**Data Match Enterprise API Guide (v 1.0.4.0)**

****

[Introduction 4](#_Toc336441501)

[Overview 4](#_Toc336441502)

[Match Definitions 4](#_Toc336441503)

[Matching Scores 5](#_Toc336441504)

[Classes, Properties, and Methods 5](#_Toc336441505)

[MatchEngine Class 6](#_Toc336441506)

[MatchEngine Methods 6](#_Toc336441507)

[MatchDefinitionSingle Class 6](#_Toc336441508)

[MatchDefinitionSingle Properties 6](#_Toc336441509)

[see Match Definitions 7](#_Toc336441510)

[MatchDefinitionSingle Methods 7](#_Toc336441511)

[MultipleMatchDefinitionsManager Class 7](#_Toc336441512)

[MultipleMatchDefinitionsManager Methods 7](#_Toc336441513)

[Example of use 7](#_Toc336441514)

# Introduction

The Data Match Enterprise API is a component written by Data Ladder for state of the art fuzzy matching, data formatting and data cleansing – amongst its most common uses are duplicate prevention, inquiry, deduplication and merge/purge. The Data Match Enterprise API splits and cases names and addresses, generates match keys for phonetic matching, generates 3-grams for more accurate fuzzy match and grades matching records. The component provides a compact and efficient solution to the problems of data quality and duplication on any Windows based system. This is the help file for the .NET Framework Data Match Enterprise API. API is written in C# programming language. This document assumes that you have familiarity with at least one .NET Framework programming language. Experience with the utilization of .NET components from within programs would be an advantage, but not essential. If you have any questions, please contact us and we will be glad to help you.

# Overview

There are two fundamental parts to the Data Match Enterprise API:

* record indexing
* record matching

These can be utilized in different scenarios:

* single data source matching
* cross data source matching
* data capture incorporating duplicate prevention

## Match Definitions

Match definition is a set of rules we apply on the fields to apply in the matching process.

Match definition for one field consisting of:

Matching type which can be:

* **Fuzzy**
* **Exact**

Before doing any of those two (Fuzzy or Exact) we can transform the input to its phonetic equivalent:

* **Phonetic**

Example: phonetic transformation of words Dayton and Deighton is equal.

If match definition is Fuzzy than we need to apply a value for the:

* **Level**

It defines the threshold for the comparator. If the results of comparison is equal of higher than Level, the match would be considered successful.

## Matching Scores

Matching score is the average value of all matching scores per individual fields. If any field has a matching level below the level the complete score will be 0.

# Classes, Properties, and Methods

The Data Match Enterprise API consists of a number of classes. These classes are listed and described

here along with their properties and/or methods.

Only the most important classes are listed, others will be added later.

|  |  |
| --- | --- |
| *Class* | *Description* |
| **MatchEngine** | Provides the core interface for using and  configuring the Data Match Enterprise API. |
| **MatchDefinitionBuilder** | Contains all settings used by the MatchEngine  class. |
| **MatchDefinitionSingle** | Contains all settings for set of mapped fields |
| **MatchDefinitionsList** | A list of MatchDefinitionSingle |
| **MultipleMatchDefinitionsManager** | More than one MatchDefinitionsList can be used in the matching process and this class contains them |
| **OnDriveTable** | Permanent table used for storage of imported data sets, indexes, temporary and final results of mthe matching proces |
| **IReaderHelper** | Interface used to import/export data from various data sources (SQL Server, mySql, Excel, CSV..) |
| **ReaderConfiguration** | Used to configure the reader |
| **ReaderToVariableTableConvertor** | Converts data from any reader to OnDriveTable for later use in the API |

## MatchEngine Class

Before instance of this class is created the MultipleMatchDefinitionsManager instance must be initiated.

### MatchEngine Methods

* public void **Add**(string mapperName, ITable2CoordsMapper mapper);

**mapperName** is the name for the imported data set which is used in the match definitions, also

**mapper** is the imported data set (can be any class which implements the  ITable2CoordsMapper interface)

* public void **AddPairToMatchList**(int datasourceIndexA, int datasourceIndexB);

add pair of indexes of datasets for matching (e.g. 1, 1 means to match within data set 1 and 2, 3 to match between data sets 2 and 3)

* public bool **DoIndex**()

stores the indexed data into files, for later use with matching process

* public bool **DoMatch**()

does the matching, all settings must be done before we call this method

* public void **ProcessFinalResults(**);

creates finals score pairs and groups

## MatchDefinitionSingle Class

### MatchDefinitionSingle Properties

* Exact
* Fuzzy
* Phonetic
* Level

## see Match Definitions

### MatchDefinitionSingle Methods

* public void **MapField**(string dataSourceName, string fieldName);

for all data sets we want to match we call this method once with the data source name and field name for mapping.

## MultipleMatchDefinitionsManager Class

More than one MatchDefinitionsList can be used in the matching process and this class contains them.

### MultipleMatchDefinitionsManager Methods

* public void **Add**(MatchDefinitionsList matchDefinitionsList);

Adds a MatchDefinitionsList  to the manager

# Example of use

**1. Load existing project**

* This is probably the simplest way to use the API. The project file is created using the Enterprise GUI application and saved.
* That file is loaded in the simple program and matching results are exported

here is the example code:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using dataladder.Data;

using dataladder.Matching;

using dataladder.Licensing;

using dataladder.Matching.Project;

using dataladder.Data.DataTransformation;

namespace UsingTheExistingProject

{

class Program

{

#region Fields

#endregion

static void Main(string[] args)

{

RegistrationWrapper registrationWrapper = new RegistrationWrapper();

registrationWrapper.CustomPathForRegistrationFile = @"C:\Projects\C sharp\Enterprise\EngineAPI\WCFSampleGeneral\registration\";

DateTime expirationTime = RegistrationWrapper.ExpirationDate;

Console.WriteLine("Expiration time: " + expirationTime.ToString());

dataladder.Matching.ApplicationSettings.DataPath = @"e:\enterprise\persistent\";

dataladder.Matching.ApplicationSettings.TempDataPath = @"e:\enterprise\tmp\";

ProjectInfo projectInfo = new ProjectInfo();

projectInfo.Load(@"C:\Users\dule\Documents\DataMatch Enterprise\projects\test 1.dmeproj");

Console.WriteLine("project: \"" + projectInfo.ProjectName + "\" loaded.");

projectInfo.RunTransformation();

Console.WriteLine("data transformed");

projectInfo.MatchEngine.DoIndex();

Console.WriteLine("indexing finished");

projectInfo.MatchEngine.DoMatch();

Console.WriteLine("matching finished");

projectInfo.MatchEngine.ProcessFinalResults();

Console.WriteLine("results processed");

DataSourceInfo dataSourceInfo = projectInfo[0];

//ReaderConfiguration readerConfiguration = dataSourceInfo.Reader.GetConfiguration();

string connectionString = @"Data Source=DULE-I7\SQL2008;Initial Catalog=delme;Integrated Security=True";

SqlDbHelper sqlDbHelper = new SqlDbHelper(connectionString);

string exportTableName = projectInfo.ProjectName + " results";

string schemaName = "dbo";

bool truncate = sqlDbHelper.TableExists(schemaName, exportTableName);

dataladder.Data.IReaderHelper.Export(projectInfo.MatchEngine.FinalScoresGroupsFilteredTable,

sqlDbHelper,

schemaName: "",

tableName: exportTableName,

onTableProgress: null,

bulkCopy: true,

truncate: truncate);

Console.WriteLine("groups exported");

Console.ReadLine();

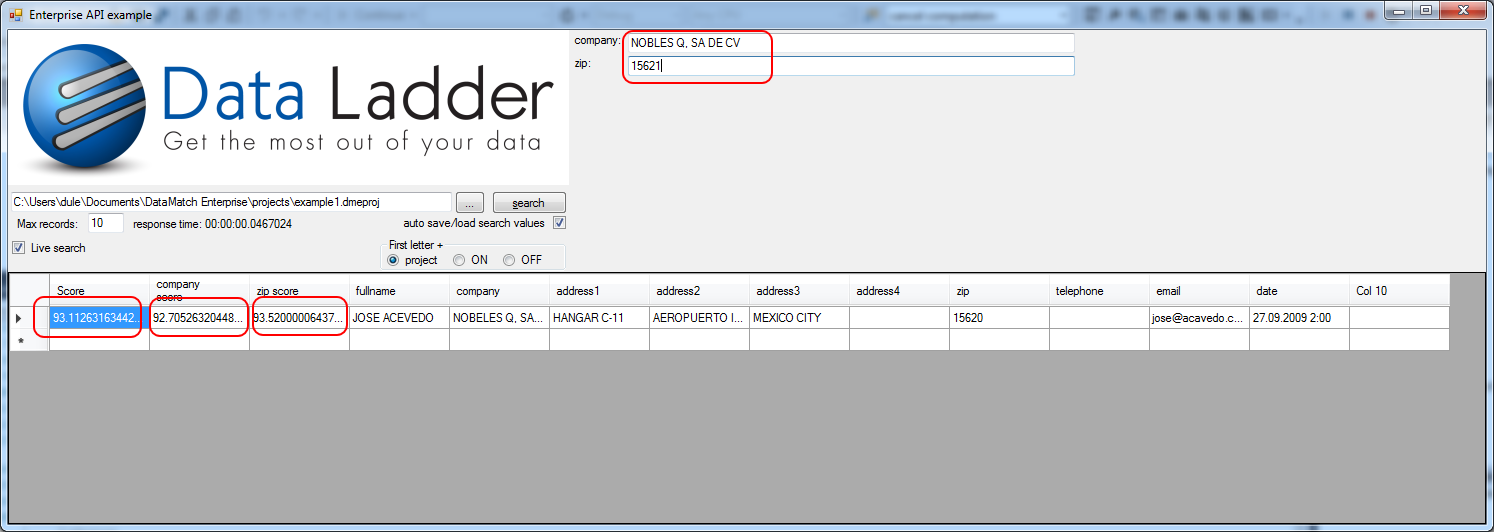
}

}

}

**2. Load existing project and do live search with GUI app**

* Like in the previous example we first create the project using the Enterprise GUI.
* We load the project from the Windows Forms application and do the "live" search:



The code for this example:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using dataladder.Matching;

namespace UsingTheExistingProjectWithGUI

{

public partial class MainForm : Form

{

#region Fields

EngineWrapper engineWrapper = null;

string engineConfigurationIniFileName = "configuration.ini";

string projectsIniFileName = "projects.ini";

string projectName = "";

string appPath;

IniParser projectsParser;

string[] previousSearchValues = new string[0];

bool doTimerSearch = false;

#endregion

#region Constructors

public MainForm()

{

InitializeComponent();

appPath = System.IO.Path.GetDirectoryName(Application.ExecutablePath);

if (!System.IO.File.Exists(projectsIniFileName))

{

using (System.IO.File.Create(projectsIniFileName)) { }

}

string fullIniName = System.IO.Path.Combine(appPath, projectsIniFileName);

projectsParser = new IniParser(fullIniName);

}

#endregion

#region Properties and Fields

bool autoSearchAndLoad { get { return autoSaveCheckBox.Checked; } }

EngineWrapper.AddWeightToFirstLetterActions addWeightToFirstLetterAction

{

get

{

EngineWrapper.AddWeightToFirstLetterActions result;

if (projectRadioButton.Checked)

{

result = EngineWrapper.AddWeightToFirstLetterActions.Project;

}

else if (onRadioButton.Checked)

{

result = EngineWrapper.AddWeightToFirstLetterActions.On;

}

else

{

result = EngineWrapper.AddWeightToFirstLetterActions.Off;

}

return result;

}

}

#endregion

#region Methods

private void openProject()

{

string fullIniName = System.IO.Path.Combine(appPath, engineConfigurationIniFileName);

IniParser parser = new IniParser(fullIniName);

string projectsPath = parser.GetSetting("AppSettings", "projectsPath");

projectOpenFileDialog.InitialDirectory = projectsPath;

if (projectOpenFileDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

string projectFileName = projectOpenFileDialog.FileName;

projectName = System.IO.Path.GetFileNameWithoutExtension(projectFileName);

projectNameTextBox.Text = projectFileName;

if (engineWrapper != null)

{

engineWrapper.Dispose();

engineWrapper = null;

}

engineWrapper = new EngineWrapper(projectFileName);

if (engineWrapper.DataSourceCount != 1)

{

engineWrapper.Dispose();

MessageBox.Show("Only projects with one data source are supported");

searchButton.Enabled = false;

return;

}

searchPanel.Controls.Clear();

const int verticalDist = 23;

const int verticalMargin = 3;

const int horizontalMargin = 3;

int maxLeft = 0;

for (int i = 0; i < engineWrapper.MatchingFieldNames.Length; i++)

{

string fieldName = engineWrapper.MatchingFieldNames[i];

Label label = new Label();

label.Width = 20;

label.AutoSize = true;

label.Name = fieldName;

label.Text = fieldName + ":";

label.Top = i \* verticalDist + verticalMargin;

label.Left = horizontalMargin;

searchPanel.Controls.Add(label);

maxLeft = Math.Max(label.Left + label.Width, maxLeft);

}

maxLeft += horizontalMargin;

for (int i = 0; i < engineWrapper.MatchingFieldNames.Length; i++)

{

string fieldName = engineWrapper.MatchingFieldNames[i];

TextBox textBox = new TextBox();

textBox.Name = fieldName;

textBox.Top = i \* verticalDist + verticalMargin;

textBox.Left = maxLeft;

textBox.Width = searchPanel.Width - maxLeft - horizontalMargin;

if (autoSearchAndLoad)

{

string value = projectsParser.GetSetting(projectName, fieldName);

textBox.Text = value;

}

searchPanel.Controls.Add(textBox);

}

searchButton.Enabled = true;

}

}

string[] getSearchValues()

{

List<string> result = new List<string>();

for (int i = 0; i < searchPanel.Controls.Count; i++)

{

Control control = searchPanel.Controls[i];

if (control is TextBox)

{

TextBox textBox = control as TextBox;

result.Add(textBox.Text.Trim());

}

}

return result.ToArray();

}

private void saveSearchValues(string[] values)

{

if (autoSearchAndLoad)

{

for (int i = 0; i < values.Length; i++)

{

string value = values[i];

projectsParser.AddSetting(projectName, engineWrapper.MatchingFieldNames[i], value);

}

projectsParser.SaveSettings();

}

}

#endregion

#region Events and Handlers

private void openProjectButton\_Click(object sender, EventArgs e)

{

openProject();

}

private void searchButton\_Click(object sender, EventArgs e)

{

search();

}

private void search()

{

if (engineWrapper != null)

{

timeLabel.Text = "...";

Application.DoEvents();

System.Diagnostics.Stopwatch stopwatch = new System.Diagnostics.Stopwatch();

stopwatch.Start();

string[] values = getSearchValues();

int bestMatchesCapacity;

if (!int.TryParse(maxCapacityTextBox.Text, out bestMatchesCapacity))

{

MessageBox.Show("invalid max capacity!");

return;

}

dataladder.Data.OnDriveTable searchResultsTable = engineWrapper.FindMatches(values, bestMatchesCapacity, addWeightToFirstLetterAction);

stopwatch.Stop();

timeLabel.Text = "response time: " + stopwatch.Elapsed.ToString();

searchResultsDataGridView.DataSource = null;

searchResultsDataGridView.DataSource = new dataladder.XtraGridHelper.VirtualListDynamic(searchResultsTable);

saveSearchValues(values);

}

}

#endregion

private void searchTimer\_Tick(object sender, EventArgs e)

{

if (engineWrapper != null)

{

string[] searchValues = getSearchValues();

if (searchValues.Length == previousSearchValues.Length)

{

for (int i = 0; i < searchValues.Length; i++)

{

if (searchValues[i] != previousSearchValues[i])

{

doTimerSearch = true;

}

previousSearchValues[i] = searchValues[i];

}

}

else

{

previousSearchValues = getSearchValues();

}

if (doTimerSearch)

{

if (!engineWrapper.SearchInProgress)

{

search();

doTimerSearch = false;

}

}

}

}

private void liveSearchCheckBox\_CheckedChanged(object sender, EventArgs e)

{

searchTimer.Enabled = liveSearchCheckBox.Checked;

}

}

}

In order to create the GUI example we have used the class EngineWrapper which is not the part of the API. It has some methods which are database specific (insert update and delete records). API requires all data to be loaded and indexed into memory before "live" search can be done. In the case when records are added and the next search needs to take new record into account we needed to introduce the log table. It contains the updated, inserted and deleted records. Becaue it doesn't contain a big number of records it can be usually indexed in sub second time and "live" search can be performed on millions of records + log table, still in sub second response time.

The code of EngineWrapper is another example how API can be used:

using System.Data;

using System.Collections.Generic;

using System;

using System.Text;

using System.Threading;

using System.IO;

using dataladder.Data;

using dataladder.Matching;

using dataladder.Matching.Indexing;

using dataladder.Licensing;

using dataladder.Matching.Project;

using dataladder.Data.DataTransformation;

using dataladder.XtraGridHelper;

namespace SampleServiceNamespace

{

public class EngineWrapper : Object, IDisposable

{

#region Fields

public readonly string TableName;

string projectsPath;

string appDataPath;

string appTempDataPath;

string pKfield;

static string importedDataPath = Directory.GetCurrentDirectory() + @"\data";

static string tempDataPath = Directory.GetCurrentDirectory() + @"\temp data";

static string resultsDataPath = Directory.GetCurrentDirectory() + @"\results";

static string finalResultsFileName = System.IO.Path.Combine(resultsDataPath, "final results.txt");

static string transformedDataSourceAFileName = System.IO.Path.Combine(resultsDataPath, "transformed data source A.txt");

//string connectionString = @"Data Source=DULE-I7\SQL2008;Initial Catalog=snapon;Integrated Security=True";

//string connectionString = @"USER ID=SYSTEM;DATA SOURCE=localhost;PERSIST SECURITY INFO=True;PASSWORD=brbabrba";

//string connectionString = @"USER ID=SYSTEM;DATA SOURCE=192.168.56.1;PERSIST SECURITY INFO=True;PASSWORD=brbabrba";

string connectionString;

string dataSourceNameSingleRow = "single row table";

string dataSourceNameLog;

OnDriveTable singleRowTable = null;

//OnDriveTable importedTableB = null;

OnDriveTable importedLogTable = null;

DataSourceInfo dataSourceInfoSearchRecord;

DataSourceInfo dataSourceInfoLog;

ProjectInfo projectInfo;

MatchEngine matchEngine;

List<int> cachedDatasetIndexes = new List<int>();

bool searchInProgress = false;

DateTime loadTime = DateTime.MinValue;

OnDriveTable resultsTable;

const int firstCol = 11; // eliminating the first columns which are not important in this application

//const int firstCol = 0;

//const string noAction = "X";

const string singleRowOriginal = "O";

public string[] MatchingFieldNames;

public string[] TableFieldNames;

private bool[] tableFieldQuotes;

HashSet<string> quotesHashSet = new HashSet<string>();

bool useLog;

#endregion

#region Constructors

public EngineWrapper(string tableName, bool useLog = false)

{

this.useLog = useLog;

IniParser parser = new IniParser("webservice.ini");

//string connectionString = @"USER ID=SYSTEM;DATA SOURCE=192.168.56.1;PERSIST SECURITY INFO=True;PASSWORD=brbabrba";

connectionString = parser.GetSetting("AppSettings", "connectionstring");

if (string.IsNullOrEmpty(connectionString))

{

throw new Exception("connection string not defined");

}

projectsPath = parser.GetSetting("AppSettings", "projectsPath");

if (string.IsNullOrEmpty(projectsPath))

{

throw new Exception("projects path not defined");

}

appDataPath = parser.GetSetting("AppSettings", "dataPath");

if (string.IsNullOrEmpty(appDataPath))

{

throw new Exception("app data path not defined");

}

appTempDataPath = parser.GetSetting("AppSettings", "tempDataPath");

if (string.IsNullOrEmpty(appTempDataPath))

{

throw new Exception("app temp data path not defined");

}

pKfield = parser.GetSetting("PkFieldName", tableName);

if (useLog)

{

if (string.IsNullOrEmpty(pKfield))

{

throw new Exception("pKfield not defined");

}

}

string pathForRegistrationFile = parser.GetSetting("AppSettings", "pathForRegistrationFile");

this.TableName = tableName;

dataSourceNameSingleRow += tableName;

dataSourceNameLog = tableName + "Log";

RegistrationWrapper registrationWrapper = new RegistrationWrapper();

registrationWrapper.CustomPathForRegistrationFile = pathForRegistrationFile;

DateTime expirationTime = RegistrationWrapper.ExpirationDate;

removeRemainedFiles();

string error;

loadData(out error);

}

#endregion

#region Methods

/// <summary>

/// for now it assumes all fields are text type

/// </summary>

/// <param name="hash"></param>

/// <param name="tableName"></param>

/// <param name="values"></param>

/// <param name="fieldNames"></param>

/// <param name="error"></param>

/// <returns></returns>

public bool InsertRecord(string tableName, string[] values, out string error)

{

bool result = false;

error = "";

try

{

StringBuilder sb = new StringBuilder();

sb.Append("insert into ");

sb.AppendLine(tableName);

sb.AppendLine("(");

for (int i = 0; i < TableFieldNames.Length - 1; i++) // last field is ID, doesn't used in insert operation

{

sb.Append(TableFieldNames[i]);

if (i < TableFieldNames.Length - 2)

{

sb.AppendLine(",");

}

else

{

sb.AppendLine("");

}

}

sb.AppendLine(")");

sb.AppendLine("values");

sb.AppendLine("(");

for (int i = 0; i < TableFieldNames.Length - 1; i++)

{

if (tableFieldQuotes[i])

{

sb.Append("'");

}

sb.Append(values[i]);

if (tableFieldQuotes[i])

{

sb.Append("'");

}

if (i < TableFieldNames.Length - 2)

{

sb.AppendLine(",");

}

else

{

sb.AppendLine("");

}

}

sb.AppendLine(")");

executeCmd(sb.ToString());

}

catch (Exception ex)

{

error = ex.Message;

}

return result;

}

public bool UpdateRecord(string tableName, string[] values, string id, out string error)

{

bool result = false;

error = "";

try

{

StringBuilder sb = new StringBuilder();

sb.Append("update ");

sb.AppendLine(tableName);

sb.Append(" set ");

for (int i = 0; i < TableFieldNames.Length - 1; i++) // last field is ID, doesn't used in insert operation

{

sb.Append(TableFieldNames[i]);

sb.Append(" = ");

if (tableFieldQuotes[i])

{

sb.Append("'");

}

sb.Append(values[i]);

if (tableFieldQuotes[i])

{

sb.Append("'");

}

if (i < TableFieldNames.Length - 2)

{

sb.AppendLine(",");

}

else

{

sb.AppendLine("");

}

}

sb.Append("where " + pKfield + " = ");

if (quotesHashSet.Contains(id.ToLower()))

{

sb.Append("'");

}

sb.Append(id);

if (quotesHashSet.Contains(id.ToLower()))

{

sb.Append("'");

}

executeCmd(sb.ToString());

}

catch (Exception ex)

{

error = ex.Message;

}

return result;

}

public bool DeleteRecord(string tableName, string id, out string error)

{

bool result = false;

error = "";

try

{

StringBuilder sb = new StringBuilder();

sb.Append("delete from ");

sb.AppendLine(tableName);

sb.AppendLine("where " + pKfield + " = ");

if (quotesHashSet.Contains(id.ToLower()))

{

sb.Append("'");

}

sb.Append(id);

if (quotesHashSet.Contains(id.ToLower()))

{

sb.Append("'");

}

executeCmd(sb.ToString());

}

catch (Exception ex)

{

error = ex.Message;

}

return result;

}

private void executeCmd(string cmd)

{

SqlDbHelper dbHelper = new SqlDbHelper(connectionString);

dbHelper.Connect();

dbHelper.ExecuteCmd(cmd);

dbHelper.Disconnect();

}

static OnDriveTable getImportedTableFromSQL(string connectionString, string cmd, string dataSourceName)

{

SqlDbHelper reader = new SqlDbHelper(connectionString);

ReaderConfiguration readerConfiguration = reader.GetConfiguration();

readerConfiguration.SelectCmd = cmd;

reader.SetConfiguration(readerConfiguration);

reader.ReadTable(readerConfiguration, true);

OnDriveTable result = ReaderToVariableTableConvertor.Copy(reader, importedDataPath, dataSourceName);

return result;

}

//static OnDriveTable getImportedTableFromOracle(string connectionString, string cmd, string dataSourceName)

//{

// SqlDbHelper sqlDbHelper = new SqlDbHelper();

// OracleDbHelper oracleDbHelper = new OracleDbHelper(connectionString);

// ReaderConfiguration readerConfiguration = oracleDbHelper.GetConfiguration();

// readerConfiguration.SelectCmd = cmd;

// oracleDbHelper.SetConfiguration(readerConfiguration);

// oracleDbHelper.ReadTable(readerConfiguration, true);

// OnDriveTable result = ReaderToVariableTableConvertor.Copy(oracleDbHelper, importedDataPath, dataSourceName);

// return result;

//}

private static void removeRemainedFiles()

{

dataladder.IO.IOHelper.RemoveAllFiles(importedDataPath);

dataladder.IO.IOHelper.RemoveAllFiles(tempDataPath);

dataladder.IO.IOHelper.RemoveAllFiles(resultsDataPath);

}

public OnDriveTable FindMatches(string tableName, string[] values, int records)

{

lock (this)

{

while (searchInProgress)

{

Thread.Sleep(10);

}

searchInProgress = true;

try

{

System.Diagnostics.Stopwatch stopwatch = new System.Diagnostics.Stopwatch();

stopwatch.Start();

dataSourceInfoSearchRecord.InputTable.MakeWritable();

for (int i = 0; i < values.Length; i++)

{

dataSourceInfoSearchRecord.InputTable.SetData(values[i], 0, MatchingFieldNames[i]);

}

matchEngine.ReindexSingleDataSource(1);

if (useLog)

{

importedLogTable = importLogTable();

dataSourceInfoLog.InputTable.ReplaceWith(importedLogTable);

dataSourceInfoLog.InputTable.MakeReadOnlyShareable();

matchEngine.ReindexSingleDataSource(2);

}

matchEngine.LoadAllInMemory = true;

matchEngine.DoMatch(records, clearAllAfterMatching: false);

matchEngine.ProcessFinalResults(clearAllAfterMatching: false);

unifyResults(matchEngine.FinalScoresGroupsFilteredTable);

stopwatch.Stop();

}

finally

{

searchInProgress = false;

}

}

return resultsTable;

}

private MatchEngine initializeMatchingEngine()

{

dataladder.Matching.ApplicationSettings.DataPath = appDataPath + @"\";

dataladder.Matching.ApplicationSettings.TempDataPath = appTempDataPath + @"\";

ProjectInfo projectInfo = new ProjectInfo();

projectInfo.Load(projectsPath + @"\" + TableName + ".dmeproj");

projectInfo[0].Refresh(projectInfo.DataPath, Path.Combine(projectInfo.DataPath, "tmp"), onTableCopyProgress: null, synchronizationContext: null);

projectInfo[0].CreateDiagram();

projectInfo[0].RunTransformation();

dataSourceInfoSearchRecord = new DataSourceInfo(null);

dataSourceInfoLog = new DataSourceInfo(null);

if (singleRowTable != null)

{

singleRowTable.Dispose();

}

singleRowTable = createSingleRowTableAndDetermineFieldNames(projectInfo);

for (int colIndex = 0; colIndex < MatchingFieldNames.Length; colIndex++)

{

string fieldName = MatchingFieldNames[colIndex];

singleRowTable.SetData("zzzzzzzzz", 0, fieldName);

}

if (useLog)

{

singleRowTable.SetData(singleRowOriginal, 0, "ACTION");

}

dataSourceInfoSearchRecord.InputTable = singleRowTable;

projectInfo.Add(dataSourceInfoSearchRecord);

if (useLog)

{

dataSourceInfoLog.InputTable = importedLogTable;

projectInfo.Add(dataSourceInfoLog);

}

MatchDefinitionBuilder matchDefinitionBuilder = projectInfo.MatchDefinitionBuilder;

// engine is prepared to support multiple match definitions, here we work only with one

MultipleMatchDefinitionsManager multipleMatchDefinitionsManager = matchDefinitionBuilder.MultipleMatchDefinitionsManager;

for (int i = 0; i < multipleMatchDefinitionsManager.Count; i++)

{

MatchDefinitionsList matchDefinitionsList = multipleMatchDefinitionsManager[i];

for (int singleDefinitionIndex = 0; singleDefinitionIndex < matchDefinitionsList.Count; singleDefinitionIndex++)

{

MatchDefinitionSingle matchDefinitionSingle = matchDefinitionsList[singleDefinitionIndex];

bool found;

string fieldName = matchDefinitionSingle.GetMappedFieldName(TableName, out found);

if (found) // in case of multiple definitions it could be found == false

{

matchDefinitionSingle.MapField(dataSourceNameSingleRow, fieldName);

if (useLog)

{

matchDefinitionSingle.MapField(dataSourceNameLog, fieldName);

}

}

}

}

//groupFields();

// initializing match engine

//MatchEngine matchEngine = new MatchEngine(multipleMatchDefinitionsManager, tempDataPath, resultsDataPath);

MatchEngine matchEngine = projectInfo.MatchEngine;

matchEngine.Add(dataSourceNameSingleRow, singleRowTable);

if (useLog)

{

matchEngine.Add(dataSourceNameLog, importedLogTable);

}

matchDefinitionBuilder.AvailableFields.InitialMapping();

matchEngine.ClearPairToMatchList(); // they are loaded with the project...

matchEngine.DoIndex();

matchEngine.AddPairToMatchList(1, 0); // to find matches between data sources A and B

if (useLog)

{

matchEngine.AddPairToMatchList(1, 2); // to find matches between data sources B and C

}

cachedDatasetIndexes.Clear();

cachedDatasetIndexes.Add(0);

matchEngine.SetCachedDataSources(cachedDatasetIndexes);

matchEngine.LoadAllInMemory = true;

matchEngine.DoMatch(clearAllAfterMatching: false);

matchEngine.ProcessFinalResults(clearAllAfterMatching: false);

List<string> tmp = new List<string>();

TableFieldNames = new string[matchDefinitionBuilder.AvailableFields.MappedFieldsRowList.Count];

for (int i = 0; i < matchDefinitionBuilder.AvailableFields.MappedFieldsRowList.Count; i++)

{

MappedFieldsRow mappedFieldsRow = matchDefinitionBuilder.AvailableFields.MappedFieldsRowList[i];

FieldMapInfo fieldMapInfo = mappedFieldsRow[this.TableName];

if (fieldMapInfo != null)

{

string fieldName = fieldMapInfo.FieldName.Trim();

if (!string.IsNullOrEmpty(fieldName))

{

tmp.Add(fieldName);

}

}

}

TableFieldNames = tmp.ToArray();

getSqlFieldTypes();

return matchEngine;

}

private void getSqlFieldTypes()

{

tableFieldQuotes = new bool[TableFieldNames.Length];

quotesHashSet = new HashSet<string>();

string cmd = "select t.name as typename, c.name as columnname from syscolumns c\r\n" +

"left join sysobjects o on (c.id = o.id) left join systypes t on (c.xtype = t.xtype)\r\n" +

"where t.status = 0 and o.name = '" + this.TableName + "'";

OnDriveTable typesTable = getImportedTableFromSQL(connectionString, cmd, "types");

for (int rowIndex = 0; rowIndex < typesTable.RecordCount; rowIndex++)

{

string typeName = typesTable.GetData(rowIndex, 0).ToString().ToLower();

if (typeName.Contains("char") || typeName.Contains("date"))

{

string columnName = typesTable.GetData(rowIndex, 1).ToString().ToLower();

quotesHashSet.Add(columnName);

}

}

for (int i = 0; i < TableFieldNames.Length; i++)

{

string fieldName = TableFieldNames[i].ToLower();

tableFieldQuotes[i] = quotesHashSet.Contains(fieldName);

}

}

private OnDriveTable createSingleRowTableAndDetermineFieldNames(ProjectInfo projectInfo)

{

OnDriveTable result = new OnDriveTable(importedDataPath, dataSourceNameSingleRow, toDeleteExisting: true);

List<string> allFieldNames = determineFieldNames(projectInfo);

int fieldsCount = allFieldNames.Count;

MatchingFieldNames = new string[fieldsCount];

for (int i = 0; i < allFieldNames.Count; i++)

{

string fieldName = allFieldNames[i];

MatchingFieldNames[i] = fieldName;

result.AddField(fieldName, typeof(string));

}

if (useLog)

{

result.AddField("ACTION", typeof(string));

}

return result;

}

private static List<string> determineFieldNames(ProjectInfo projectInfo)

{

DataSourceInfo originalDataSourceInfo = projectInfo[0];

string tableName = originalDataSourceInfo.Name;

TransformationDiagram transformationDiagram = originalDataSourceInfo.DiagramGenerator.DiagramVariableTableHelper.FirstTransformationDiagram;

List<string> tmp = new List<string>();

MultipleMatchDefinitionsManager multipleMatchDefinitionsManager = projectInfo.MatchDefinitionBuilder.MultipleMatchDefinitionsManager;

for (int matchDefinitionIndex = 0; matchDefinitionIndex < multipleMatchDefinitionsManager.Count; matchDefinitionIndex++)

{

List<MatchDefinitionMappedToField> matchDefinitionMappedToFieldList = multipleMatchDefinitionsManager[matchDefinitionIndex].DetermineFieldsToIndex(originalDataSourceInfo.Name, originalDataSourceInfo.TransformedValuesTable);

for (int i = 0; i < matchDefinitionMappedToFieldList.Count; i++)

{

MatchDefinitionMappedToField matchDefinitionMappedToField = matchDefinitionMappedToFieldList[i];

string fieldName = matchDefinitionMappedToField.FieldName;

DataFlow output = transformationDiagram.Outputs[fieldName];

if (output != null)

{

for (int ii = 0; ii < transformationDiagram.Inputs.Count; ii++)

{

DataFlow input = transformationDiagram.Inputs[ii];

if ((output.IsAfter(input)) | (input == output))

{

if (!tmp.Contains(input.Name))

{

tmp.Add(input.Name);

}

}

}

}

}

}

return tmp;

}

private void unifyResults(ITable2CoordsMapper combinedResults)

{

if (resultsTable != null)

{

resultsTable.ToDeleteFilesAfterClosing = true;

resultsTable.Dispose();

}

resultsTable = new OnDriveTable(importedDataPath, this.matchEngine.Name + " results", toDeleteExisting: true);

int count = useLog ? combinedResults.ColumnCount - 2 : combinedResults.ColumnCount;

for (int colIndex = firstCol; colIndex < count; colIndex++) // without action and last update fields

{

string fieldName = combinedResults.GetColumnName(colIndex);

resultsTable.AddField(fieldName);

}

string id;

string action;

Dictionary<string, string> logRecords = new Dictionary<string, string>();

if (useLog)

{

for (int rowIndex = 0; rowIndex < dataSourceInfoLog.InputTable.RecordCount; rowIndex++) // record 0 is the lookup record itself...

{

id = dataSourceInfoLog.InputTable.GetData(rowIndex, pKfield).ToString();

action = (string)dataSourceInfoLog.InputTable.GetData(rowIndex, "ACTION");

logRecords.Add(id, action);

}

}

for (int rowIndex = 0; rowIndex < combinedResults.RecordCount; rowIndex++)

{

int dataSourceIndex = (int)combinedResults.GetData(rowIndex, (int)MatchEngine.PreviewFinalResultsStaticFields.DataSource);

//if (action == singleRowOriginal) // this is the lookup record itself...

if (dataSourceIndex == 1)

{

continue;

}

if (useLog)

{

id = combinedResults.GetData(rowIndex, pKfield).ToString().Trim();

action = combinedResults.GetData(rowIndex, "ACTION").ToString();

if (!string.IsNullOrEmpty(action)) // for log records

{

if (action == "D") // record is from log, record deleted, ignore it...

{

continue;

}

else

{

copyResultRecord(combinedResults, rowIndex);

}

}

else // not a log record

{

id = combinedResults.GetData(rowIndex, pKfield).ToString();

if (logRecords.TryGetValue(id, out action)) // original record, changed ignore it...

{

continue;

}

else

{

copyResultRecord(combinedResults, rowIndex);

}

}

}

else

{

copyResultRecord(combinedResults, rowIndex);

}

}

}

private void copyResultRecord(ITable2CoordsMapper combinedResults, int rowIndex)

{

int newRowIndex = resultsTable.RecordCount;

int count = useLog ? combinedResults.ColumnCount - 2 : combinedResults.ColumnCount;

for (int colIndex = firstCol; colIndex < count; colIndex++) // without action and last update fields

{

object obj = combinedResults.GetData(rowIndex, colIndex);

resultsTable.SetData(obj, newRowIndex, colIndex - firstCol);

}

}

private void loadData(out string error)

{

error = "";

System.Diagnostics.Stopwatch stopwatch = new System.Diagnostics.Stopwatch();

stopwatch.Start();

projectInfo = new ProjectInfo();

if (matchEngine != null)

{

matchEngine.Dispose();

matchEngine = null;

}

loadTime = DateTime.Now;

if (useLog)

{

importedLogTable = importLogTable();

}

matchEngine = initializeMatchingEngine();

stopwatch.Stop();

}

//private void groupFields()

//{

// MatchDefinitionsList matchDefinitionsList = this.projectInfo.MatchDefinitionBuilder.MultipleMatchDefinitionsManager[0];

// for (int i = 0; i < matchDefinitionsList.Count; i++)

// {

// MatchDefinitionSingle matchDefinitionSingle = matchDefinitionsList[i];

// if (false) // for now...

// {

// matchDefinitionSingle.GroupId = 0;

// matchDefinitionSingle.GroupLevel = 0.60f;

// matchDefinitionSingle.MaxEmptyWeightBelow = 200;

// matchDefinitionSingle.MaxMismatchWeightBelow = 200;

// matchDefinitionSingle.MaxTotalWeightBelow = 200;

// }

// else

// {

// matchDefinitionSingle.GroupId = -1;

// }

// }

//}

private OnDriveTable importLogTable()

{

string cmd = "select l.\* from " + TableName + "log l " +

"right join (select id, max(lastupdate) as lastupdate from " + TableName + "log group by id) l2 on l.id = l2.id and l.lastupdate = l2.lastupdate " +

"where l.lastupdate > " +

"'" + SqlDbHelper.DateTimeToString(loadTime) + "'";

// OracleDbHelper.DateTimeToString(loadTime);

OnDriveTable result = getImportedTableFromSQL(connectionString, cmd, dataSourceNameLog);

return result;

}

#endregion

#region IDisposable members

public void Dispose()

{

if (singleRowTable != null)

{

singleRowTable.Dispose();

singleRowTable = null;

}

if (importedLogTable != null)

{

importedLogTable.Dispose();

importedLogTable = null;

}

//if (importedTableB != null)

//{

// importedTableB.Dispose();

// importedTableB = null;

//}

this.matchEngine.Dispose();

}

#endregion

}

}