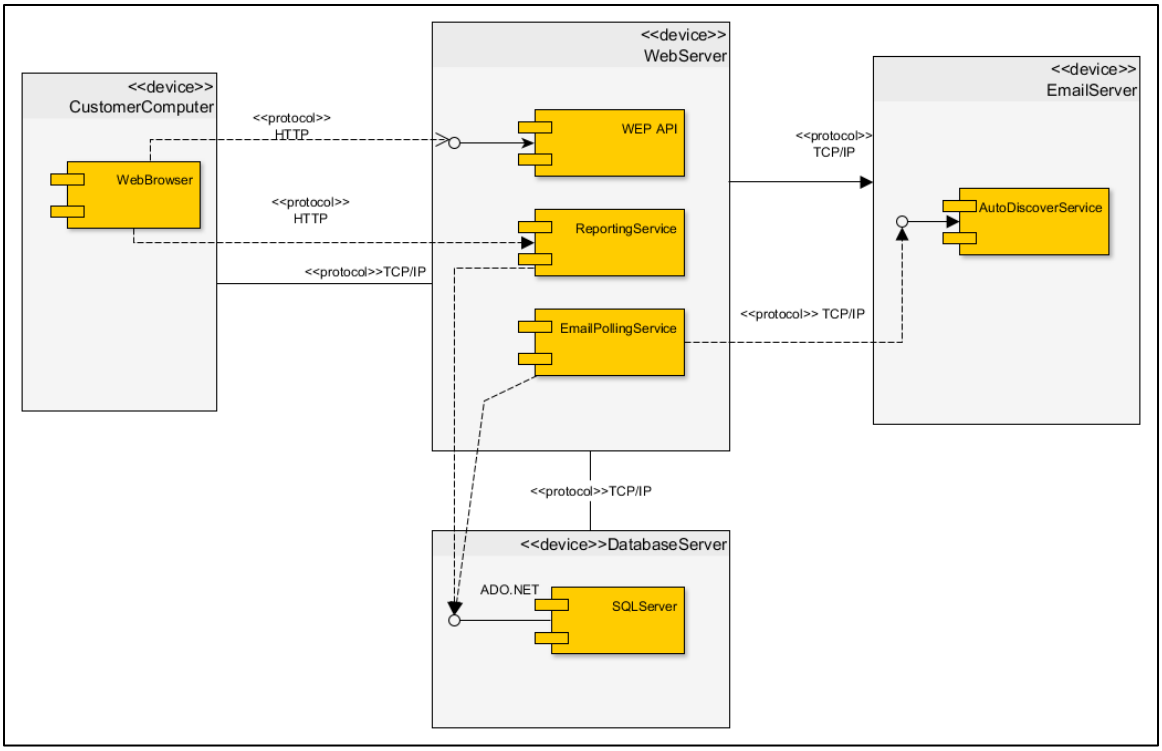

Figure 13. Deployment Diagram

Use Case Diagram

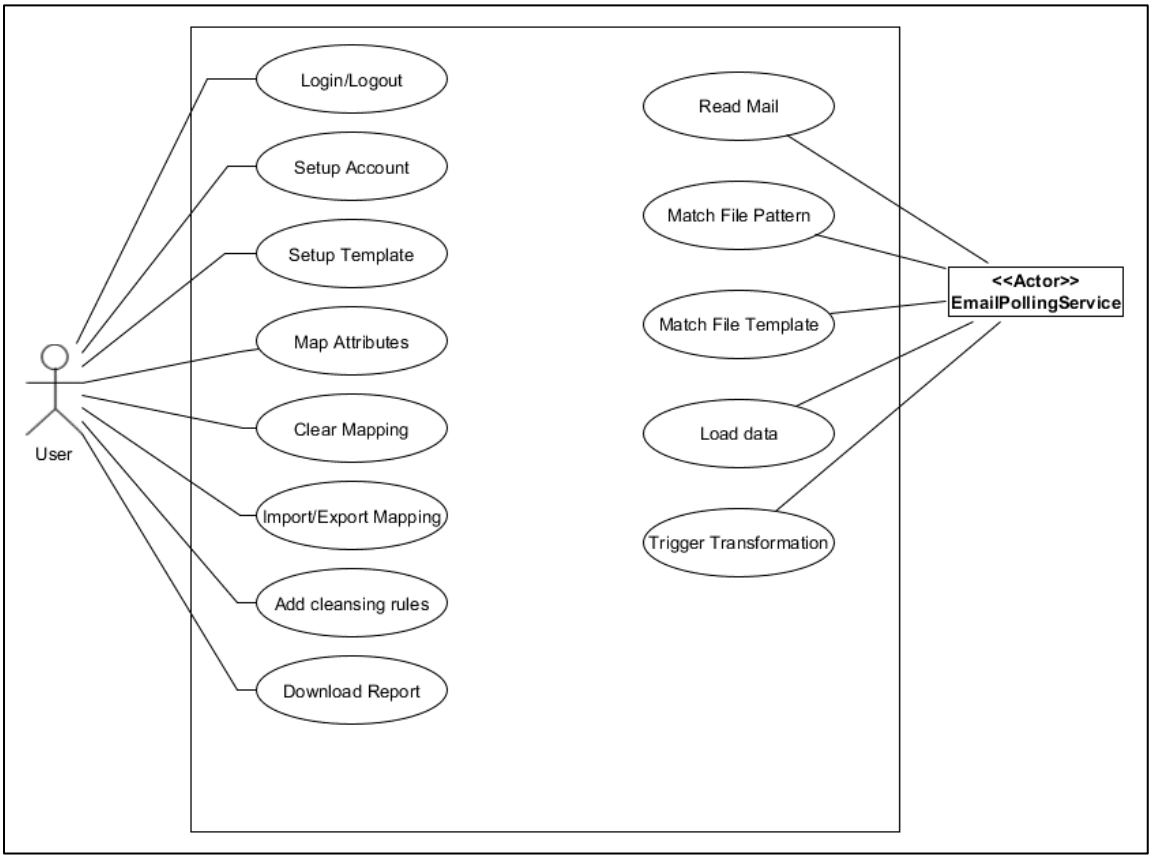

Figure 14. Use Case Diagram

Activity Diagram

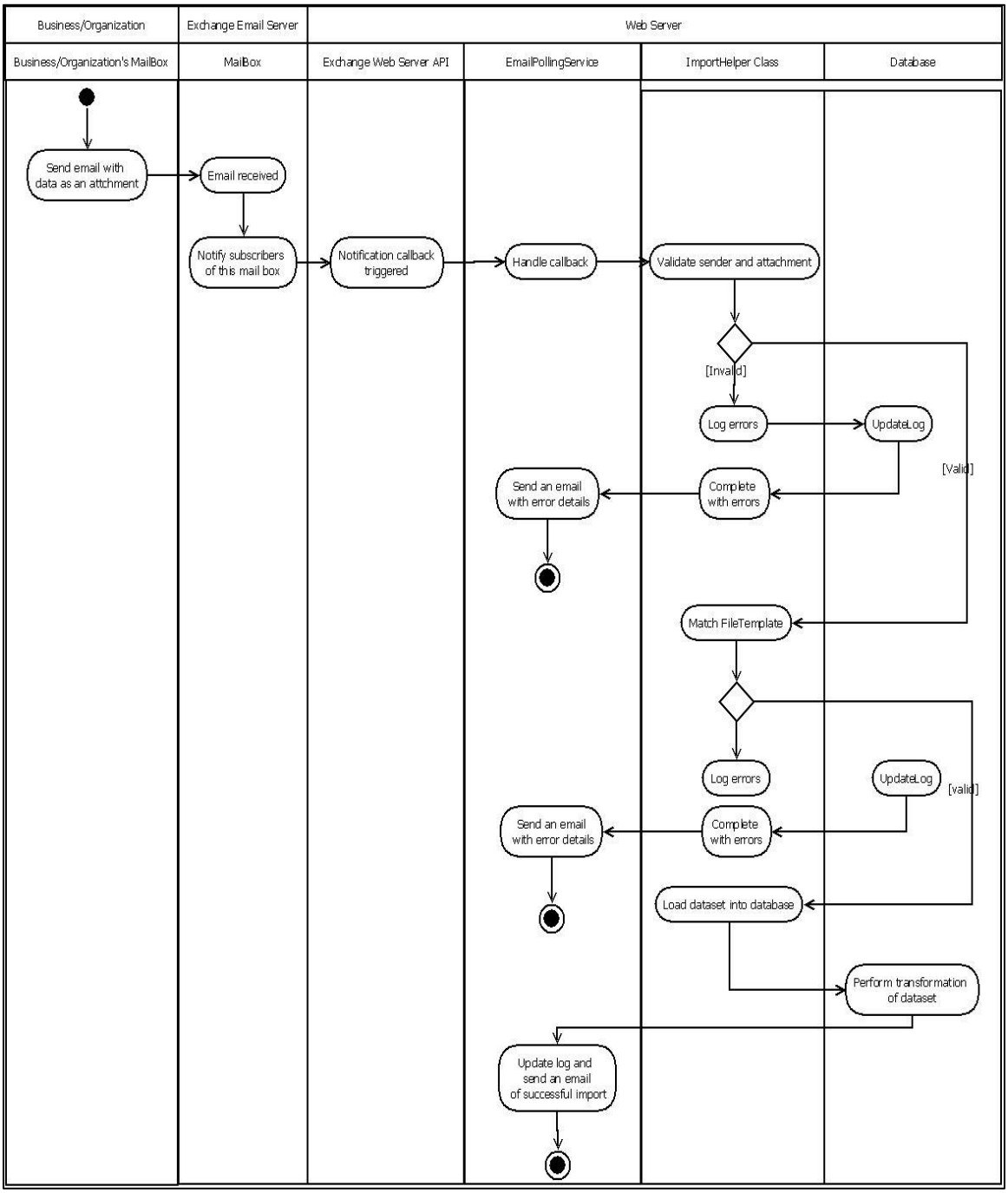

Figure 15. Activity Diagram

Sequence Diagrams

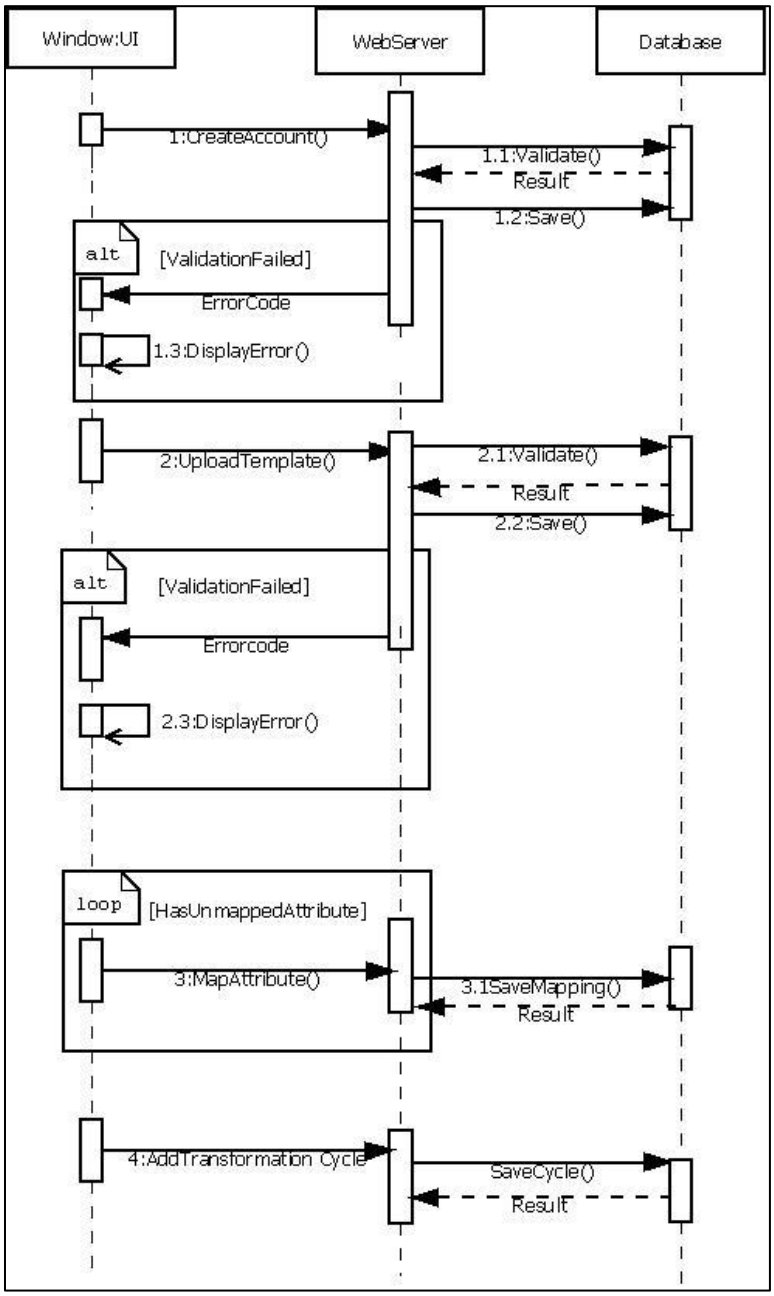

Figure 16. Sequence Diagram for Account Setup

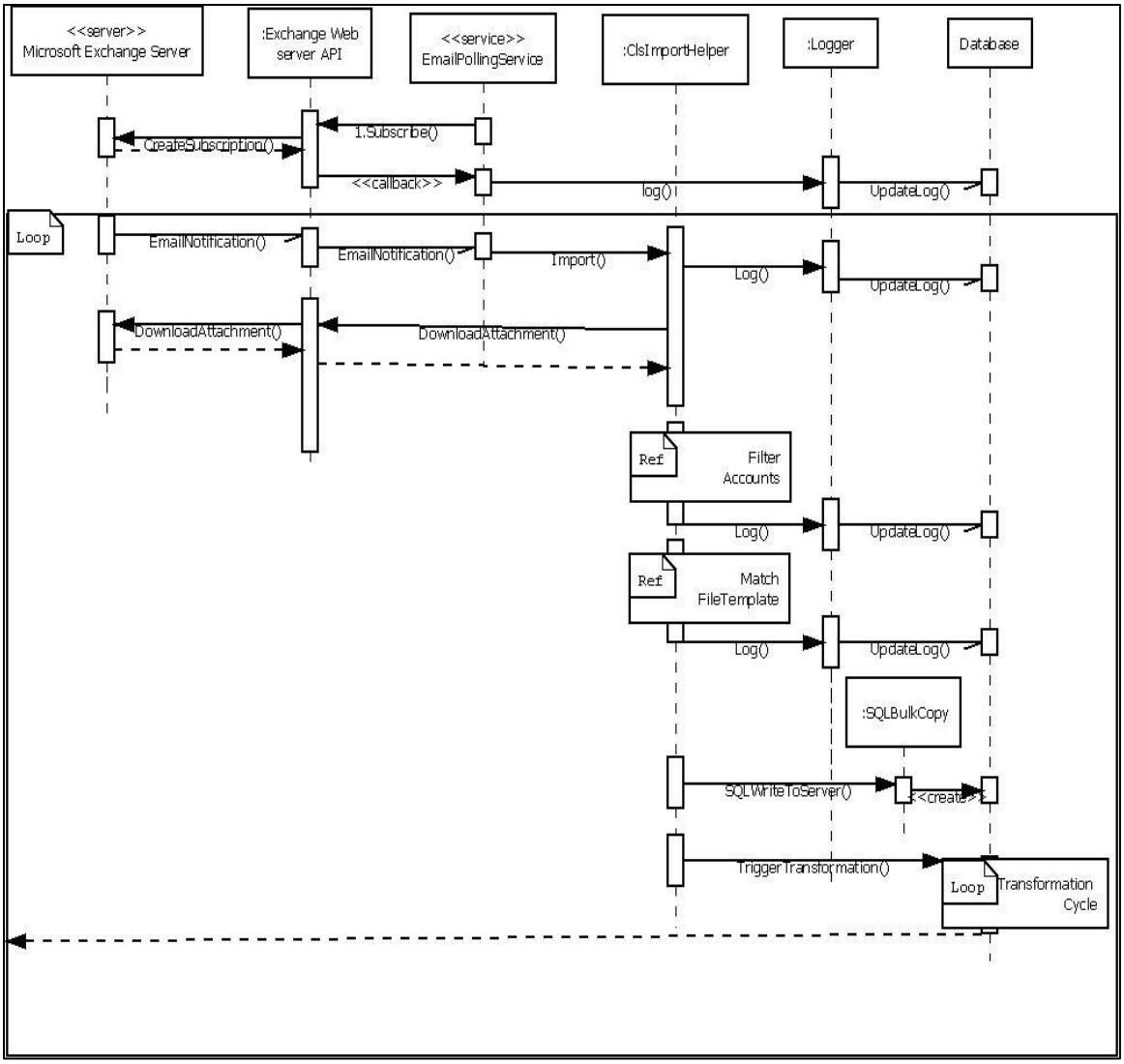

Figure 17. Sequence Diagram of Import Process

Data Import Service in Depth

Data import service is a windows service i.e. a background application that can execute under their own windows session. It makes use of Exchange Web Server (EWS) managed API [5] to interact with Mailbox. As shown in Figure 18. Data import service is an EWS client application.

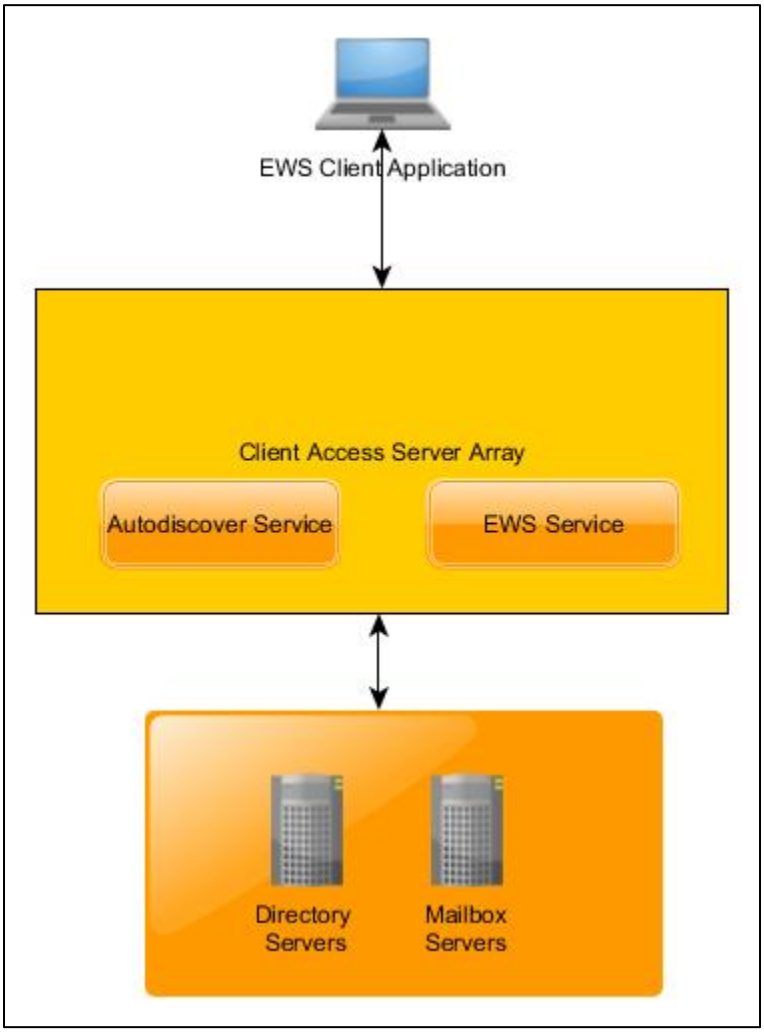

Figure 18. Exchange Web Service (EWS) Client Setup

Autodiscover Service

This service is responsible to find the closest endpoint server to the client. Exchange Web Server (EWS) managed API implements this process allowing client applications to make use of it. Figure 18 shows the signature of the method AutodiscoverUrl; it takes an email address and validation method as input and returns URL of the nearest mail server.

```
//
// Summary:
//     Initializes the Url property to the Exchange Web Services URL for the specified
//     e-mail address by calling the Autodiscover service.
//
// Parameters:
//   emailAddress:
//     The email address to use.
//
//   validateRedirectionUrlCallback:
//     The callback used to validate redirection URL.
public void AutodiscoverUrl(string emailAddress, AutodiscoverRedirectionUrlValidationCallback validateRedirectionUrlCallback);
```

Figure 19 Autodiscover Method Signature

Email Subscription

Exchange web server API provides a method to subscribe for notifications from a mailbox as new mail arrives. There are three types of subscriptions implemented in Exchange web server

1. Pull notifications – Notifications requested/pulled by client.
2. Push notifications – Notifications are sent to the client from the server in the form of a callback function.
3. Streaming notifications – Notifications are sent to the client via a real-time connection using protocol TCP/IP. Connection expires after some period.

This tool uses streaming notifications to receive email notifications. The method signature used in the tool is shown in Figure 19.

```
// Summary:
//     Subscribes to streaming notifications. Calling this method results in a call
//     to EWS.
//
// Parameters:
//   folderIds:
//     The Ids of the folder to subscribe to.
//
//   eventTypes:
//     The event types to subscribe to.
//
// Returns:
//     A StreamingSubscription representing the new subscription.
public StreamingSubscription SubscribeToStreamingNotifications(IEnumerable<FolderId> folderIds, params EventType[] eventTypes);
```

Figure 20. Streaming Notifications Method Signature

Custom Assembly for Data Transformation

This tool allows a user to upload custom assembly which can contain their logic to transform the dataset for each template. Figure 5 shows a form to create a file template with the option to upload the assembly code. The assembly must implement the given interface shown in Figure 21. It contains only one method which accepts the Datatable object and returns transformed Datatable object.

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ETLAutomation.ExtensionMethods
{
    /// <summary>
    /// Implement interface to add custom transformation
    /// tasks
    /// </summary>
    public interface ITransformProvider
    {
        DataTable Run(DataTable _ds);
    }
}
```

Figure 21. Supplied Interface

CHAPTER FIVE: TESTING

This tool automates the workflow of manual data manipulation, so all the classes were developed following test-driven development. After completion of development System testing on this tool was done manually matching expected results.

Unit testing

During the development, test cases for each method were written with the help of method specification. After each code update in a method, all test cases for that method were run and additional test cases were added as needed. After all test cases were passed code was refactored to match the acceptable standard.

The NUnit testing library [9] is used to perform unit testing. The NUnit generates basic test cases in each class. Advanced test cases were manually written for each method to validate the behavior. Sample positive test cases for method ConvertToBoolean() and GetExcelStrings() are shown in Figure 22. The addition of new test cases was done as function behavior changed. The test explorer window shows the status of all test cases in each class. Figure 23 shows some test cases under the Utility class.

```
[Test]
public void convertToBoolean_StateUnderTest_ExpectedBehavior()
{
    // Arrange
    var unitUnderTest = this.CreateUtility();
    string s = "2";

    // Act
    var result = unitUnderTest.convertToBoolean(
        s);

    // Assert
    Assert.IsFalse(result);
}


[Test]
public void GetExcelStrings_StateUnderTest_ExpectedBehavior()
{
    // Arrange
    var unitUnderTest = this.CreateUtility();

    // Act
    var result = unitUnderTest.GetExcelStrings();

    // Assert
    Assert.IsNotEmpty(result);
}
```

Figure 22 Unit Test Sample

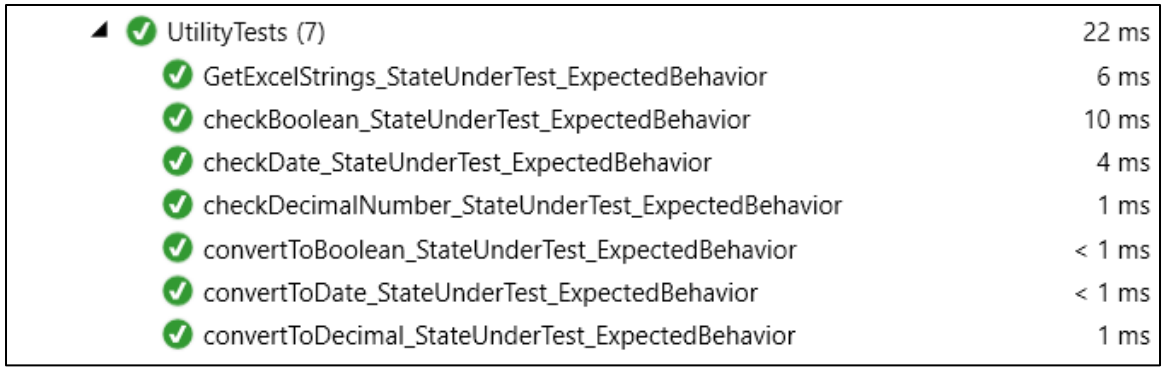

Figure 23 Unit Test Status Sample

CHAPTER SIX: FUTURE WORK

Layout Parser

In the present business ecosystem; a PDF file is the preferred form to exchange the business data. A machine has limitations on reading the PDF file text since the PDF file doesn’t define structured tags. The Optical Character Recognition (OCR) technique converts handwritten documents and images. Each file template can store the layout information to parse documents. Embedding the open source Optical Character Recognition (OCR) libraries would amplify the practical benefit of this tool.

CHAPTER SEVEN: CONCLUSIONS

This tool can be useful to correct redundant data errors and reduce manual intervention. It can process bulk emails with attachments of any size with background worker process. A user is aware of file progress with the log viewer and email notifications. Finally, the output of the processed data can be used to generate reports or exported in different formats.

This project is developed using Test Driven Development (TDD) approach and each function in this project has multiple unit test cases covering positive, negative and edge cases. Manual testing of this tool has covered different types of datasets, file types, data types and transformation cycles.

APPENDIX A: CODE OF CRITIAL PARTS

Email Service Main Class

```
internal partial class CronService : ServiceBase
    {
        private const string ConstMailInterval = "MailInterval";
        static readonly ILog Logger = LogManager.GetLogger(typeof(CronService));
        private Mailhandler _mailhandler;
        private Timer _timer;

        public CronService()
        {
            log4net.Config.XmlConfigurator.Configure();
            InitializeComponent();
        }

        protected override void OnStart(string[] args)
        {
            try
            {
                EventLog.WriteEntry("Service Starting",
EventLogEntryType.Information);
                Logger.InfoFormat("Start Method: {0}",
System.Reflection.MethodBase.GetCurrentMethod().Name);
                System.Threading.Thread.Sleep(25000);
                _mailhandler = new Mailhandler();
                ConfigurationHelper configHelper = new ConfigurationHelper();
                try
                {
                    string strInterval =
ConfigurationManager.AppSettings[ConstMailInterval];
                    double interval =
TimeSpan.FromMinutes(Convert.ToDouble(strInterval)).TotalMilliseconds;

                    _timer = new Timer {Interval = interval};
                    _timer.Elapsed += _mailhandler.ReadEmails;
                    _timer.Enabled = true;

                }
                catch (Exception ex)
                {
                    System.Threading.Thread.Sleep(1000);
                    Logger.Error(ex.Message, ex);
                    throw new Exception(ex.Message, ex);
                }

                Logger.InfoFormat("End Method: {0}",
System.Reflection.MethodBase.GetCurrentMethod().Name);
                EventLog.WriteEntry("EmailService Started",
EventLogEntryType.Information);
            }
            catch (Exception ex)
            {
                System.Threading.Thread.Sleep(1000);
                Logger.Error(ex.Message, ex);
```

```
                string msg = "EmailService could not start.\r\n" + ex.Message;
                if (ex.InnerException != null && ex.InnerException.Message.Length
> 0)
                {
                    msg += "\r\n" + ex.InnerException.Message;
                }
                EventLog.WriteEntry(msg, EventLogEntryType.Error);
                throw new Exception("Error while starting service", ex);
            }
        }

        protected override void OnStop()
        {
            EventLog.WriteEntry("EmailService Stopped",
EventLogEntryType.Information);
        }
        }
```

EmailHandler class

```
internal class Mailhandler
    {
        //System.Configuration.AppSettingsReader reader;
        private static readonly ILog Logger =
LogManager.GetLogger(typeof(Mailhandler));

        private static readonly object LockObj = new object();

        private ConfigurationHelper _configHelper;
        private ExchangeService _service;
        private string _sharedLocation;
        private ATEImportBO _boObject;
        private StreamingSubscriptionConnection _connection;
        private bool _connectionClosedInternally;

        private int _counterForEmailInterval;

        private bool _dbUp;

        /// <summary>
        ///     C0nstructor
        /// </summary>
        internal Mailhandler()
        {
            try
            {
                Init();
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
```

```
                //throw ex;
            }
        }

        private void Init()
        {
            var connectionString = ConfigurationHelper.GetConnectionString();


            if (!Helper.CheckConnection(connectionString))
            {
                _dbUp = false;
            }

            else
            {
                _boObject = new ATEImportBO(connectionString);
                XmlConfigurator.Configure();
                var crypt = new Cryptography();
                _service = new ExchangeService(ExchangeVersion.Exchange2010_SP1);
                _configHelper = new ConfigurationHelper();
                _service.Credentials = new
WebCredentials(crypt.Decrypt(ConfigurationManager.AppSettings["username"]),
crypt.Decrypt(ConfigurationManager.AppSettings["password"]));

_service.AutodiscoverUrl(crypt.Decrypt(ConfigurationManager.AppSettings["username"
]), RedirectionCallback);

                _sharedLocation =
Convert.ToString(_configHelper.GetXML(Const.ConstSharedLocation));
                if (!Directory.Exists(_sharedLocation))
                    Directory.CreateDirectory(_sharedLocation);
                SetStreamingNotifications(_service);
                _dbUp = true;
            }
        }

        private static bool RedirectionCallback(string url)
        {
            // Return true if the URL is an HTTPS URL.
            return url.ToLower().StartsWith("https://");
        }

        private static bool CheckConnection(string connectionString)
        {
            var retryCount = 5;
            var interval = 120000;

            var configHelper = new ConfigurationHelper();
            try
            {
                var strInterval =
configHelper.GetXML(Const.ConstDatabaseWaitingTime);
                var strRetryCount =
configHelper.GetXML(Const.ConstRetryConnectionAttempts);
```

```
                if (!int.TryParse(

Convert.ToString(TimeSpan.FromMinutes(Convert.ToDouble(strInterval)).TotalMillisec
onds,
                        CultureInfo.InvariantCulture), out interval)) interval =
120000;

                if (!int.TryParse(strRetryCount, out retryCount)) retryCount = 5;
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
            }


            var isValid = false;
            for (var index = 0; index < retryCount; index++)
                if (!Helper.CheckConnection(connectionString))
                {
                    Thread.Sleep(interval);
                }
                else
                {
                    isValid = true;
                    break;
                }

            return isValid;
        }

        private void SendDBErrorEmail()
        {
            var configHelper = new ConfigurationHelper();
            try
            {
                var fromMail = configHelper.GetXML(Const.ConstfromMail);
                var toMail =
configHelper.GetXML(Const.ConstEmailServiceFailureRecepients);
                var smtpServer = configHelper.GetXML(Const.ConstsmtpServer);
                //Helper.SendEmail(fromMail, toMail, smtpServer, subject, message,
MailPriority.High);
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
            }
        }

        private void RenewAutoDiscoverUrl()
        {
            try
            {
                lock (LockObj)
                {
```

```
                    _connectionClosedInternally = true;
                    var crypt = new Cryptography();
                    _service.Credentials = new
WebCredentials(crypt.Decrypt(ConfigurationManager.AppSettings["username"]),
crypt.Decrypt(ConfigurationManager.AppSettings["password"]));
                    Logger.InfoFormat("EWS Url - {0}", _service.Url);
                    if (_connection != null)
                    {
                        if (_connection.CurrentSubscriptions.Any())
                        {
                            _connection.OnDisconnect -= OnDisconnect;
                            _connection.OnNotificationEvent -= OnEvent;
                            _connection.OnSubscriptionError -= OnError;
                        }

                        if (_connection.IsOpen) _connection.Close();

                        foreach (var subscription in
_connection.CurrentSubscriptions)
                            _connection.RemoveSubscription(subscription);


                        _connection = null;
                    }


_service.AutodiscoverUrl(crypt.Decrypt(ConfigurationManager.AppSettings["username"
]), RedirectionCallback);

                    SetStreamingNotifications(_service);

                    Logger.InfoFormat("New EWS Url - {0}", _service.Url);
                }
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
            }
        }

        internal void Initialize(object obj, ElapsedEventArgs args)
        {
            try
            {
                var crypt = new Cryptography();
                _service = new ExchangeService(ExchangeVersion.Exchange2010_SP1)
                {
                    Credentials = new
WebCredentials(crypt.Decrypt(ConfigurationManager.AppSettings["username"]),
crypt.Decrypt(ConfigurationManager.AppSettings["password"]))
                };

_service.AutodiscoverUrl(crypt.Decrypt(ConfigurationManager.AppSettings["username"
]), RedirectionCallback);
```

```
                _sharedLocation =
Convert.ToString(_configHelper.GetXML(Const.ConstSharedLocation));
                if (!Directory.Exists(_sharedLocation))
                    Directory.CreateDirectory(_sharedLocation);
                SetStreamingNotifications(_service);
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
            }
        }

        private void SetStreamingNotifications(ExchangeService service)
        {
            try
            {
                Logger.Info("SetStreamingNotifications Start");
                lock (LockObj)
                {
                    var streamingsubscription =
service.SubscribeToStreamingNotifications(
                        new FolderId[] { WellKnownFolderName.Inbox },
                        EventType.NewMail);
                    _connection = new StreamingSubscriptionConnection(service,
30);
                    _connection.AddSubscription(streamingsubscription);
                    // Delegate event handlers.

                    _connection.OnNotificationEvent += OnEvent;
                    _connection.OnSubscriptionError += OnError;
                    _connection.OnDisconnect += OnDisconnect;
                    _connection.Open();
                }

                Logger.Info("SetStreamingNotifications End");
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
            }
        }

        private void OnDisconnect(object sender, SubscriptionErrorEventArgs args)
        {
            try
            {
                Logger.Info("OnDisconnect Start");
                lock (LockObj)
                {
                    var conn = (StreamingSubscriptionConnection)sender;
                    _connection = conn;

                    if (!_connectionClosedInternally)
                    {
                        try
```

```
                        {
                            if (!conn.IsOpen)
                                conn.Open();
                        }
                        catch (Exception ex)
                        {
                            Logger.Info("Connection already Closed", ex);
                            _connection = null;
                        }

                        Logger.Info("Streaming Connection Renewed");
                    }

                    //RenewAutoDiscoverURL(conn);
                    //if (conn != null && !conn.IsOpen &&
conn.CurrentSubscriptions.Count() > 0)
                    //{
                    //    logger.Info("Reopening Connection");
                    //    conn.Open();
                    //}
                }

                Logger.Info("OnDisconnect End");
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
            }
            finally
            {
                _connectionClosedInternally = false;
            }
        }

        private void OnError(object sender, SubscriptionErrorEventArgs args)
        {
            var e = args.Exception;
            Logger.Error("Error in Streaming notifications", e);
        }

        private void OnEvent(object sender, NotificationEventArgs args)
        {
            try
            {
                lock (LockObj)
                {
                    foreach (var notification in args.Events)
                        if (notification is ItemEvent)
                        {
                            // The NotificationEvent for an email message is an
ItemEvent.
                            var itemEvent = (ItemEvent)notification;
                            Item item = null;
                            Logger.InfoFormat("Email received Unique Id : {0}",
itemEvent.ItemId.UniqueId);
```

```
                            try
                            {
                                item = Item.Bind(_service, itemEvent.ItemId);
                                var message = EmailMessage.Bind(_service,
itemEvent.ItemId,
                                    new PropertySet(ItemSchema.Attachments,
ItemSchema.Subject,
                                        EmailMessageSchema.Sender));
                                Logger.InfoFormat("Email Subject : {0} Sender :
{1}", message.Subject, message.Sender);
                            }
                            catch (Exception ex)
                            {
                                Logger.Error("Error Binding Item to Email", ex);
                                if (item != null)
                                {
                                    item.Delete(DeleteMode.HardDelete);
                                    Logger.Info("Deleted Item Reason - Invalid
EmailType");
                                }
                            }

                            DownloadFiles(sender, null);
                        }
                    }
                }
                catch (Exception ex)
                {
                    Logger.Error(ex.Message, ex);
                }
            }

            internal void ReadEmails(object obj, ElapsedEventArgs e)
            {
                try
                {
                    Logger.InfoFormat("Start : {0}",
MethodBase.GetCurrentMethod().Name);
                    GC.Collect();

                    if (!_dbUp)
                    {
                        var connectionString =
ConfigurationHelper.GetConnectionString();
                        if (CheckConnection(connectionString))
                            Init();
                        else
                            SendDBErrorEmail();
                    }
                    else
                    {
                        DownloadFiles(obj, e);
```

```
                    _counterForEmailInterval = (_counterForEmailInterval + 1) %
Const.DeleteMod;

                    if (_counterForEmailInterval == 0)
                    {
                        RenewAutoDiscoverUrl();
                        DeleteEmailWithoutAttachments();
                        //DeleteEmailWithIncompatibleFiletype();
                        DeleteExpiredEmails();
                    }
                    else
                    {
                        StreamingConnectionCheck();
                    }
                }

                _connectionClosedInternally = false;
                Logger.InfoFormat("End : {0}",
MethodBase.GetCurrentMethod().Name);
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
            }
        }

        /// <summary>
        ///     Download Unread mails having attachments
        /// </summary>
        //[Synchronization(true)]
        internal void DownloadFiles(object obj, ElapsedEventArgs e)
        {
            try
            {
                lock (LockObj)
                {
                    Logger.Info("Start :" + MethodBase.GetCurrentMethod().Name);
                    var view = new ItemView(1000, 0, OffsetBasePoint.Beginning);
                    view.OrderBy.Add(ItemSchema.DateTimeReceived,
SortDirection.Ascending);

                    SearchFilter filter = new
SearchFilter.SearchFilterCollection(LogicalOperator.And,
                        new SearchFilter.IsEqualTo(ItemSchema.HasAttachments,
true),
                        new SearchFilter.IsEqualTo(EmailMessageSchema.IsRead,
false));
                    FindItemsResults<Item> findResults;
                    do
                    {
                        findResults =
_service.FindItems(WellKnownFolderName.Inbox, filter, view);
                        Logger.InfoFormat("Found {0} unread emails ",
findResults.Items.Count);
                        foreach (var item in findResults)
```

```
                        {
                            EmailMessage message = null;
                            try
                            {
                                message = EmailMessage.Bind(_service, item.Id,
                                    new PropertySet(ItemSchema.DateTimeReceived,
ItemSchema.Attachments,
                                        ItemSchema.HasAttachments,
ItemSchema.Subject, EmailMessageSchema.Sender));
                            }
                            catch (Exception ex)
                            {
                                Logger.Error("Error Binding Item to Email", ex);
                            }

                            if (message == null)
                            {
                                Logger.Info("Deleted Item from Inbox Reason -
Ïnvalid Emailtype");
                                item.Delete(DeleteMode.HardDelete);
                                continue;
                            }

                            if (message.Attachments.Count > 0)
                                if
(!Convert.ToString(_configHelper.GetXML(Const.ConstFileTypes)).Split(',').Any(p =>
                                    message.Attachments.Any(att =>
                                        att.Name.EndsWith(p,
StringComparison.InvariantCultureIgnoreCase))))
                                {
                                    Logger.InfoFormat(
                                        "Deleted Email Reason-Doesn't contain
compatible filetypes , Sender:{0} , Subject:{1} , Attachments:{2}",
                                        message.Sender, message.Subject,
                                        string.Join(",",
message.Attachments.Select(p => p.Name).ToArray()));
                                    message.Delete(DeleteMode.HardDelete);
                                    continue;
                                }

                            message.Attachments.ToList().ForEach(attachment =>
                            {
                                var fattachment = attachment as FileAttachment;
                                if (fattachment == null)
                                {
                                    Logger.Error("Invalid Attachment ");
                                    return;
                                }

                                Logger.InfoFormat("Attached File name : {0}",
fattachment.Name);
                                if
(Convert.ToString(_configHelper.GetXML(Const.ConstFileTypes))
                                    .Split(',').Any(p =>
```

```
                                        fattachment.Name.EndsWith(p,
StringComparison.InvariantCultureIgnoreCase)))
                                {
                                    if (new FileInfo(Path.Combine(_sharedLocation,
attachment.Name)).Exists)
                                        new FileInfo(_sharedLocation +
attachment.Name).Delete();

                                    Logger.InfoFormat("Started downloading File
:{0}", attachment.Name);
                                    fattachment.Load(_sharedLocation +
attachment.Name);
                                    Logger.InfoFormat("Completed downloading File
:{0}", attachment.Name);

                                    Logger.InfoFormat("Started Importing File
:{0}", attachment.Name);
                                    try
                                    {
                                        _boObject.ImportFile(_sharedLocation +
attachment.Name, message.Sender.Address);
                                    }
                                    catch (Exception ex)
                                    {
                                        Logger.Error("Error in Importing File",
ex);
                                    }
                                    finally
                                    {
                                        if (new FileInfo(_sharedLocation +
attachment.Name).Exists)
                                            new FileInfo(_sharedLocation +
attachment.Name).Delete();
                                    }

                                    Logger.InfoFormat("Completed Importing File
:{0}", attachment.Name);
                                }
                                else
                                {
                                    Logger.InfoFormat("FileType is not supported
File : {0}", fattachment.Name);
                                }
                            });
                            message.IsRead = true;
                            message.Update(ConflictResolutionMode.AutoResolve);
                        }

                        if (findResults.NextPageOffset.HasValue) view.Offset =
findResults.NextPageOffset.Value;
                    } while (findResults.MoreAvailable);

                    Logger.Info("End :" + MethodBase.GetCurrentMethod().Name);
                }
            }
```

```
            catch (Exception ex)
            {
                Logger.Error("Error in Downloading unread attchment", ex);
            }
        }

        private void DeleteEmailWithoutAttachments()
        {
            try
            {
                lock (LockObj)
                {
                    Logger.Info("Start :" + MethodBase.GetCurrentMethod().Name);
                    //soft Delete all mails without attachments
                    var view = new ItemView(1000, 0, OffsetBasePoint.Beginning);
                    view.OrderBy.Add(ItemSchema.DateTimeReceived,
SortDirection.Ascending);
                    SearchFilter filter = new
SearchFilter.SearchFilterCollection(LogicalOperator.And,
                        new SearchFilter.IsEqualTo(ItemSchema.HasAttachments,
false));
                    FindItemsResults<Item> findResults;

                    do
                    {
                        findResults =
_service.FindItems(WellKnownFolderName.Inbox, filter, view);
                        Logger.InfoFormat("Found {0} read emails without
attachment", findResults.Items.Count);
                        if (findResults.Items.Count > 0)
                            findResults.Items.ToList().ForEach(item =>
                            {
                                try
                                {
                                    EmailMessage message = null;
                                    try
                                    {
                                        message = EmailMessage.Bind(_service,
item.Id,
                                            new
PropertySet(ItemSchema.DateTimeReceived, EmailMessageSchema.Sender,
                                                ItemSchema.Subject));
                                    }
                                    catch (Exception ex)
                                    {
                                        Logger.Error("Error Binding Item to
Email", ex);
                                    }

                                    if (message != null)
                                        Logger.InfoFormat(
                                            "Deleted Email Reason-Doesn't contain
attachments , Sender:{0} , Subject:{1} ",
                                            message.Sender, message.Subject);
```

```
                                item.Delete(DeleteMode.HardDelete);
                            }
                            catch (Exception exc)
                            {
                                Logger.Error(exc.Message, exc);
                            }
                        });

                    if (findResults.NextPageOffset.HasValue) view.Offset =
findResults.NextPageOffset.Value;
                } while (findResults.MoreAvailable);

                Logger.Info("End :" + MethodBase.GetCurrentMethod().Name);
            }
        }
        catch (Exception ex)
        {
            Logger.Error(ex.Message, ex);
        }
    }

    private void DeleteEmailWithIncompatibleFiletype()
    {
        try
        {
            lock (LockObj)
            {
                Logger.Info("Start :" + MethodBase.GetCurrentMethod().Name);
                var view = new ItemView(1000, 0, OffsetBasePoint.Beginning);
                view.OrderBy.Add(ItemSchema.DateTimeReceived,
SortDirection.Ascending);
                SearchFilter filter = new
SearchFilter.SearchFilterCollection(LogicalOperator.And,
                    new SearchFilter.IsEqualTo(ItemSchema.HasAttachments,
true),
                    new SearchFilter.IsEqualTo(EmailMessageSchema.IsRead,
true));
                FindItemsResults<Item> findResults;

                do
                {
                    findResults =
_service.FindItems(WellKnownFolderName.Inbox, filter, view);
                    Logger.InfoFormat("Found {0} read emails with attachment",
findResults.Items.Count);

                    if (findResults.Items.Count > 0)
                        findResults.Items.ToList().OrderByDescending(p =>
p.DateTimeReceived).ToList().ForEach(
                            item =>
                            {
                                EmailMessage message = null;
                                try
                                {
```

```
                                    message = EmailMessage.Bind(_service,
item.Id,
                                        new
PropertySet(ItemSchema.DateTimeReceived, ItemSchema.Attachments,
                                            ItemSchema.HasAttachments,
EmailMessageSchema.Sender,
                                            ItemSchema.Subject));
                                }
                                catch (Exception ex)
                                {
                                    Logger.Error("Error Binding Item to
Email", ex);
                                }

                                if (message == null)
                                {
                                    item.Delete(DeleteMode.HardDelete);
                                    return;
                                }

                                if (message.HasAttachments &&
message.Attachments.Count > 0)
                                {
                                    if
(!Convert.ToString(_configHelper.GetXML(Const.ConstFileTypes)).Split(',')
                                        .Any(p => message.Attachments.Any(att
=>
                                            att.Name.EndsWith(p,
StringComparison.InvariantCultureIgnoreCase))))
                                    {
                                        Logger.InfoFormat("Attachments : {0}",
                                            string.Join(",",
message.Attachments.Select(p => p.Name).ToArray()));
                                        message.Delete(DeleteMode.HardDelete);
                                        Logger.InfoFormat(
                                            "Deleted Email Reason-Doesn't
contain compatible filetypes , Sender:{0} , Subject:{1} , Attachments:{2}",
                                            message.Sender, message.Subject,
                                            string.Join(",",
message.Attachments.Select(p => p.Name).ToArray()));
                                    }
                                    else
                                    {
                                        Logger.InfoFormat("Email has valid
attachments {0}",
                                            string.Join(",",
message.Attachments.Select(p => p.Name).ToArray()));
                                    }
                                }
                            });
                        if (findResults.NextPageOffset.HasValue) view.Offset =
findResults.NextPageOffset.Value;
                    } while (findResults.MoreAvailable);

                    Logger.Info("End :" + MethodBase.GetCurrentMethod().Name);
```

```
                }
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
            }
        }

        private void DeleteExpiredEmails()
        {
            try
            {
                lock (LockObj)
                {
                    Logger.Info("Start :" + MethodBase.GetCurrentMethod().Name);
                    var view = new ItemView(1000, 0, OffsetBasePoint.Beginning);
                    view.OrderBy.Add(ItemSchema.DateTimeReceived,
SortDirection.Ascending);
                    var retentionDays = Const.ConstEmailRetentionDays;
                    try
                    {
                        if
(int.TryParse(_configHelper.GetXML(Const.ConstEmailRetentionTime), out
retentionDays))
                        {
                        }
                        else
                        {
                            Logger.Error("Invalid RetentionDays specified in
configuration file");
                        }
                    }
                    catch (Exception)
                    {
                        Logger.Error("Invalid RetentionDays specified in
configuration file");
                    }

                    var ts = new TimeSpan(retentionDays, 0, 0, 0);
                    SearchFilter filter = new
SearchFilter.SearchFilterCollection(LogicalOperator.And,
                        new SearchFilter.IsLessThan(ItemSchema.DateTimeReceived,
DateTime.UtcNow.Subtract(ts)),
                        new SearchFilter.IsEqualTo(EmailMessageSchema.IsRead,
true));
                    FindItemsResults<Item> findResults;
                    do
                    {
                        findResults =
_service.FindItems(WellKnownFolderName.Inbox, filter, view);
                        Logger.InfoFormat("Found {0} expired emails",
findResults.Items.Count);
                        if (findResults.Items.Count > 0)
                            findResults.Items.ToList().ToList().ForEach(item =>
                            {
```

```
                                EmailMessage message = null;
                                try
                                {
                                    message = EmailMessage.Bind(_service, item.Id,
                                        new
PropertySet(ItemSchema.DateTimeReceived, ItemSchema.Attachments,
                                            ItemSchema.HasAttachments,
EmailMessageSchema.Sender, ItemSchema.Subject,
                                            ItemSchema.DateTimeReceived));
                                }
                                catch (Exception ex)
                                {
                                    Logger.Error("Error Binding Item to Email",
ex);
                                }

                                if (message == null)
                                {
                                    item.Delete(DeleteMode.HardDelete);
                                    return;
                                }

                                message.Delete(DeleteMode.HardDelete);
                                Logger.InfoFormat(
                                    "Deleted Email Reason - Expired Retention
period {3} Received Time : {2} , Sender:{0} , Subject:{1} ",
                                    message.Sender, message.Subject,
message.DateTimeReceived, retentionDays);
                            });

                        if (findResults.NextPageOffset.HasValue) view.Offset =
findResults.NextPageOffset.Value;
                    } while (findResults.MoreAvailable);

                    Logger.Info("End :" + MethodBase.GetCurrentMethod().Name);
                }
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
            }
        }

        private void StreamingConnectionCheck()
        {
            try
            {
                Logger.Info("Start :" + MethodBase.GetCurrentMethod().Name);

                if (_connection == null) RenewAutoDiscoverUrl();
                Logger.Info("End :" + MethodBase.GetCurrentMethod().Name);
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message, ex);
```

```
            }
        }
    }
```

REFERENCES

[1] AngularJS: Developer Guide. Retrieved from <https://docs.angularjs.org/guide> (n.d.).

[2] Caputo, Linda. *EWS applications and the Exchange architecture.* Retrieved from <https://docs.microsoft.com/en-us/exchange/client-developer/exchange-web-services/ews-applications-and-the-exchange-architecture>, 2015.

[3] Cote, Christian, et al. *SQL Server 2017 Integration Services Cookbook: ETL Techniques to Load and Transform Data from Various Sources Using SQL Server 2017 Integration Services.* Packt Publishing, 2017.

[4] Elfassy, David. *Mastering Microsoft Exchange Server 2013*. Sybex, 2014

[5] Exchange Web Server managed API available from <https://docs.microsoft.com/en-us/exchange/client-developer/exchange-web-services/get-started-with-ews-managed-api-client-applications> (cit. 2019-04-02).

[6] Jorgensen, Adam. *Microsoft SQL Server 2012 Bible.* Wiley, 2012. pp.

[7] Lakshmiraghavan, Badrinarayanan. *Pro ASP.NET Web API Security: Securing ASP.NET Web API.* Apress, 2013. pp. 8-10.

[8] Nagel, Christian. *Professional C# 7 and .NET Core 2.0*. Wrox, a Wiley Brand, 2018.

[9] NUnit testing Available from <https://github.com/nunit/docs/wiki>. (cit. 2019-04-02).

[10] Regular Expression (Wikipedia) Last revision on 5th April 2019. Available from <https://en.wikipedia.org/wiki/Regular_expression>. (cit 2019-04-05)

[11] SQL Server 2017. Available from <https://www.microsoft.com/en-us/sql-server>. (cit. 2019-04-02).

[12] SQLBulkCopy Class Available from <https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqlbulkcopy>. (cit. 2019-04-02).

[13] Wasson, Mike. *Routing in ASP.NET Web API.* Retrieved from <https://docs.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api>