DataMatch Enterprise

Copyright Data Ladder©  2001 – 2017. All Rights Reserved.

Table of Contents:

[**1.** **What is DataMatch Enterprise?** 2](#_Toc493235887)

[1.1 Who can need DME? FromAbove 2](#_Toc493235888)

[**2. Getting started with DataMatch Enterprise** 2](#_Toc493235889)

[2.1 Localization 3](#_Toc493235890)

[2.2.System requirements 3](#_Toc493235891)

[2.3 General Setup Tasks 3](#_Toc493235892)

[2.4 DataMatch Version Differences 5](#_Toc493235893)

[2.5 Data sources supportability 5](#_Toc493235894)

[2.6 Project 6](#_Toc493235895)

[**3. Data operations** 6](#_Toc493235896)

[3.1 Address Verification 6](#_Toc493235897)

[3.2 Data Cleansing and Standardization 7](#_Toc493235898)

[3.3 Matching process 7](#_Toc493235899)

[3.4 Matching Algorithms 8](#_Toc493235900)

[**4. DataMatch Enterprise API** 8](#_Toc493235901)

[4.1 How DataMatch Enterprise API can be useful 8](#_Toc493235902)

[4.2 What our API offers 10](#_Toc493235903)

[4.3 Examples of Use 11](#_Toc493235904)

[**5. Benefits of choice DataMatch Enterprise** 18](#_Toc493235905)

[**6.  Certifications and Licenses** 19](#_Toc493235906)

# **What is DataMatch Enterprise?**

DataMatch Enterprise – Is a high-performance desktop platform used to manipulate variable volumes of data, allowance for the consistent configuration of processing parameters at each stage, and monitor their transformation at each iteration. It is based on a unique software core that allows high-precision cleansing, standardization and deduplication of large data volumes while guaranteeing maximum data security.

## Who can need DME? FromAbove

DataMatch Enterprise - is a platform for companies who work with varied sizes of databases: contacts, products, addresses, reports, specifications, etc. DataMatch Enterprise allows the user to work simultaneously with several data sources of various types. Perform calculations, find and link customer data, consolidate data across multiple sources, and remove unwanted records, with possible transformations or replacements. This significantly reduces the processing and preparation time for the matching and deduplication process.

DataMatch Enterprise can be used for formatting and cleaning of data which is provided by a wide range of operations with content, user able to work with the data at each stage of process with the ability to save and export transformed data sources without the risk of content lost.

# **2. Getting started with DataMatch Enterprise**

## 2.1 Localization

Languages: English, Spanish

## 2.2.System requirements

Minimum requirements:

* Processor: Dual Core
* RAM: 4GB
* Operating system: 32-bit Microsoft Windows 7 SP 1

Recommended requirements:

* Processor: Core 2 Quad (i5/i7, AMD A10)  or higher
* RAM: 16GB or higher
* Operating system: 64-bit Microsoft Windows 7 or higher
* SSD or SAS

## 2.3 General Setup Tasks

1. Unpack archive WebServiceExample.zip in the folder. The location is by your choice.

2. You will find WCFHostingSample.sln Visual Studio solution in the folder. After you open it with Visual Studio (recommended 2015 and above versions) you can build the whole solution or individual projects.

3. Some of the projects contain \*.ini files. In order to run those projects you will need to edit \*.ini file(s)

In the SelfHostConsole\bin\Debug project there is a webservice.ini file. It contains few mandatory settings:

Example:

[AppSettings] (except connection string all other values are mandatory)

connectionstring=Server=[SERVER\_NAME];Database=[DATABASE\_NAME];Trusted\_Connection=True;

pathForRegistrationFile=C:\ProgramData\DataMatch Enterprise\Registration\

projectsPath=c:\Users\[USER\_NAME]\Documents\DataMatch Enterprise\projects

dataPath=D:\enterprise\API persistent

tempDataPath=D:\enterprise\tmp\Data Ladder

[PkFieldName] (mandatory if insert/update/delete is performed on the database table)

Example1=ID

Companies1M=ID

s2tech=SSN

SAP=ID

[RAM MB per project] (mandatory for all DME projects used by service, integer value is in kB. If the free memory on the system is less than that value service will return a message about it and will continue working after the memmory is available again)

business\_names=1024

person\_names=2048

example1=4096

SAP=4096

Companies 100k=4096

Companies 100k 2=4096

[RAM MB] (both mandatory)

AllInMemory=true // for now should always be true

minFree=10480 // do not start if this amount in kB is not available.

License

License key generates for each independent user and has relation to machine installed on. It can be provided via email or by DataMatch Enterprise support team.

User passes the license key in Registration window and can work with application

## 2.4 DataMatch Version Differences

DataMatch Enterprise Standard:

* Address Verification disabled

DataMatch Enterprise with Address Verification:

* Includes Address Verification functionality.
* СASS - Additional libraries for the Address Verification module.

DataMatch Enterprise Trial – Unregistered Trial Version for 30 Day:

* Export disabled
* Address Verification disabled
* Scheduler disabled (Automatic start of the process according to the specified criteria)
* Import - limit on 1 million records

## 2.5 Data sources supportability

* SQL Server
* Oracle
* DBF
* MySQL
* Ole DB
* ODBC
* DB2
* SalesForce
* Fixed Width Text File
* Exсel
* Text Delimited

## 2.6 Project

DataMatch Enterprise allows the user to create projects with multiple data sources and save them for future use.  Configurations,various stages of content transformation, comparison results may be accessed when the user opens an existing project. Users can switch between projects which greatly improves the user experience and the number of possible solutions.

In registered versions, an optional scheduler is available in order to automatically run projects with the specified configurations and subsequent export.

# **3. Data operations**

## 3.1 Address Verification

This functionality allows the user to verify addresses by leveraging the United States Postal Service database as well as Canadian Post databases. This function becomes available when the user purchases  the version with address verification as well as installing of additional libraries.

## 3.2 Data Cleansing and Standardization

SDK allows the user to cleanup input data and transform it to uniform view. It is possible to apply simple transformations:

* Removing of trailing and leading spaces.
* Replace and remove unwanted  characters.
* Casing  to upper or lower case, and complex converting.
* Parse input data using regular expressions (inputting expression directly or using the Pattern Builder™).
* Perform word replacement using proprietary development WordSmith™.

Also DataMatch Enterprise allows the user  to merge several columns of tabular data into the new field and define special types for columns like Name, Address, Company name, etc. After defining such columns extra fields are usually created.

Transformed data is stored in separate tables on local hard drive and then is used for the additional  steps of data processing in DataMatch Enterprise or DataMatch Enterprise API.

## 3.3 Matching process

The principle function of DataMatch Enterprise, and DataMatch Enterprise SDK, is the Match Engine. It allows the user to perform a  cluster analysis of the large amounts of data using defined criteria.

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are  similar (in some sense or another) to each other. This process’ main task is exploratory data mining which is a common technique for statistical data analysis, machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

## 3.4 Matching Algorithms

DataMatch Enterprise API uses several different algorithms for clustering (matching) data:

* Exact - combines records into the same group only if the values of matched cells of a table are equivalent.
* Fuzzy - cell values should be similar by defined percent of similarity.
* Numeric - this algorithm can be used for numeric data only, numbers should be similar according to determined tolerance (percentage).

The results of matching are presented in a table containing grouped results by various criteria with the ability to replace, merge, and find values based desired cell content, Also, a  summary report is created which includes statistical information about the match event

DataMatch Enterprise allows the user to export content at any stage of the project.  This means that the user may export cleansed data or verified addressed without having to run a match event. The data can be exported  the file format desired..

# **4. DataMatch Enterprise API**

## 4.1 How DataMatch Enterprise API can be useful

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.

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.

|  |  |
| --- | --- |
| *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 |

In DataMatch Enterprise the user is offered a large range of options and functionality for processing large amounts of data in the shortest time possible, however, the API allows   developers to implement this powerful match functionality into their own applications in order to achieve a real time solution. For these purposes SDK includes API which allows developers to work with DataMatch Enterprise projects, perform data transformation, cleansing and logically group similar data sets. The user can import processed data into different formats and types of data sources.

## 4.2 What our API offers

API includes several Visual Studio projects with API main functions demonstration and code examples:

* WCF service for API. It allows developers to create client-server applications both for Desktop and Web.
* Desktop application that allows to search some data in data sources of loaded DataMatch Enterprise project.
* Sources can be transformed.
* Application that perform matching of DataMatch Enterprise project data in automated mode.

DataMatch Enterprise API exposes for developers most of functions that available in the Data Match Enterprise application. It’s possible to:

* + Import some DataMatch Enterprise project.
  + Perform transformation.
  + Export transformed data.
  + Perform Clustering (Matching) of the data.
  + Export matched groups and pairs of source rows.
  + Analyze quality of the data (Create data profile).

## 4.3 Examples of Use

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

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();

}

}

}

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.

# **5. Benefits of choice DataMatch Enterprise**

The main advantages of DataMatch Enterprise is the speed and accuracy of the enterprise level beating IBM and SAS, ability to track the process within your project, specify additional configurations, export and import data.

DataMatch Enterprise provides more than a dozen different sources for importing and exporting data, including Big Data and a large number Supported formats. The

Quick Data Profile tool fixes errors in the content of your data in less than 5 minutes of installation.

Scheduler tool allows to create tasks for automatically start your project at a specified time.

Unique Matching Algorithms:

On the low-level the simple string metrics algorithms with the speed optimization are used for text matching and special mathematical calculations are used for the numeric matching.

Special preliminary filtration is used for knowingly different items that does not satisfy match criteria.

Proprietary algorithms are used that allows to work with huge amount of data that is stored on HDD. Fast caching approaches are involved that provides excellent speed for small and medium data sources and acceptable match time for extremely huge amount of data (50 millions + of records, Gigabytes on HDD).

# **6.  Certifications and Licenses**

**Prerequisites**

Prerequisites should be verified before using the SDK.

**License**

Each copy of Data Match Enterprise should have a license key. The license key is a multi-character string that user obtains from Data Match Enterprise support team. Support team provides the license key as a string, created in additional key generation program.

Note: DataMatch Enterprise is available in short version for 30 days without registration key

Setting the Environment

Compiling Sample Code

SDK Missing Items

* Reference to sample Project
* Discuss Field Types and locations
* Use Cases
* API Architecture (perhaps flow chart?)
* Interfaces/Data Types we accept
* Bill of materials (files that you need to download)
* Prerequisites
* General setup tasks
  + License
  + Setup environment
  + Compiling sample code
* Sample code for testing
* Glossary/Index