

Data Match Enterprise API – DME Classes



DME Version 3.3.11

Contents

IReader Interface	5
ReaderToVariableTableConvertor Class	6
ReaderConfiguration Class.....	8
SourceDb Enum.....	9
Field Class.....	11
Fields Class	12
IContainer2CoordsMapper Class	12
OnDriveTable Class	14
Main members.....	15
MatchEngine Class	18
Main members.....	19
MultipleMatchDefinitionsManager Class	22
MatchCriteriaList Class.....	24
Main members.....	24
MatchCriteria Class	26
AvailableFields Class	29
AvailableFieldsFromOneTable Class	31
MappedFieldsRow Class	33
FieldMapInfo Class.....	34
ProjectSpec Class	36
ProjectSpecHelper Class.....	39
ColumnTransformationsSpec Class.....	40
ColumnTransformationSpec Class	41
MergeInfoSpec Class.....	43
MergedFieldSpec Class	45
MatchDefinitionSpec Class	45
DefinitionsSpec Class	46
DefinitionSpec Class.....	47
MatchCriteriaSpec Class	47
MappedFieldsSpec Class	50
MappedFieldSpec Class.....	50

RegistrationWrapper Class	52
IOHelper Class	53
TransformationDiagram Class	54
DataFlow Class	56
DataFlow.InputTypes Enum	58
DataFlowCollection Class	59
TransformationBlock Class	60
CassInputTypes Enum	62
CassOutputParts Enum	63
TransformationTypes Enum	66
IColumnTransformation Interface	68
ColumnTransformation Class	70
ColumnTransformationList Class	72
FirstNameBlock Class	74
FullNameBlock Class	75
CleanBlock Class	76
CopyColumnBlock Class	77
RegexBlock Class	78
RegexBlockNative Class	79
RegexBlockAdvanced Class	80
RegexBlockFactory Class	81
MergeBlock Class	82
MergeBlockSingleStorage Class	83
MergeBlocksStorage Class	84
WordSmithBlock Class	86
AddressBlock Class	88
AddressCassBlock Class	89
AddressVerificationSettings Class	90
CassAddress Class	92
CassGeoCoder Class	97
CassParser Class	98
ICassRequest Interface	99
ICassResponse Interface	100

IAddressRequest Interface.....	101
AddressRequest Class	102
IAddressResponse Interface	103
AddressResponse Class	104
CassManagerFactory Class.....	106
ICassManager Interface	107
CassManager Class.....	108

IReader Interface

Namespace:	dataladder.Data
Assemblies:	DataMatch.Connectors.dll

Provides base methods to work with data sources. Extends the set defined in the ISimpleReader interface.

C#
public interface IReader : ISimpleReader

Remarks

This interface allows to manipulate different types of data sources in the same manner. Methods described below use widely the ReaderConfiguration class, this class is described in its own section and its appointment to keep settings of data source.

Methods:

GetConfiguration()	Gets the reader's configuration.
SetConfiguration(ReaderConfiguration)	Sets the reader's configuration.
ReadTable(ReaderConfiguration configuration , Boolean toDetermineFields)	Prepares for reading the data from data source. Opens connections, creates necessary structures, etc. configuration - Settings that contain information about data source, toDetermineFields - If true then information about columns will be refreshed from data source during reading. If false, column information will be given from ReaderConfiguration settings returns - true if successful
GetTables()	Gets the source's table list in a tabular format.
CreateTable(String, String, List<Field>, Boolean, Boolean)	Creates a table in the data source with the characteristics specified.
TruncateTable()	Truncates the table by its full name specified.
InsertValues(String, List<Object>, List<Field>)	Inserts values into the table specifying corresponding fields.
Connect(out String)	Tries to connect to the source.
Disconnect()	Releases resources (closes connections to DB, flushes files, closes file streams, etc.)

ReaderToVariableTableConvertor Class

Namespace:	dataladder.Data
Assemblies:	DataMatch.Api.dll

This class used to copy data from ISimpleReader to a table placed on hard disk or in memory.

C#
public class ReaderToVariableTableConvertor

Examples

The example shows importing from SQL Server.

```
DbHelper reader = new SqlDbHelper (connString: @"Data Source=localhost\SQLEXPRESS;
    Initial Catalog=API; Integrated Security=True");

ReaderConfiguration readerConfiguration = reader.GetConfiguration();
readerConfiguration.SelectCommand = "select top(20) * from example1";
reader.SetConfiguration(readerConfiguration);
reader.ReadTable(readerConfiguration, true);

//read first 20 rows from SQL Server table
//save them on disk in OnDriveTable storage
var helper = new ReaderToVariableTableConvertor();
OnDriveTable onDrive = helper.Copy(reader, @"D:\API\data\folderForExample1",
    "example1", out string error, new System.Threading.CancellationToken());

var path = onDrive.FilesPath;
var baseName = onDrive.FileNameBase;

//close OnDriveTable storage and release resources
onDrive.Dispose();
onDrive = null;

using (var storage = new OnDriveTable(path, baseName))
{
    int rows = storage.RecordCount;
    int cols = storage.ColumnCount;
    string[] colNames = storage.GetColumnNames();
}
```

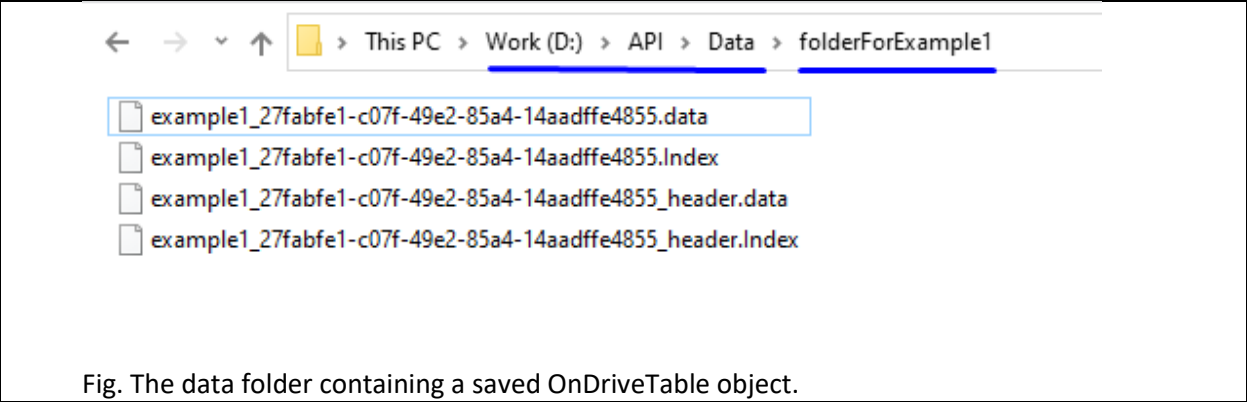


Fig. The data folder containing a saved OnDriveTable object.

Remarks

This class contains a single static method Copy() that loads the data from any data source into the internal DME format.

Methods

<code>Copy(IReader <i>reader</i>, String <i>path</i>, String <i>name</i>, out String <i>error</i>, CancellationToken <i>cancel</i>, ProgressDelegate <i>progress</i> = null, Int32 <i>toRow</i> = Int32.MaxValue, String <i>fileNameBase</i> = null, OperationModes <i>operationMode</i> = OperationModes.Disk, MinSpaceReachedDelegate <i>onMinSpaceReached</i> = null)</code>	<p>Loads data from a data source which is wrapped by IReader into a file storage (OnDriveTable).</p> <p><i>reader</i> - IReader that can wrap various data source types, <i>path</i> - folder to store data in, <i>name</i> - base part of the name of the storage file (a GUID value is appended to avoid ambiguity if <i>fileNameBase</i> isn't specified), <i>error</i> - message containing any errors, <i>cancel</i> - cancellation token for canceling the process, <i>progress</i> - delegate for showing the import progress, <i>toRow</i> – upper limit of row number to import, <i>fileNameBase</i> – storage file name, <i>operationMode</i> – indicates whether to create the storage either on drive or in memory, <i>onMinSpaceReached</i> – delegate firing when the drive lacks free space.</p> <p>Returns - imported data as an OnDriveTable object.</p>
---	--

ReaderConfiguration Class

Namespace:	dataladder.Data
Assemblies:	DataMatch.Connectors.dll

Represents settings for data import and export.

C#
public class ReaderConfiguration : ICloneable

Examples

```
//Prepares for reading a data source from Excel file
ReaderConfiguration rc = new ReaderConfiguration();
rc.DataSourceTypeDescription = SourceDb.Excel.ToString();
rc.DataSourceId = 0;
rc.FileName = @"D:\ContactListMaster.xls";
// name that will be used in the application is assigned to TableName
rc.TableName = @"Contact List";
// name of the Excel sheet is assigned to OriginalTableName property
rc.OriginalTableName = @"Sheet1";
```

Remarks

This class contains all data source settings allowing to make import or export.

Different properties of this class are important for different types of sources. For example, file-based sources (CSV, Excel, etc.) needs the file name property to be filled, but for database sources ConfigurationString is obligatory.

Constructors

public ReaderConfiguration()	Default constructor.
public ReaderConfiguration(DataRow)	Creates an instance of ReaderConfiguration using the data stored in the DataRow specified.

Properties

DataSourceTypeDescription	String	get; set;	Gets or sets the data source type as a string.
DataSourceType	SourceDb	get;	Gets or sets the data source type.
DataSourceTypeDescriptionFriendly	String	get;	Gets the data source description
ConfigurationString	String	get; set;	Gets or sets the connection string (file path for file-based sources).

SchemaName	String	get; set;	Gets or sets the schema name for SQL based data sources.
TableName	String	get; set;	Gets or sets the table name.
OriginalTableName	String	get; set;	Get or sets the original name of the source.
TableAlias	String	get; set;	Gets or sets the table alias additional to the original name.
SelectCmd	String	get; set;	Gets or sets the SQL select query to be used for a SQL database source.
FileName	String	get; set;	Gets or sets the full path to the file.
ColumnsSettings	DataTable	get; set;	Gets or sets the column settings (count of columns, their names and types) as a DataTable object.
DataSourceId	Int32	get; set;	Gets or sets the ID of the data source.
IsFileBased	Boolean	get;	Indicates whether the data source is file-based.
ImportIssuesNumber	Int32	get; set;	Gets or sets the number of problems encountered during import.

Methods

LoadFromDataRow(DataRow)	Loads the state from the DataRow specified.
WriteToDataRow(DataRow)	Writes the state to the DataRow specified.
ToTable()	Writes the state to a DataTable.
FromTable(DataTable)	Loads the state from the DataTable.
LoadFromString(String)	Loads the state from the XML string specified.
Clone()	Returns a full copy of current instance.

SourceDb Enum

Namespace:	dataladder.Data
Assemblies:	DataMatch.Connectors.dll

Represents a type of a data source.

C#
public enum SourceDb

Fields

Excel	XLS file.
-------	-----------

Excel2007	XLSX file.
CSV	Delimited text file (CSV, TXT, DSV, etc.)
DBase	DBase table (DBF).
FixedWidthTextFile	Fixed width text file.
SqlServer	MS SQL Server database table.
Mysql	MySQL database table.
Odbc	ODBC data source.
DB2	DB2 database table.
Oracle	Oracle database table.
Salesforce	Salesforce CRM table.
PostgreSql	PostgreSQL database table.
OleUniversal	OLE data source.
Teradata	Teradata database table.

Remarks The table preceding contains the values currently supported by DME API. Other values are deprecated.

Field Class

Namespace:	dataladder.Data
Assemblies:	DataMatch.Connectors.dll

Represents a column of a data source.

C#
public class Field

Remarks

When a value is assigned to Name property, it is checked whether it contains prohibited symbols or such name is reserved.

If it is true , the value will be complemented with underscore symbols and stored in NewName property. This property's value is returned when Name or NewName is requested. Original value is stored in OriginalName property.

If property UseOriginalName is set to true, then Name property returns the original value of the name.

Constructors

Field()	Default constructor.
---------	----------------------

Properties

Name	String	get; set;	Gets or sets the name of the column.
NewName	String	get; set;	Gets or sets an alias of the column.
OriginalName	String	get;	Gets the name of this column in the original data source.
UseOriginalName	Boolean	get; set;	Indicates whether the original column name should be used.
Type	Type	get; set;	Type of column
Included	Boolean	get; set;	Indicates whether the column is selected for import.
Ordinal	Int32	get; set;	Gets or sets an index of this column in the table.
MaxLength	Int32	get; set;	Gets or sets maximum length of an object kept in a cell of the column.

Methods

SetName(String)	Sets the name of the column. This function rewrites the original column name.
-----------------	---

Fields Class

Namespace:	dataladder.Data
Assemblies:	DataMatch.Connectors.dll

Holds a list of data source columns.

C#
public class Fields : IContainer2CoordsMapper

Constructors

Fields()	Default constructor.
----------	----------------------

Properties

this[Int32]	Field	get; set;	Gets or sets a column by the index.
this[String]	Field	get; set;	Gets or sets a column by the name.
FieldNames	IEnumerable<String>	get;	Gets the names of the fields in the set.

Methods

Add(Field <i>field</i>)	Adds a new field to the end of the list of columns. <i>field</i> - field to add.
Clear()	Clears the list of columns.
Remove(Field <i>field</i>)	Removes the specified field from the list. <i>field</i> - field to remove.
SaveColumnsSettings()	Stores the state of current object to a DataTable and returns the table.
LoadColumnsSettings(DataTable <i>settings</i>)	Loads the state of a <see cref="Fields"/> object from the <see cref="DataTable"/> specified. <i>settings</i> - State stored in a tabular format.
GetIndex(String)	Gets the column index by the column name.
ExtractFieldsForRead(Boolean = false)	Resets the ordinals of the fields to consequent values for a master data source.
TestFieldByName(String)	Tests if current field list contains a field with the specified name and returns it's main characteristics if yes.

IContainer2CoordsMapper Class

Namespace:	dataladder.XtraGridHelper
Assemblies:	DataMatch.Core.dll

Provides basic functionality for working with table data.

C#
public interface IContainer2CoordsMapper

Properties

RecordCount	Int32	get;	Gets the record count.
ColumnCount	Int32	get;	Gets the column count.

Methods

GetData(Int32 <i>rowIndex</i> , Int32 <i>colIndex</i>);	Gets the value of a specific cell. <i>rowIndex</i> - row index of the cell, <i>colIndex</i> - column index of the cell, <i>returns</i> - value of the cell.
SetData(Object <i>obj</i> , Int32 <i>rowIndex</i> , Int32 <i>colIndex</i>)	Sets the value of a specific cell. <i>obj</i> – value to set, <i>rowIndex</i> - row index of the cell, <i>colIndex</i> - column index of the cell.
GetColumnName(Int32 <i>colIndex</i>)	Gets the name of a specific column. <i>colIndex</i> - index of the column, <i>returns</i> - column name.

OnDriveTable Class

Namespace:	dataladder.Data
Assemblies:	DataMatch.DataStorage.dll

Represents tabular data stored on disk.

C#
public class OnDriveTable : ITable2CoordsMapper, IGetMultipleReadOnlyViews

Examples

```
/// <summary>
/// Create simple storage.
/// </summary>
/// <param name="path">folder where new storage will be created in</param>
/// <param name="name">name of storage</param>
/// <returns></returns>
public static OnDriveTable CreateDemoTable(String path, String name)
{
    OnDriveTable onDriveTable = new OnDriveTable(path, name);
    onDriveTable.AddField("FirstName", typeof(String));
    onDriveTable.AddField("LastName", typeof(String));
    onDriveTable.SetData("John", 0, 0);
    onDriveTable.SetData("Smith", 0, 1);
    onDriveTable.SetData("Joan", 1, 0);
    onDriveTable.SetData("Smit", 1, 1);

    return onDriveTable;
}

/// <summary>
/// Prints the contents of an OnDriveTable in console window.
/// </summary>
/// <param name="mapper">data source that will be printed</param>
public static void PrintInConsole(OnDriveTable mapper)
{
    // Prints header
    for (Int32 i = 0; i < mapper.ColumnCount; i++)
    {
        String columnName = mapper.GetColumnName(i);
        Console.Write($"{i}:{columnName} ");
    }
    Console.WriteLine("|");
    Console.WriteLine("-----");

    //Prints body
    for (Int32 j = 0; j < mapper.RecordCount ; j++)
    {
        for (Int32 i = 0; i < mapper.ColumnCount; i++)
        {
            Object cellValue = mapper.GetData(j, i);
            String cellValueString;
            if (cellValue is Double doubleValue)
```

```

        {
            cellValueString = doubleValue.ToString("n2");
        }
        else
        {
            cellValueString = cellValue?.ToString() ?? String.Empty;
        }
        Console.Write($"| {i}:{cellValueString} ");
    }
    Console.WriteLine("|");
}
}

```

Remarks

Allows to access data, switch to RAM mode and back to disk, sort table rows. Supports own column data types that are mapped to .NET data types. This class keeps file streams open during its work, so it is important to clear resources when the instance is not used.

Main members

Constructors

OnDriveTable(String <i>path</i> , String <i>fileNameBase</i> , FileAccess <i>fileAccess</i> = FileAccess.ReadWrite, FileShare <i>fileShare</i> = FileShare.Read, Boolean <i>toDeleteExisting</i> = false, Boolean <i>loadExisting</i> = true, OperationModes <i>operationMode</i> = OperationModes.Disk, Boolean <i>inMemoryTableCompactMode</i> = false, PrioritiesForMemory <i>priorityForMemory</i> = PrioritiesForMemory.Normal, Int32 <i>estimatedRecordCount</i> = 0)		Creates a new instance of OnDriveTable class specifying full information.
<i>path</i>	Path of a folder where the table is to be placed to.	
<i>fileNameBase</i>	Base file name of the table.	
<i>fileAccess</i>	File access settings.	
<i>fileShare</i>	File share settings.	
<i>toDeleteExisting</i>	Defines whether the existing files have to be removed.	
<i>loadExisting</i>	Defines whether the data from an existing table should be loaded.	
<i>operationMode</i>	Specifies the target to place table data in, memory or disk.	
<i>inMemoryTableCompactMode</i>	Defines whether the compact mode should be used.	

<i>priorityForMemory</i>	Specifies the priority of the memory.
<i>estimatedRecordCount</i>	Gets an estimated record count of the table.
OnDriveTable(OperationModes operationMode = OperationModes.Disk, Boolean inMemoryTableCompactMode = false, PrioritiesForMemory priorityForMemory = PrioritiesForMemory.Normal, Boolean loadExisting = true)	Creates a new instance of OnDriveTable class. A GUID value is used as the file base name, common path as the data folder.
OnDriveTable(String path , OperationModes operationMode = OperationModes.Disk, Boolean inMemoryTableCompactMode = false, PrioritiesForMemory priorityForMemory = PrioritiesForMemory.Normal, Boolean loadExisting = true)	Creates a new instance of OnDriveTable class. A GUID value is used as the file base name, common path as the data folder.

Properties

this[Int32]	OnDriveField	get;	Returns a column by its index.
this[String]	OnDriveField	get;	Returns a column by its name.
ColumnCount	Int32	get;	Gets column count.
RecordCount	Int32	get;	Gets row count.
Name	String	get; set;	Gets or sets the name of the table.
ToDeleteFilesAfterClosing	Boolean	get; set;	Do remove files from disk after disposing of?
PriorityForMemory	PrioritiesForMemory	get; set;	Gets or sets the memory priority used for current table.
ValidDataInMemoryOnly	Boolean	get;	Gets a value indicating whether the memory-based table is changed.

Methods

AddField(String fieldName , Type type , Boolean allowNulls = true)	Adds to table new field. fieldName - name of new column, must be unique type - type of objects that will be kept in this column allowNulls - does it allow to be null (default 'true') returns – new field pointer
GetColumnName(Int32 colIndex)	Gets name of specific column.

	colIndex - index of column returns - column name
GetColumnNames()	Gets list of column names. Returns - array of column names
SetData(Object obj , Int32 rowIndex , Int32 colIndex)	Sets specific cell value (by column index or column name). obj - new value of cell
SetData(Object obj , Int32 rowIndex , String fieldName)	rowIndex - row index of cell colIndex - column index of cell fieldname – column name of cell
GetData(Int32 rowIndex , Int32 colIndex)	Gets value from specific cell (by column index or column name). rowIndex - row index of cell. colIndex - column index of cell. colName – column name of cell. returns - value from cell.
GetData(Int32 rowIndex , String colName)	
MakeWritable(OperationModes operationMode = OperationModes.Disk)	Makes the table writable.
static CreateNewTableWithTheStructureOfExisting (ITable2CoordsMapper table , String resultFileNameBase , String path = null, OperationModes operationMode = OperationModes.Disk)	[Obsolete] Creates new OnDriveTable with same count of columns, same column names table - table that is pattern resultFileNameBase - name of result table path - folder where new table will be created, if it is null then new table will be created in same folder that pattern table operationMode - will it be created on disk or in memory? returns - new instance of table
Move(String newPath)	Moves current table to a new folder. newPath – destination folder.
Move(String newPath , String newFileNameBase)	Moves current table to a new folder. newPath – destination folder; newFileNameBase - new base file name.
Dispose()	Releases the resources.

MatchEngine Class

Namespace:	dataladder.Matching
Assemblies:	DataMatch.Matching.dll

Provides an API for finding similar records in tabular data sources.

C#
public class MatchEngine : IDisposable

Examples

```
/// <summary>
/// Creates a match engine instance that works with a single data source.
/// </summary>
/// <param name="onDriveTable">file storage</param>
/// <param name="manager">matching </param>
/// <param name="tempPath">temp folder</param>
/// <param name="dataPath">result folder</param>
/// <param name="engineName">name of the engine</param>
public static void Matching(OnDriveTable onDriveTable,
    MultipleMatchDefinitionsManager manager,
    String tempPath, String dataPath, String engineName)
{
    // 1.Init
    MatchEngine matchEngine = new MatchEngine(manager, false,
        tempPath, dataPath, engineName);

    matchEngine.InputDataMapperDict.Clear();
    matchEngine.InputDataMapperDict.Add(onDriveTable.Name, onDriveTable);

    matchEngine.DataSourceIndexPairList.Clear();
    matchEngine.DataSourceIndexPairList.Add(new MatchEngine.DataSourceIndexPair(0, 0));

    // 2.Match
    matchEngine.DoIndex();
    matchEngine.DoMatch();
    matchEngine.ProcessFinalResults();

    // 3. Show results
    OnDriveTable pairsTable = matchEngine.PairsScoresTable;
    OnDriveTable groupsTable = matchEngine.FinalScoresGroupsTable;

    ConsoleHelper.PrintInConsole(pairsTable);
    ConsoleHelper.PrintInConsole(groupsTable);

    // 4. Clean
    matchEngine.Dispose();
}
```

Remarks

Matching involves consequent steps listed below:

- data source definition (*matchEngine.InputDataMapperDict*)
- data source pair configuration (*matchEngine.DataSourceIndexPairList*)
- indexing (*matchEngine.DoIndex()*)
- duplicate search (*matchEngine.DoMatch()*)
- process the search results, produce pair and group result views (*matchEngine.ProcessFinalResults()*)

Main members

Constructors

```
MatchEngine (MultipleMatchDefinitionsManager multipleMatchDefinitionsManager,  
             Boolean allRecordsInGoupMustBeSimilar,  
             String tmpPath, String dataPath, String name = "")
```

Create an instance of MatchEngine class.

<i>multipleMatchDefinitionsManager</i>	Manager containing match definition settings.
<i>allRecordsInGoupMustBeSimilar</i>	Value indicating whether only similar records should constitute a result group.
<i>tmpPath</i>	Temporary folder path.
<i>dataPath</i>	Common data folder path. Each match engine creates a dedicated subfolder to store its data.
<i>name</i>	Name of the engine. Used as a name of folder containing the main data.

Fields

DataPath	(readonly) Gets the main data folder path.
UncompleteDataPath	(readonly) Gets the temporary folder path.
MultipleMatchDefinitionsManager	(readonly) Gets the manager of the match definitions used in the matching.
InputDataMapperDict	(readonly) Gets the dictionary of data sources by their names (key - data source name, value - data source itself).
InputDataMapperList	(readonly) Gets the list of data sources.
DataSourceIndexPairList	(readonly) Gets the list of data source pairs involved into matching. Indices are permutable, e.g. 1-2 and 2-1 are the same pairs.
HighLevelScoresTable	Represents the table that contains MatchDefinition, Source IDs, Row IDs and Scores for every matching column.
PairsScoresTable	Represents the table aggregating matched record pairs and all the associated values calculated.

FinalScoresGroupsTable	Represents the table aggregating matched data joined into groups of similar records. This table is built basing on PairsScoresTable.

Properties

LoadAllInMemory	Boolean	get; set;	This property is used in API and SDK modes. If 'true' - match engine will work with some restrictions it will not be cleared, some structures will not created it is needed for increasing speed In regular version this is 'false'
FinalExportTable	OnDriveTable	get; set;	Gets or sets the table aggregating the results of matching and overwriting the data sources.
AllRecordsInGroupMustBeSimilar	Boolean	get; set;	Gets or sets the value indicating whether any result group should contain records that match to each record in the group.
MaxMatchesPerGroup	Int32	get; set;	Gets the maximum capacity of a group of matching results.

Methods

SetCachedDataSources(List<Int32> <i>list</i>)	Defines data sources which are not changing from matching to matching. So there is no necessity to reindex them. This feature is used in SDK and API. list - list with indexes of data sources which will be cached
DoIndex()	Indexes the data sources involved in the matching. return : value indicating whether the indexing completed successfully.
ReindexSingleDataSource(Int32 dataSourceIndex , OperationModes operationMode = OperationModes.Disk)	Calculates indexes only for one data source. Other sources will not be changed. dataSourceIndex - index of recalculated data source operationMode - keep result in memory or disk
DoMatch()	Runs the matching process with the settings specified.

Int32 maxMatchesForOneRow = 3, bool clearAllAfterMatching = true, Int32 maxMatchesPerGroup = 0)	<p>maxMatchesForOneRow - Maximum number of matches allowed for a single record.</p> <p>clearAllAfterMatching" - Indicates whether it is necessary to clear all the temporary data after the completion of the matching process.</p> <p>maxMatchesPerGroup" - Maximum capacity of matched records permitted.</p>
ProcessFinalResults(Boolean clearAllAfterMatching = true)	<p>Processes the raw matching results and creates the resulting pair and group tables.</p> <p>clearAllAfterMatching - Indicates whether to clean the indexers.</p>
Dispose()	Releases the resources used by the match engine. Clears up all the infrastructure involved.

MultipleMatchDefinitionsManager Class

Namespace:	dataladder.Matching
Assemblies:	DataMatch.Matching.dll

Class containing match definitions and match criteria settings, settings how to map columns from input data sources to output final table.

C#
public class MultipleMatchDefinitionsManager : IDisposable

Examples

```
MultipleMatchDefinitionsManager manager = new MultipleMatchDefinitionsManager();

// Create a simple criterion. Function 'CreateDummyDefinition' is
// shown in listing in the section about MatchCriteriaList class
MatchCriteriaList matchCriteriaList = CreateDummyDefinition(0, "Table", "FirstName");
manager.Add(matchCriteriaList);

// For simplifying of example here is used empty AvailableFields instance
// Example how AvailableFields instance can be initialized is shown in
// 'AvailableFields' class section
manager.AvailableFields = new AvailableFields();

manager.SetAbsoluteIndices();
```

Remarks

There are two structures in this class that define matching rules - AvailableFields and MultipleMatchCriteriaList.
First contains settings for mapping of input data source columns and which of them will go to final results. The second contains a list of definitions with all settings of all criteria.

Constructors

MultipleMatchDefinitionsManager ()	Default constructor.
------------------------------------	----------------------

Fields

AvailableFields	Holds settings how to map columns from input data sources to output final table.
-----------------	--

Properties

this[Int32]	MatchCriteriaList	get;	Gets a definition by its index
Count	Int32	get;	Gets the number of match definitions.

Methods

Add(MatchCriteriaList <i>definition</i>)	Add a new match definition. <i>definition</i> - new definition
Remove(Int32 <i>index</i>)	Remove a definition by its index. <i>index</i> - index of removed definition
SetAbsoluteIndices()	Recalculate indices of the matching criteria. This method should be called after finishing changing criteria and definitions and before starting matching.
Dispose()	Releases resources.

MatchCriteriaList Class

Namespace:	dataladder.Matching
Assemblies:	DataMatch.Matching.dll

Represents a set of match criteria and criteria groups constituting a single match definition.

C#
public class MatchCriteriaList : MatchCriteriaListCommon

Examples

```
/// <summary>
/// Create simple definition with one simple criterion
/// and with given column of some table
/// </summary>
/// <param name="id">id of definition</param>
/// <param name="tableName">name of table</param>
/// <param name="columnName">name of column</param>

public static MatchCriteriaList CreateDummyDefinition(int id, String tableName, String columnName)
{
    MatchCriteriaList definition = new MatchCriteriaList(id);

    // Create a simple criterion. Function 'CreateDummyCriterion' is
    // shown in listing in the section about MatchCriteria class
    MatchCriteria criterion = CreateDummyCriterion(tableName, columnName);

    definition.Add(criterion);
    definition.MarkTheFirstFieldInEveryGroup();

    return definition;
}
```

Main members

Constructors

MatchCriteriaList (Int32 <i>index</i>)	Creates an instance of this class specifying its index in the list of match definitions. <i>index</i> - Match definition index.
---	--

Properties

this[Int32]	MatchCriteria	get;	Gets a criterion by its index
Count	Int32	get;	Gets the number of criteria in the list.

ExactDefinitions	List<MatchCriteria>	get;	Gets the list of exact match criteria specified for the match definition.
AllDefinitionsAreExact	Boolean	get;	Gets a value indicating whether all the match criteria constituting the match definition are exact.
IndexFieldCount	Int32	get;	Gets the overall number of fields to be indexed.
FuzzyIndexFieldsCount	Int32	get;	Gets the number of fuzzy fields to be indexed.

Methods

Add()	Creates a new match criterion and adds it to this definition. Returns the newly added match criteria.
Add(MatchCriteria <i>matchCriterion</i>)	Adds a new match criterion to this definition. <i>matchCriterion</i> - existing criterion Returns added criterion.
MarkTheFirstFieldInEveryGroup()	Finds the first criterion in every group and marks it.
DetermineFieldsToIndex(String, IContainer2CoordsMapper)	Determines data source fields to be indexed for the current match criteria list.
GetMatchCriteriaByMatchingIndex(Int32)	Returns a match criteria by its matching index.

MatchCriteria Class

Namespace:	dataladder.Matching
Assemblies:	DataMatch.Matching.dll

Represents a single match criterion.

C#
public class MatchCriteria : IDisposable

Examples

```
/// <summary>
/// Create a simple criterion.
/// </summary>
/// <param name="tableName">the name of the table whose column
/// will be attached to the criterion. </param>
/// <param name="columnName">attached column </param>
/// <returns></returns>
public static MatchCriteria CreateDummyCriterion(String tableName, String columnName)
{
    MatchCriteria criterion = new MatchCriteria();

    criterion.Fuzzy = true;
    criterion.AddWeightToFirstLetter = false;
    criterion.Exact = false;
    criterion.Numeric = false;
    criterion.UseMetaphone = false;
    criterion.IgnoreCase = true;
    criterion.Level = 0.7f;
    criterion.GroupLevel = 0.0f;
    criterion.MinAllowedLevelInGroup = 0.0f;
    criterion.GroupId = -1;
    criterion.CrossColumnGroupId = -1;
    criterion.Weight = 100.0f;

    criterion.MapField(tableName, columnName);

    return criterion;
}
```

Constructors

MatchCriteria()	Default constructor.
-----------------	----------------------

Properties

Fuzzy	Boolean	get; set;	Gets or sets the value indicating whether the criteria implies fuzzy matching.
Exact	Boolean	get; set;	Gets or sets the value indicating whether the criteria implies exact matching.
Numeric	Boolean	get; set;	Gets or sets the value indicating whether the criteria implies numeric matching.
AddWeightToFirstLetter	Boolean	get; set;	Gets or sets the value indicating whether an additional value should be added to the score if the first letters of the words compared are equal.
UseMetaphone	Boolean	get; set;	Gets or sets the value indicating whether the criteria implies metaphone matching.
IgnoreCase	Boolean	get; set;	Gets or sets the value indicating whether casing is ignored during matching.
GroupId	Int32	get; set;	Gets or sets the identifier of the criteria group current criterion belongs to.
GroupLevel	Double	get; set;	Gets or sets the aggregate similarity level for a group of match criteria.
MinAllowedLevelInGroup	Double	get; set;	Gets or sets the level the criterion should meet to be matched as a grouped criterion.
CrossColumnGroupId	Int32	get; set;	Gets or sets the Cross Column ID set by user.
MaxTotalWeightBelow	Int32	get; set;	Gets or sets the upper limit for the sum of criteria weights that are similar.
MaxMismatchWeightBelow	Int32	get; set;	Gets or sets the upper bound for the sum of weights of mismatched fields in a criteria group.
MaxEmptyWeightBelow	Int32	get; set;	Gets or sets the upper bound of the sum of weights of empty fields compared.
Level	Double	get; set;	Gets or sets the degree of similarity for the column values selected for matching. Varies between 0 and 1 inclusively.
Weight	Double	get; set;	Gets or sets the weight of the criterion; cannot be less than 1 and greater than 1000.

Methods

GetMappedFieldName(String <i>dataSourceName</i> , out Boolean <i>fieldFound</i>)	Gets a field name for a specified data source from the current match criterion. <i>dataSourceName</i> - data source name <i>fieldFound</i> - result: true - if this criterion contains any column from specific table; false - if there is not any column from given data source Returns - found column name, if column is not found then null
ClearMapping()	Clears the set of fields constituting the match criterion.

MapField(String <i>dataSourceName</i> , String <i>fieldName</i>)	Adds a field from a data source to the current criterion. <i>dataSourceName</i> - table name <i>fieldname</i> - column name
UnmapField(String)	Removes a field from the criteria by the name of the data source it belongs to.
AdjustMaxTotalWeightBelow()	Corrects the value of MaxTotalWeightBelow basing on MaxMismatchWeightBelow and MaxEmptyWeightBelow values.

AvailableFields Class

Namespace:	dataladder.Matching
Assemblies:	DataMatch.Matching.dll

Manages the list of field mappings used to create the final export view.

C#
public class AvailableFields : ILoadableContainer2CoordsMapper

Examples

```
// 1.Create instance
AvailableFields af = new AvailableFields();

// Create FieldMapInfo. Function 'CreateFieldMapInfo' is
// shown in listing in the section about 'FieldMapInfo' class
FieldMapInfo fmiFirstName = CreateFieldMapInfo(0, "Table", "FirstName", 0);
FieldMapInfo fmiLastName = CreateFieldMapInfo(0, "Table", "LastName", 1);

// 2. Adding fields from table
// Create AvailableFieldsFromOneTable. Function 'CreateAFOT' is
// shown in listing in the section about 'AvailableFieldsFromOneTable' class
AvailableFieldsFromOneTable afot = CreateAFOT(onDriveTable,
        new List<FieldMapInfo> { fmiFirstName, fmiLastName });
af.TableList.Add(afot);

// 3. Map fields
MappedFieldsRow mfrFirstName = new MappedFieldsRow();
mfrFirstName["Table"] = fmiFirstName;
af.MappedFieldsRowList.Add(mfrFirstName);
MappedFieldsRow mfrLastName = new MappedFieldsRow();
mfrLastName.Add(fmiLastName);
af.MappedFieldsRowList.Add(mfrLastName);
```

Remarks

This class defines which columns from different sources will be mapped in one output final column
It contains two lists.

1. 'MappedFieldsRowList' contains definitions for every column in final table – this definition contains information which columns from input data sources to use and to include or not this column in final table
2. 'TableList' contains information about columns for every input data sources

Constructors

AvailableFields ()	Default constructor.
--------------------	----------------------

Properties

TableList	Holds sets of fields for all the data sources.
MappedFieldsRowList	Holds the set of field mappings used to compose the final results table.

Properties

this[Int32]	AvailableFieldsFromOneTable	get;	Gets the data source's field information by its index.
this[String]	AvailableFieldsFromOneTable	get;	Gets the data source's field information by its name.

Methods

GetTableName(Int32 <i>tableIndex</i>)	Gets the name of a data source by its index.
GetIncludedMappedFieldsRowList()	Gets the list of included field mappings (excluding empty mappings).
IsOneRecordPerGroupActiveForAnyDataSource()	Indicates whether at least one data source the AvailableFieldsFromOneTable.OneRecordPerGroup flag set.
IsRowIncluded(Int32)	Indicates whether the mapping row with a specific index is included in the final results.

AvailableFieldsFromOneTable Class

Namespace:	dataladder.Matching
Assemblies:	DataMatch.Matching.dll

Manages the list of fields in a particular data source.

C#
public class AvailableFieldsFromOneTable

Examples

```
/// <summary>
/// Creates AvailableFieldsFromOneTable instance. And adds a few field
/// </summary>
/// <param name="onDriveTable">file storage</param>
/// <param name="fmis">list of fields</param>
public static AvailableFieldsFromOneTable CreateAFOT(OnDriveTable onDriveTable,
List<FieldMapInfo> fmis)
{
    AvailableFieldsFromOneTable afot = new AvailableFieldsFromOneTable();
    afot.Table = onDriveTable;

    foreach (var fmi in fmis)
    {
        afot.Add(fmi);
    }

    return afot;
}

/// <summary>
/// Gets from AvailableFieldsFromOneTable instance information about columns
/// </summary>
/// <param name="afot">AvailableFieldsFromOneTable instance</param>
public static List<FieldMapInfo> GetMapInfo(AvailableFieldsFromOneTable afot)
{
    List<FieldMapInfo> list = new List<FieldMapInfo>();

    for (int i = 0; i < afot.Count; i++)
    {
        list.Add(afot[i]);
    }

    return list;
}
```

Constructors

AvailableFieldsFromOneTable()	Default constructor.
-------------------------------	----------------------

Properties

this[Int32]	FieldMapInfo	get;	Gets the field information by its index.
this[String]	FieldMapInfo	get;	Gets the field information by its name.
Table	ITable2CoordsMapper	get; set;	Gets or sets the data source the fields belong to.
Count	Int32	get;	Get the number of data source's field.
OneRecordPerGroup	Boolean	get; set;	Indicates whether a single result group must contain only a single record from the data source described.

Methods

Add(FieldMapInfo)	Adds a field information to the list.
Clear()	Clears the list of field information.

MappedFieldsRow Class

Namespace:	dataladder.Matching
Assemblies:	DataMatch.Matching.dll

Represents a column in the final results table.

Maps columns from different data sources to one resulting column.

C#
public class MappedFieldsRow

Examples

```
// Creates FieldMapInfo. Function 'CreateFieldMapInfo' is
// shown in listing in the section about 'FieldMapInfo' class
FieldMapInfo fmiFirstName = CreateFieldMapInfo(0, "Table", "FirstName", 0);
FieldMapInfo fmiLastName = CreateFieldMapInfo(0, "Table", "LastName", 1);

// Creates MappedFieldsRow instances and attaches FieldMapInfo in different ways
MappedFieldsRow mfrFirstName = new MappedFieldsRow();
mfrFirstName["Table"] = fmiFirstName; // in this case, the table name is indicated in explicit way

MappedFieldsRow mfrLastName = new MappedFieldsRow();
mfrLastName.Add(fmiLastName); // in this case table name is taken from FieldMapInfo
```

Constructors

FieldMapInfo()	Default constructor.
----------------	----------------------

Properties

this[String]	FieldMapInfo	get; set;	Gets or sets information about a column for the data source specified by name.
FirstNotEmptyFieldMapInfo	FieldMapInfo	get;	Gets the first not empty field in the mapped row.
Include	Boolean	get; set;	Get or sets a value indicating whether the column will be included into the final results table.

Methods

AddField(FieldMapInfo)	Adds a field to the mapped row.
RemoveField(String)	Removes a record from the row for the table specified.
IsEmpty()	Checks whether there is any field existing in the mapped row.
IsPlaceUsed(String)	Checks if the mapped row contains a field from the table specified.

FieldMapInfo Class

Namespace:	dataladder.Matching
Assemblies:	DataMatch.Matching.dll

Describes a single data source's field.

C#
public class FieldMapInfo : IDisposable

Examples

```
/// <summary>
/// Create instance of FieldMapInfo
/// </summary>
/// <param name="sourceId">index of data source</param>
/// <param name="tableName">name of data source</param>
/// <param name="columnName">name of column</param>
/// <param name="columnId">index of column</param>
/// <returns></returns>
public static FieldMapInfo CreateFieldMapInfo(Int32 sourceId,
    String tableName, String columnName, Int32 columnId)
{
    FieldMapInfo fmi = new FieldMapInfo(sourceId);
    fmi.FieldName = columnName;
    fmi.TableName = tableName;
    fmi.ColumnTransformation = null;
    fmi.FieldIndex = columnId;

    return fmi;
}
```

Constructors

FieldMapInfo(Int32 <i>dataSourceIndex</i>)	Create an instance of the class. <i>dataSourceIndex</i> - data source index.
---	--

Properties

FieldName	String	get; set;	Get or sets the name of the field.
FieldIndex	Int32	get; set;	Get or sets the index of the field in the data source.
TableName	String	get; set;	Get or sets the name of the data source.

DataSourceIndex	Int32	get;	Get or sets the index of the data source.
Mapped	Boolean	get; set;	Indicates whether the field is mapped to another field or not.
ColumnTransformation	IColumnTransformation	get; set;	Get or sets the field's standardization settings.

Methods

Dispose()	Releases the resources.
-----------	-------------------------

ProjectSpec Class

Namespace:	DataMatch.Project.Descriptors
Assemblies:	DataMatch.Project.dll

Class extracting the settings from 'dmeproj' file

C#
public class ProjectSpec : ITableSerializable

Examples

The example shows parsing of prepared project and outputs information about definitions and criteria in console

```
// load project
ProjectSpec projectSpec = new ProjectSpec();
ProjectSpecHelper.LoadProject(projectSpec, @"D:\API\Projects\testproject.dmeproj");
// get description of project
String description = projectSpec.Description;

// iterate through data sources
var sources = projectSpec.DataSource.DataSources;
foreach (DataSourceSpec source in sources)
{
    // find all fields that were merged in new one
    MergeInfoSpec merge = source.MergingInfo;
    foreach (var mf in merge.MergedFields)
    {
        Console.WriteLine($"merge field {String.Join(",", mf.FieldNames)} into
            '{mf.MergedFieldName}');
    }
}

// explore definitions and criteria
MatchDefinitionSpec matchDefSpec = projectSpec.MatchDefinition;
DefinitionsSpec definitions = matchDefSpec.Definitions;
// iterate through definitions
for (int i = 0; i < definitions.Definitions.Count; i++)
{
    DefinitionSpec definition = definitions.Definitions[i];
    Console.WriteLine($"Definition No {i}");

    // iterate through criteria
    for (int ii = 0; ii < definition.MatchCriteriaList.Count; ii++)
    {
        MatchCriteriaSpec criteria = definition.MatchCriteriaList[ii];
        Console.WriteLine($" Criteria No {ii}");
        foreach (var table_field in criteria.TableFieldDictionary)
        {
            Console.WriteLine($" {table_field.Key} - {table_field.Value}");
        }
    }
}
```

```

C:\WINDOWS\system32\cmd.exe
merge field City,State into 'City State'
Definition No 0
Criteria No 0
customer master - Company Name
Press any key to continue . . .

```

Remarks

'DataMatch Enterprise' application works with projects. Application has a rich user interface that allows to user investigate, standardize data, prepare a complex matching settings in the comfortable way. All settings and actions of user are kept in project file, file with 'dmeproj' extension

This class allows to get these settings from project file.

Constructors

ProjectSpec()	Create instance of class
---------------	--------------------------

Properties

Description	String	get; set;	description of porject
DataSource	DataSourcesSpec	get; set;	data sources settings
MatchDefinition	MatchDefinitionSpec	get; set;	matching settings
SavedResultsMatchDefinition	MatchDefinitionSpec	get; set;	matching settings for saved results
MatchingDataSourcePairs	SourcePairsSpec	get; set;	matching pairs settings
FinalExportSettings	FinalExportSpec	get; set;	Final Export Settings
DuplicatePairsBaseName	String	get; set;	Path to pairs table results
DuplicateGroupsBaseName	String	get; set;	Path to groups table results
DuplicateGroupsFirstRow	Int32	get; set;	
MergeSettings	ResultOverwriterSpec	get; set;	Merge Survivorship Overwrite settings
MasterRecordDeterminant	ResultOverwriterSpec	get; set;	Merge Survivorship master record settings

AddressVerificationSettings	AddressVerificationSpec	get; set;	Address Verification settings
ProfilerPatternOptions	ProfilerPatternSpec	get; set;	profiler settings

Methods:

FromTable(DataTable table)	Loads this file properties from DataTable instance table - DataTable inside which was saved instance of this class
---------------------------------------	--

ProjectSpecHelper Class

Namespace:	DataMatch.Project.Helpers
Assemblies:	DataMatch.Project.dll

Class containing methods that help working with DME project file

C#
public static class ProjectSpecHelper

Examples

```
ProjectSpec projectSpec = new ProjectSpec();  
ProjectSpecHelper.LoadProject(projectSpec, @"D:\test.dmeproj");  
String description = projectSpec.Description;
```

Methods

LoadProject(ProjectSpec <i>projectSpec</i> , String <i>fileName</i>)	Load to specific ProjectSpec instance settings from DME project file <i>projectSpec</i> - modified ProjectSpec instance <i>filename</i> - path to DME project file
--	--

ColumnTransformationsSpec Class

Namespace:	DataMatch.Project.Descriptors.DataSource
Assemblies:	DataMatch.Project.dll

Data source standardization settings

C#
public class ColumnTransformationsSpec : ITableSerializable

Examples

```
// How get ColumnTransformationsSpec from project file

ProjectSpec projectSpec = new ProjectSpec();
ProjectSpecHelper.LoadProject(projectSpec, @"D:\test.dmeproj");
String description = projectSpec.Description;
var sources = projectSpec.DataSource.DataSources;

foreach (DataSourceSpec source in sources)
{
    ColumnTransformationsSpec stand = source.StandardizedInfo;
}
```

Constructors:

ColumnTransformationsSpec()	Create instance of class
-----------------------------	--------------------------

Properties

Transformations	List <ColumnTransformationSpec>	get; set;	List of column transformation settings This list contains settings for each data source column
-----------------	------------------------------------	--------------	--

Methods

FromTable(DataTable table)	Loads from DataTable instance table - DataTable inside which was saved instance of this class
---------------------------------------	---

ColumnTransformationSpec Class

Namespace:	DataMatch.Project.Descriptors.DataSource
Assemblies:	DataMatch.Project.dll

Column standardization settings

C#
public class ColumnTransformationSpec

Constructors

ColumnTransformationSpec ()	Create instance of class
-----------------------------	--------------------------

Properties

FieldName	String	get; set;	Column name
Type	String	get; set;	Type of column. String representation of TransformationType enum.
CopyField	Boolean	get; set;	Should copy this field?
ChangeCase	Boolean	get; set;	Should change case?
UpperCase	Boolean	get; set;	Should transform to upper case?
LowerCase	Boolean	get; set;	Should transform to lower case?
ProperCase	String	get; set;	Proper case settings
RemoveNonPrintableCharacters	Boolean	get; set;	Should remove non printable chars?
ReplacementForNonPrintableCharacters	String	get; set;	What to replace non printable chars
ReplacementForEmptyValues	String	get; set;	Replacement for empty values
RemoveLeadingSpaces	Boolean	get; set;	Should remove leading spaces?
RemoveTrailingSpaces	Boolean	get; set;	Should remove trailing spaces?
CharactersToRemove	String	get; set;	Chars for removing
CharactersToReplace	String	get; set;	Chars replacement settings
CharactersToReplaceCaseSensitive	String	get; set;	Chars replacement is case sensitive

RemoveSpaces	Boolean	get; set;	Should remove all spaces?
RemoveLetters	Boolean	get; set;	Should remove all letters?
RemoveDigits	Boolean	get; set;	Should remove numbers?
ReplaceZerosWithOs	Boolean	get; set;	Should replace zeros with 'O'?
ReplaceOsWithZeros	Boolean	get; set;	Should remove letter 'O' with zeros?
Regex	String	get; set;	Pattern Builder settings
WordSmith	String	get; set;	Wordsmith settings

MergeInfoSpec Class

Namespace:	DataMatch.Project.Descriptors.DataSource
Assemblies:	DataMatch.Project.dll

Settings for merging fields during standardization

C#
public class MergeInfoSpec : ITableSerializable

Examples

```
// How get MergeInfoSpec from project file

ProjectSpec projectSpec = new ProjectSpec();
ProjectSpecHelper.LoadProject(projectSpec, @"D:\test.dmeproj");
String description = projectSpec.Description;
var sources = projectSpec.DataSource.DataSources;

foreach (DataSourceSpec source in sources)
{
    MergeInfoSpec merge = source.MergingInfo;
}
```

Remarks

This class has property 'Merged Fields'. Every item from the list represents one merged field and settings for its. Class with settings is described in separate section of this document (see 'MergedFieldSpec')
Delimiter for all merged fields is the same.

Constructors

MergeInfoSpec()	Create instance of class
-----------------	--------------------------

Properties

MergedFields	List<MergedFieldSpec>	get; set;	Settings for every merged field
Delimiter	String	get; set;	Delimiter that will be added between merged values

Methods

FromTable(DataTable <i>table</i>)	Loads properties from DataTable instance <i>table</i> - DataTable inside which was saved instance of this class
--	--

MergedFieldSpec Class

Namespace:	DataMatch.Project.Descriptors.DataSource
Assemblies:	DataMatch.Project.dll

Class describing how get new merged column

C#
public class MergedFieldSpec : ITableSerializable

Example

The example from ProjectSpec class section shows how possible to work with this class

Constructors

MergedFieldSpec()	Create instance of class
-------------------	--------------------------

Properties

FieldName	String	get; set;	New column name
MergedFields	List<String>	get; set;	Names of columns that will be merged into new column

Methods

FromTable(DataTable <i>table</i>)	Loads properties from DataTable instance <i>table</i> - DataTable inside which was saved instance of this class
------------------------------------	--

MatchDefinitionSpec Class

Namespace:	DataMatch.Project.Descriptors.MatchDefinition
Assemblies:	DataMatch.Project.dll

Matching settings: definitions and how to map fields

C#
public class MatchDefinitionSpec : ITableSerializable

Example

The example from ProjectSpec class section shows how possible to work with this class

Constructors

MatchDefinitionSpec()	Create instance of class
-----------------------	--------------------------

Properties

MappedFields	MappedFieldsSpec	get; set;	settings for mapping fields
Definitions	DefinitionsSpec	get; set;	setings for definitions
AllRecordsInGoupMustBeSimilar	Boolean	get; set;	should similarity between each record to others in group must be more than level of criterion
AutogenerateReport	Boolean	get; set;	generate report after matching

Methods

FromTable(DataTable table)	Loads MatchDefinitionSpec properties from DataTable instance table - DataTable inside which was saved instance of this class
------------------------------------	---

DefinitionsSpec Class

Namespace:	DataMatch.Project.Descriptors.MatchDefinition
Assemblies:	DataMatch.Project.dll

List of definitions settings

C#
public class DefinitionsSpec : ITableSerializable

Example

The example from ProjectSpec class section shows how possible to work with this class

Constructors

DefinitionsSpec()	Create instance of class
-------------------	--------------------------

Properties

Definitions	List<DefinitionSpec>	get; set;	List of definitions
-------------	----------------------	--------------	---------------------

Methods

FromTable(DataTable table)	Loads DefinitionsSpec properties from DataTable instance table - DataTable inside which was saved instance of this class
------------------------------------	---

DefinitionSpec Class

Namespace:	DataMatch.Project.Descriptors
Assemblies:	DataMatch.Project.dll

Match Definition description – class that contains list of criterion, list of rules for matching

C#
public class DefinitionSpec : ITableSerializable

Example

The example from ProjectSpec class section shows how possible to work with this class

Constructors

DefinitionSpec()	Create instance of class
------------------	--------------------------

Properties

MatchCriteriaList	List<MatchCriteriaSpec>	get; set;	Criteria belonging to this definition
-------------------	-------------------------	--------------	---------------------------------------

Methods

FromTable(DataTable table)	Loads DefinitionSpec properties from DataTable instance table - DataTable inside which was saved instance of this class
------------------------------------	--

MatchCriteriaSpec Class

Namespace:	DataMatch.Project.Descriptors.MatchDefinition
Assemblies:	DataMatch.Project.dll

Settings for match criterion

C#
public class MatchCriteriaSpec

Example

The example from ProjectSpec class section shows how possible to work with this class

Remarks

The example from ReaderToVariableTableConverter class section shows how possible to work with this class
--

Constructors

MatchCriteriaSpec()	Create instance of class
---------------------	--------------------------

Properties

TableFieldDictionary	Dictionary <String, String>	get; set;	Dictionary that contains columns attaching to this criterion Key - table name, Value - Column name
Fuzzy	Boolean	get; set;	Use fuzzy matching for this criterion
Exact	Boolean	get; set;	Use exact matching for this criterion
Numeric	Boolean	get; set;	Make matching of columns attached to this criterion as numbers
AddWeightToFirstLetter	Boolean	get; set;	Increase a score of matching if first letters of records are same
UseMetaphone	Boolean	get; set;	Use matching based on the pronunciation of records
IgnoreCase	Boolean	get; set;	Ignore case during matching
GroupId	Int32	get; set;	Identifier of group. All criterion with same Id will enter in one group.

GroupLevel	Double	get; set;	Aggregate similarity level for a group of match criteria.
MinAllowedLevelInGroup	Double	get; set;	Level for a group. If criterion belongs to group, this is additional checking level. If score of matching is more than this value in this case records will be similar
CrossColumnGroupId	Int32	get; set;	Cross Column ID set by user
MaxTotalWeightBelow	Int32	get; set;	The upper limit for the sum of criteria's weight that are similar
MaxMismatchWeightBelow	Int32	get; set;	The upper limit for the sum of criteria's weight that are mismatched
MaxEmptyWeightBelow	Int32	get; set;	Max empty weight for a group. Every criterion in the group that has empty records adds its own weight to the sum of empty weights. This sum must be less than this property. If the sum is more than this limit then such row will not be recognized as similar
Level	Double	get; set;	Value indicating a degree of expected similarity for the column values selected for matching. Must be in limits [0.0 ... 1.0]
Weight	Double	get; set;	Weight of the criterion; cannot be less than 1 and greater than 1000
MatchingIndex	Byte	get; set;	Index of fuzzy criterion in criteria list. Only fuzzy is calculated
AbsoluteMatchingIndex	Byte	get; set;	Index of criterion in criteria list

Methods

FromTable(DataTable table)	Loads MatchCriteriaSpec properties from DataTable instance table - DataTable inside which was saved instance of this class
------------------------------------	--

MappedFieldsSpec Class

Namespace:	DataMatch.Project.Descriptors.MatchDefinition
Assemblies:	DataMatch.Project.dll

List of mapped fields. How columns from input data sources are mapped each to others

C#
public class MappedFieldsSpec : ITableSerializable

Constructors

MappedFieldsSpec()	Create instance of class
--------------------	--------------------------

Properties

MappedFields	List<MappedFieldSpec>	get; set;	List of mapped field
--------------	-----------------------	-----------	----------------------

Methods

FromTable(DataTable table)	Loads MappedFieldsSpec properties from DataTable instance table - DataTable inside which was saved instance of this class
ToTable()	Stores current instance's state in a DataTable.

MappedFieldSpec Class

Namespace:	DataMatch.Project.Descriptors.MatchDefinition
Assemblies:	DataMatch.Project.dll

Represents settings for one output column what input columns to map together

C#
public class MappedFieldSpec

Remarks

This class contains information how to join a few input columns from input sources into one output column.
--

Constructors

MappedFieldSpec()	Initializes an instance of MappedFieldSpec with an empty set of mapped fields.
-------------------	--

Properties

Include	Boolean	get; set;	Gets or sets the value indicating whether the mapping described is to be included into the resulting table.
MappedFields	Dictionary <String, String>	get; set;	Gets or sets the dictionary of fields constituting the mapping. Key - table name, value - column name

RegistrationWrapper Class

Namespace:	dataladder.Licensing
Assemblies:	DataMatch.Core.dll

Helper for accessing the registration.

C#
public class RegistrationWrapper

Examples

```
// How to point path to folder where license.txt file is placed
RegistrationWrapper Registration = new RegistrationWrapper();
Registration.CustomPathForRegistrationFile = @"D:\API\Registration";

// How to check expiration date
DateTime expirationDate = RegistrationWrapper.ExpirationDate;
Boolean registered = expirationDate >= DateTime.Now;
```

Remarks

It is important to call property ExpirationDate because this make initiation of this class.

Constructors

RegistrationWrapper(String)	Creates an instance RegistrationWrapper specifying the product ID.
-----------------------------	--

Properties

CustomPathForRegistrationFile	String	get; set;	Gets or sets the path where the file with the registration key is placed.
RegistrationFileName	String	get;	Gets the name of the registration file.
ExpirationDate	DateTime	get;	Gets the expiration date.
ThisPcSignature	String	get;	Gets the signature of the machine.

Methods

WriteRegistrationFile(String)	Creates a text file with the key placed inside.
-------------------------------	---

IOHelper Class

Namespace:	dataladder.IO
Assemblies:	DataMatch.Core.dll

Helper for working with files, folders, etc.

C#
public static class IOHelper

Examples

```
bool res = false;
if (!(res = System.IO.Directory.Exists(@"D:\Some folder")))
{
    Console.WriteLine(res);    //true - folder doesn't exist yet
    res = dataladder.IO.IOHelper.CreateDirectoryIfNotExist(@"D:\Some folder");
    Console.WriteLine(res);    //true - checks that function returned 'true'
    res = System.IO.Directory.Exists(@"D:\Some folder");
    Console.WriteLine(res);    //true - checks that folder was created
    res = dataladder.IO.IOHelper.CreateDirectoryIfNotExist(@"D:\Some folder");
    Console.WriteLine(res);    //true - that function returns true when folder
                                //already exists
}
```

Constructors

This is static class so there isn't necessary to create instance of this class

Methods

CreateDirectoryIfNotExist(String <i>dirPath</i>)	Checks the path to a specific directory if this path doesn't exist then all directories on this path will be created <i>dirPath</i> – checked/created path <i>returns</i> - directory exists
--	---

TransformationDiagram Class

Namespace:	<code>dataladder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Encapsulates a chain of transformations, input and output fields. Allows to perform transformation for a single record.

C#
<code>public class TransformationDiagram</code>

Examples

The following code example creates `TransformationDiagram` array for multithreaded transformation.

C#
<pre>var transformationDiagramArr = new TransformationDiagram[coreCount]; for (Int32 taskIndex = 0; taskIndex < coreCount; taskIndex++) { transformationDiagramArr[taskIndex] = new TransformationDiagram(taskIndex); }</pre>

Remarks

Constructors

<code>TransformationDiagram(Int32)</code>	Create an instance of <code>TransformationDiagram</code> for a task with specific index.
---	--

Properties and Public Fields

<code>ContainsCassBlock</code>	Gets or sets the value indicating whether at least one CASS transformation is applied.
<code>TransformedOutputs</code>	Gets the outputs which are a product of some data transformation with transformation blocks. Other kind of outputs is a non-transformed column from input.
<code>TaskIndex</code>	Gets the index of a task which performs transformation (for multithreaded processing).
<code>Inputs</code>	Gets the column names and indexes in input data source.
<code>Outputs</code>	Gets the list of output columns after the transformation.

Methods

<code>AddTransformationBlock(TransformationBlock)</code>	Adds a new atomic <code>TransformationBlock</code> to the transformation list.
<code>AddInput(String, Int32)</code>	Creates a new data flow, adds it to the list of inputs and returns the created object.
<code>AddOutput(DataFlow)</code>	Add a data flow to the list of outputs.
<code>Clear()</code>	Clears the list of transformation blocks, inputs and outputs.
<code>CreateOutputs()</code>	Creates outputs as the result of the chain of transformations.
<code>Prepare()</code>	Arranges the transformation blocks that constitute the diagram and resets the <code>TransformedOutputs</code> collection.
<code>GetTransformedOutputCount()</code>	Returns the amount of output columns that are the results of transformations applied to the data source's columns.
<code>IsBlockBeforeOther(TransformationBlock before, TransformationBlock after)</code>	Determines whether the first transformation block in the parameter list precedes the second block.
<code>Process(Int32)</code>	Performs all the transformations defined in the Transformation Diagram for a single row with index <code>Int32</code> .

DataFlow Class

Namespace:	dataaladder.Data.DataTransformation
Assemblies:	DataMatch.Transformation.dll

Represents a data flow concept consisted of the value, the index of input/output column the value came from/goes to, and the blocks of transformation the data flow connects.

C#
<pre>public class DataFlow</pre>

Examples

The following code example creates a new [DataFlow](#) and adds it to a [DataFlowCollection](#).

C#
<pre>String fieldName = "FirstName"; Int32 fieldIndex = 1; DataFlowCollection flowCollection = new DataFlowCollection(); DataFlow dataFlow = new DataFlow(fieldName, fieldIndex); flowCollection.Add(dataFlow, fieldName);</pre>

Remarks

Constructors

DataFlow(String, Int32)	Creates the new data flow, defines its name String and the index Int32 in input data source.
DataFlow(String)	Creates a data flow with the name String .

Properties and Public Fields

InputIndex	Gets the index of the field that represents the input of the data flow.
OutputTransformationBlocks	Gets the list of transformation blocks that have the current data flow as an input.
Name	Gets or sets the name of the data flow. Can hold either input or output column name.
OutputIndex	Gets or sets the index of the output field produced by the data flow.
InputTransformationBlock	Gets or sets the TransformationBlock object the data flow has its input from.
InputType	Gets the type of the input of the data flow. It can be either the original data source or results of another transformation.
Value	Gets or sets the value handled by the data flow.
Tag	Get or sets the tag that can be attached to the DataFlow object.

<code>InDiagramOutput</code>	Gets or sets the value indicating whether the data flow belongs to the inputs of a transformation diagram.
------------------------------	--

Methods

<code>IsAfter(DataFlow)</code>	Determines whether current <code>DataFlow</code> instance is before the <code>DataFlow</code> instance specified as the parameter.
--------------------------------	--

DataFlow.InputTypes Enum

Namespace:	<code>dataaladder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Defines the origin of data flow - it can be just a copy of input data column, or be a result of some transformation.

C#
<code>public enum InputTypes</code>

Fields

<code>DirectFromDataSource</code>	0	Data is copied from input data source without any transformations.
<code>FromTransformationBlock</code>	1	Data is the result of some transformation.

DataFlowCollection Class

Namespace:	dataladder.Data.DataTransformation
Assemblies:	DataMatch.Transformation.dll

Represents a collection of DataFlow objects.

C#
<pre>public class DataFlowCollection</pre>

Examples

The following code example creates the new [DataFlow](#) and adds it to a [DataFlowCollection](#).

C#
<pre>String fieldName = "FirstName"; Int32 fieldIndex = 1; DataFlowCollection flowCollection = new DataFlowCollection(); DataFlow dataFlow = new DataFlow(fieldName, fieldIndex); flowCollection.Add(dataFlow, fieldName);</pre>

Remarks

Constructors

DataFlowCollection()	Creates the new instance of DataFlowCollection , initializes the list of data flows.
--------------------------------------	--

Properties and Public Fields

this[Int32]	Gets or sets a DataFlow element by its index. For output columns ensures that column names are unique.
this[String]	Gets a DataFlow element by its name. For output columns ensures that column names are unique.
Count	Gets the number of entries in the collection.

Methods

Add(DataFlow, String)	Adds a data flow to the collection.
Remove(DataFlow)	Removes a data flow from the collection.
Contains(DataFlow)	Returns the value indicating whether the collection contains a data flow.
Sort()	Sorts data flows in the list using default DataFlowComparer .
Clear()	Clears the collection.

TransformationBlock Class

Namespace:	<code>dataladder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

A base class for all types of transformations. Responsible for storing input and output fields, includes prototypes for the main actions like process, add inputs & outputs, clearing inputs and outputs.

C#
<code>public abstract class TransformationBlock</code>

Examples

The following code example creates the new `CopyColumnBlock` and assigns it to `TransformationBlock`.

C#
<pre>String fieldName = "FirstName"; Int32 fieldIndex = 1; TransformationDiagram diagram = new TransformationDiagram(0); DataFlow input = diagram.AddInput(fieldName, fieldIndex); TransformationBlock copyColumnBlock = new CopyColumnBlock(input);</pre>

Remarks

In this class occurs every kind of data transformation, like convert to lower case, upper case, remove non-printable characters, parse names, addresses, merge values. Any transformation block can receive data from input data directly or from other transformation block; the only limitation to making a diagram with transformation blocks is recursion. Output from one block cannot return to the same block directly or through other block(s).

Constructors

<code>TransformationBlock()</code>	A default constructor.
------------------------------------	------------------------

Properties and Public Fields

<code>Inputs</code>	Holds the collection of data flows constituting the inputs of the transformation block.
<code>Outputs</code>	Holds the collection of data flows constituting the outputs of the transformation block.

Methods

<code>SetInput(Int32, DataFlow)</code>	Adds a data flow to the inputs of the transformation block at a specific index.
<code>CleanIO()</code>	Clears input and output lists.
<code>CleanOutputs()</code>	Clears the list of outputs.

CleanInputs()	Clears the list of inputs.
Process()	Performs the transformation defined by the block.

CassInputTypes Enum

Namespace:	dataadder.Data.DataTransformation
Assemblies:	DataMatch.Transformation.dll

Defines the list of possible input field types available in Address Verification module.

C#
public enum CassInputTypes

Fields

CompanyName	0	Input field contains Company or Firm name.
PrimaryAddress	1	Input column contains primary address.
SecondaryAddress	2	Input column contains secondary address.
CityName	3	Input column contains city name.
StateName	4	Input column contains state name.
ZipCode	5	Input column contains US postal code (Zip code).
Urbanization	6	Denotes an area, sector, or residential development within a geographic area.
Country	7	Column contains country name. (Used for Canadian address verification).
PostalCode	8	Canadian postal code (if Country field is defined and equals 'Canada') or US Zip code for US addresses.

Examples

The following code example shows how to use [CassInputTypes](#) enumeration.

C#

Remarks

CassOutputParts Enum

Namespace:	dataadder.Data.DataTransformation
Assemblies:	DataMatch.Transformation.dll

Defines all possible output field types that are populated after validating of address using CASS Address Verification module.

C#
public enum CassOutputParts

Fields

Status	0	Result return code. Possible values: V - address verified; M - multiple response; N - address not verified.
ResidentialDelivery-Indicator	1	Residential Delivery Indicator (RDI). Possible values: Y = Residential Delivery; N = Not Residential Delivery; Blank = Did not query RDI.
CompanyFirm	2	Firm/Company.
PrimaryAddress	3	Standardized Primary Address.
SecondaryAddress	4	Secondary Address.
City	5	Standardized Preferred City.
State	6	Standardized State Abbreviation.
ZipCode	7	Standardized ZIP Code.
PostalCode	8	Standardized Postal Code (Canada Only).
Plus4	9	+4 Code.
DeliveryPointCheck-Digit	10	Delivery Point and Check Digit.
DeliveryPoint	11	Delivery Point.
Plus6	12	Zip + 6.
CarrierRouteCode	13	Carrier Route Code.
LineOfTravelNumber	14	Line of Travel Number.
LineOfTravelCode	15	Line of Travel Asc/Desc Code.
AddressRecordType	16	Address Type.
Urbanization	17	Urbanization.
StateFips	18	State FIPS Code.
CountyFips	19	County FIPS Number.
CountyName	20	County Name.
CongressDist	21	Congressional District Number.
PreferredCity	22	Standardized Preferred City.
AbbreviatedCity	23	Standardized Abbreviated City Name (if available).
LastLine	24	Complete Standardized Last Line.
LacsMatch	25	LACS Match.

LacsLink	26	LACSLink Return Code.
LacsLinkIndicator	27	LACSLink Indicator.
SuiteLink	28	SuiteLink Code.
ReturnCode	29	Return Code. Possible values: 10 = Invalid Address; 11 = Invalid ZIP code; 12 = Invalid State Code; 13 = Invalid City; 21 = Address not found; 22 = Multiple response; 31 = Single response (Exact Match); 32 = Default response (Missing information - Ste #, or Invalid Ste #).
ErrorAndWarning	30	Warnings or Errors. Possible values: A# ZIP; B# City/State Corrected C# Invalid city/state/zip D# No ZIP assigned E# ZIP assigned for multiple response F# No ZIP available G# Part of firm moved to address H# Secondary number missing I# Insufficient/incorrect data J# Dual input K# Multi caused by cardinal rule L# Deliver address component add/del/chg M# Street name spelling changed N# Delivery address was standardized O# Low +4 tie-breaker (multi-response) P# Better delivery address exists Q# Unique ZIP Code R# No match due to EWS (Early Warning System) S# Invalid secondary number T# Multiple caused by magnet rule U# Unofficial Post Office name V# Unverifiable city/state W# Small town default X# Unique ZIP code generated Y# Military match Z# ZIP move match
BuildingNumber	31	Parsed Primary Number.
PreDirection	32	Parsed Pre-direction.
StreetName	33	Parsed Street Name.
PostDirection	34	Parsed Post Direction.
Suffix	35	Parsed Suffix.
SecondaryName	36	Parsed Unit Description.
SecondaryNumber	37	Parsed Secondary Number.
PMBIndicator	38	Private Mail Box Description.

PMBNumber	39	Private Mail Box Number.
VacancyFlag	40	Delivery Point Confirmation Indicators - DPV Vacancy Indicator.
CountyFIPSCode	41	County FIPS Number.
StateFIPSCode	42	State FIPS Code.
CongressDistNumber	43	Congressional District Number.
DPV	44	Delivery Point Confirmation Indicators. Combines the next result values into a single string: DPV Confirmation Indicator; DPV CMRA Indicator; DPV False Positive Indicator; DPV Vacancy Indicator; DPV No Stats Indicator.
DPVFlag	45	Delivery Point Confirmation Footnotes. A combination of results of the next variables: 1) AA Input Address Matched to the ZIP + 4 file; 2) BB Input Address Matched to DPV (all components). If parameter (1) returns - adds 'AA'; If parameter (2) returns - adds 'BB'; Example: "", "AA", "BB", "AABB".
LACSFlag	46	LACSLink Return Code.
LACSReturnCode	47	LACSLink Return Code.
StelinkInd	48	SuiteLink Indicator.
CensusTract	49	Census Tract.
CensusBlockGroup	50	Census Block Group.
Latitude	51	Latitude.
Longitude	52	Longitude.
ErrorExplanation	53	Textual explanation of address verification error code.

Examples

The following code example shows how to use [CassOutputParts](#) enumeration.

C#

Remarks

TransformationTypes Enum

Namespace:	<code>dataadder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Defines a list of all possible transformation types.

C#
<pre>public enum TransformationTypes</pre>

Fields

None	0	No special column type defined.			
FullName	1	Splits column that contains Full Name into First Name, Last Name, Gender, etc.			
FirstName	2	Adds new columns with Common Name and Gender. E.g. Ron -> Ronald; Jim -> James, Carrie -> Charles, etc.			
Address	3	Can be applied to input fields that contains Primary Address, Secondary Address, City, Zip, State. If more input fields with type Address are added, more accurate parsed address will be retrieved.			
Zip	4	Can be applied to fields that contains US Zip codes or Canadian Postal Codes. If this transformation type is applied, a field with postal code will be split into ZipA, ZipB and Country subfields.			
		Zip	ZipA	ZipB	Country
		49512-1368	49512	1368	USA
		X0E 0C0	X0E 0C0	–	Canada
VCompanyName	5	Company/Firm.			
VPrimaryAddress	6	Address Line 1 (Primary Address).			
VSecondaryAddress	7	Address Line 2 (Secondary Address or Suite/Apt).			
VCityName	8	City.			
VStateName	9	State.			
VZipCode	10	ZIP or ZIP+4 (Do not use for Canada).			
VUrbanization	11	Urbanization.			
VCountry	12	Country (Required Canada Only). Should be "CANADA".			
VPostalCode	13	Canadian Postal code (Required Canada Only). E.g. "X0E 0C0".			

Examples

The following code example shows how to use `TransformationTypes` enumeration.

```
C#
ColumnTransformation columnTransformation;
TransformationDiagram diagram;
DataFlow input;
. . .

switch (columnTransformation.TransformationType)
{
    case TransformationTypes.Zip:
        ZipBlock zipBlock = new ZipBlock();
        diagram.AddTransformationBlock(zipBlock);
        zipBlock.AddInput(input);
        break;
    default:
        break;
}

. . .
```

Remarks

IColumnTransformation Interface

Namespace:	<code>dataadder.Data</code>
Assemblies:	<code>DataMatch.Core.dll</code>

Interface that describes all transformations that can be applied to a single input column. Includes Case Changing, Replacement, Remove Characters operations, etc.

C#
<code>public interface IColumnTransformation : IDisposable</code>

Examples

The following code example shows how to use `IColumnTransformation` interface.

C#

Remarks

Properties	Gets or sets the name of the input data source field to which transformation will be applied.
<code>FieldName</code>	
<code>CopyField</code>	Gets or sets the value indicating whether a copy of source input field will be just transferred to outputs without any changes. Transformation will be applied to original input field that will be modified and added to outputs.
<code>ChangeCase</code>	Gets or sets the value indicating that symbols in the column must change their case, e.g. "Boston" -> "bOSTON".
<code>UpperCase</code>	Gets or sets the value indicating that symbols in the column must be made uppercase.
<code>LowerCase</code>	Gets or sets the value indicating that symbols in the column must be made lowercase.
<code>ProperCase</code>	Gets or sets the value indicating that symbols in the column must be made proper case, e.g. "BOSTON" -> "Boston".
<code>RemoveNonPrintable-Characters</code>	Gets or sets the value indicating that all the characters that are categorized by Unicode as whitespaces (except whitespace symbol itself) must be removed.
<code>ReplacementForNon-PrintableCharacters</code>	Gets or sets the value indicating that all the symbols that are treated by Unicode as whitespaces (except whitespace itself) should be replaced with a value defined by current property.
<code>ReplacementFor-EmptyValues</code>	Gets or sets the replacement for cells that contains NULLs or empty string values.
<code>RemoveLeadingSpaces</code>	Gets or sets the value indicating that all the whitespace characters in the beginning of the input text should be removed.
<code>RemoveTrailingSpaces</code>	Gets or sets the value indicating that all the whitespace characters in the end of the source text should be removed.

CharactersToRemove	Specifies a list of characters that will be removed from the source text after transformation.
CharactersToReplace	Gets or sets characters and their replacements in a special packed format.
RemoveSpaces	Gets or sets characters and their replacements in a special packed format.
RemoveLetters	Gets or sets the value indicating whether symbols that are categorized as Unicode letters should be removed from the input text.
RemoveDigits	Gets or sets the value indicating whether to remove/keep all Unicode decimal digits.
ReplaceZerosWithOs	Gets or sets the value indicating whether Zero symbols '0' should be replaced with the capital 'O' symbol.
ReplaceOsWithZeros	Gets or sets the value indicating whether the capital 'O' symbols should be replaced with Zero symbols '0'.
Active	Indicates if at least one transformation option is different from its default value.
CleanEmails	Gets or sets the value indicating if Email cleaning module should be used.

Methods

ResetSettings()	Sets all transformation and cleansing options to their default values.
-----------------	--

ColumnTransformation Class

Namespace:	<code>dataadder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

This class describes all the transformations applied to a single input column. WordSmith settings, Regular Expression options, advanced Proper Case settings, etc. are added to current class.

C#
<code>public class ColumnTransformation : IColumnTransformation</code>

Examples

The following code example shows how to use `ColumnTransformation` class.

C#

Remarks

Constructors

<code>ColumnTransformation(String)</code>	Creates cleansing options class instance for a single input column transformation using <code>String</code> path to temporary folder that is used for intermediate operations.
---	--

Properties and Public Fields

<code>TransformationType</code>	Gets or sets input field type for transformation. Depending on this type additional output columns can be created.
<code>TransformationTypeSource</code>	Gets or sets field type that defined by the Prediction module.
<code>RegexSettings</code>	Gets regular expression output options.
<code>UseWordSmith</code>	Gets the value indicating whether WordSmith transformation is used for a column.
<code>WordSmithVisualizator</code>	Get an instance of <code>WordSmithVisualizator</code> necessary only for visualizing (GUI); otherwise should not be used.
<code>ProperCaseSettings</code>	Gets or sets additional proper case options. Can effect on French and Irish names. E.g. "McGregor" instead of "Mcgregor", "de la Croix" instead of "De La Croix".
<code>UseAddressVerification</code>	Gets the value indicating if any Address Verification field is selected as Type for the input column.

Methods

<code>ShallowClone()</code>	Creates a new instance of a <code>ColumnTransformation</code> and copies all the settings to it from the current transformation class object.
<code>ToString()</code>	Returns a textual representation of all the column transformations applied to input field.
<code>Dispose()</code>	Custom <code>IDisposable</code> interface implementation.

ColumnTransformationList Class

Namespace:	<code>dataadder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Describes a list of transformations defined for all the columns from input data source. Can be used even without any input data sources, just for describing any list of column transformations. An instance of such class can be transformed to `System.Data.DataTable` class and loaded from such table in the future.

C#
<code>public class ColumnTransformationList : ILoadableContainer2CoordsMapper, IDisposable</code>

Examples

The following code example shows how to use `ColumnTransformationList` class.

C#

Remarks

Constructors

<code>ColumnTransformationList(String)</code>	Initializes an instance of the <code>ColumnTransformationList</code> class specifying the path to the temporary folder for intermediate calculations.
---	---

Properties and Public Fields

<code>this[Int32]</code>	Gets or sets a single column transformation by its index in the transformation list.
<code>this [String]</code>	Gets a column transformation by column name.
<code>RecordCount</code>	Gets transformations count in the list.
<code>ColumnCount</code>	Gets all possible transformation type count.
<code>Count</code>	Gets the number of entries in the list.
<code>OnProjectChanged</code>	Holds an Action that is fired every time any transformation option is changed for a single data source.

Methods

<code>ToTable()</code>	Serializes transformation options to <code>DataTable</code> object.
------------------------	---

<code>FromTable(DataTable)</code>	Deserializes transformation options from <code>DataTable</code> object.
<code>Add(ColumnTransformation)</code>	Adds a column transformation to the holding list.
<code>Copy(ColumnTransformationList)</code>	Copies the column transformation settings from an instance of the <code>ColumnTransformationList</code> class to the current list. Existing list of transformations is cleared beforehand.
<code>Clear()</code>	Clears the contents of the list.
<code>IndexByName(String)</code>	Gets input field index by its name.
<code>GetData(Int32, Int32)</code>	Gets a single transformation option (second <code>Int32</code>) for a single transformation column (first <code>Int32</code>).
<code>SetData(Object, Int32, Int32)</code>	Sets <code>Object</code> atomic transformation option (second <code>Int32</code>) for defined transformation column (first <code>Int32</code>).
<code>GetColumnName(Int32)</code>	Returns transformation column name by its index <code>Int32</code> .
<code>AddMapperRow(DataRow, out Boolean)</code>	Adds a new empty column transformation to the list. Please note that <code>DataRow</code> parameter is ignored at the moment.
<code>AddMapperRow(out Boolean)</code>	Adds a new empty Column Transformation to the list.
<code>ClearMapper()</code>	Clears the list of column transformations.
<code>IsRowActive(Int32, Int32)</code>	Determines if a row in the grid corresponds with the field editor.
<code>Dispose()</code>	Releases the resources used.

FirstNameBlock Class

Namespace:	<code>dataadder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Responsible for parsing first names. Allows to define Common Name and Gender by First Name. Adds new columns with Common Name and Gender.

C#
<code>public class FirstNameBlock : TransformationBlock</code>

Examples

The following code example shows how to use `FirstNameBlock` class.

C#

Remarks

Constructors

<code>FirstNameBlock()</code>	Creates a new instance of FirstName transformation block and initializes outputs.
-------------------------------	---

Properties and Public Fields

--	--

Methods

<code>AddInput(DataFlow)</code>	Sets a new input column for current transformation. Previous input is removed.
<code>Process()</code>	Performs first name transformation.

FullNameBlock Class

Namespace:	<code>dataadder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Responsible for parsing full names. Produces first, middle, last, prefix, suffix, common name and gender.

C#
<code>public class FullNameBlock : TransformationBlock</code>

Examples

The following code example shows how to use `FullNameBlock` class.

C#

Remarks

--

Constructors

<code>FullNameBlock()</code>	Creates a new full name transformation block using the parser specified.
------------------------------	--

Properties and Public Fields

--	--

Methods

<code>AddInput(DataFlow)</code>	Sets a new input column for current transformation. Previous input is removed.
<code>Process()</code>	Performs full name transformation.

CleanBlock Class

Namespace:	<code>dataladder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Performs the upper case, lower case, reverse case and proper case transformation and other simple string operations.

C#
<pre>public class CleanBlock : TransformationBlock</pre>

Examples

The following code example shows how to use `CleanBlock` class.

C#

Remarks

Constructors

<code>CleanBlock(ColumnTransformation, Parsers.AbbreviationParser, ProperCaseOptions)</code>	Creates a new instance of Clean transformation block with transformation options <code>ColumnTransformation</code> , an instance of initialized <code>AbbreviationParser</code> and custom <code>ProperCaseOptions</code> .
--	---

Properties and Public Fields

<code>ColumnTransformation</code>	Holds column transformation options.
<code>NonPrintableCharacters</code>	Contains the list of Unicode characters that categorized as whitespaces (except regular whitespace symbol itself).

Methods

<code>SetInput(Int32, DataFlow)</code>	Assigns a data flow to input with the specified index and adds current transformation block to the output transformation block list.
<code>Process()</code>	Starts transformation process.

CopyColumnBlock Class

Namespace:	dataaladder.Data.DataTransformation
Assemblies:	DataMatch.Transformation.dll

Copies an existing column to a new named '*_original'.

C#
<pre>public class CopyColumnBlock : TransformationBlock</pre>

Examples

The following code example shows how to use [CopyColumnBlock](#) class.

C#

Remarks

Constructors

CopyColumnBlock(DataFlow)	Creates a new instance of CopyColumnBlock with input DataFlow .
---	---

Properties and Public Fields

--	--

Methods

Process()	Performs copy column transformation.
---------------------------	--------------------------------------

RegexBlock Class

Namespace:	DataMatch.Transformation.Blocks.RegexBlock
Assemblies:	DataMatch.Transformation.dll

Base abstract class for a transformation that uses regular expressions.

C#
<pre>public abstract class RegexBlock : TransformationBlock</pre>

Examples

The following code example shows how to use `RegexBlock` class.

C#

Remarks

Constructors

<code>RegexBlock(ErrorHandler.OnErrorDelegate, Int32)</code>	Creates a new instance of <code>RegexBlock</code> with error handler and ordinal index specified.
--	---

Constants

<code>RegexErrorOutputName</code>	Represents the regex error field name.
-----------------------------------	--

Properties and Public Fields

<code>Regex</code>	Holds the regular expression instance.
<code>RegexError</code>	Holds the error message occurred during regex parsing.
<code>RegexOrdinal</code>	Holds the absolute index of regular expression transformation for a single data source.

Methods

<code>Process()</code>	Starts transformation process (inherited from the abstract parent class).
------------------------	---

RegexBlockNative Class

Namespace:	DataMatch.Transformation.Blocks.RegexBlock
Assemblies:	DataMatch.Transformation.dll

Transformation block that assumes default regular expression parsing not using advanced output options. Output columns are created only for regular expression groups with names. Regular expression group names become column names of the transformed table.

C#
<pre>public class RegexBlockNative : RegexBlock</pre>

Examples

The following code example shows how to use [RegexBlockNative](#) class.

C#

Remarks

Constructors

RegexBlockNative (String , ErrorHandler.OnErrorDelegate , Int32)	Creates a new instance of RegexBlockNative with regular expression, error handler and ordinal specified.
---	--

Properties and Public Fields

Regex	Holds the regular expression instance.
RegexError	Holds the error message occurred during regex parsing.
RegexOrdinal	Holds the absolute index of regular expression transformation for a single data source.

Methods

Process ()	Perform transformation using regular expression block.
----------------------------	--

RegexBlockAdvanced Class

Namespace:	DataMatch.Transformation.Blocks.RegexBlock
Assemblies:	DataMatch.Transformation.dll

Represents a regular expression block that uses advanced output options. This class allows not only to parse out regular expression parts into new separated columns but combine these output columns into a single column and add custom pieces of static text.

C#
<pre>public class RegexBlockAdvanced : RegexBlock</pre>

Examples

The following code example shows how to use [RegexBlockAdvanced](#) class.

C#

Remarks

Constructors

RegexBlockAdvanced (RegexSettings , ErrorHandler.OnErrorDelegate , Int32)	Creates a new instance of advanced regular expression block using settings, error handler and ordinal specified.
---	--

Properties and Public Fields

RegEx	Holds the regular expression instance.
RegexError	Holds the error message occurred during regex parsing.
RegexOrdinal	Holds the absolute index of regular expression transformation for a single data source.

Methods

Process ()	Perform transformation using regular expression block.
----------------------------	--

RegexBlockFactory Class

Namespace:	DataMatch.Transformation.Blocks.RegexBlock
Assemblies:	DataMatch.Transformation.dll

Generates proper regular expression block depending on the usage of advanced options.

C#
<pre>public static class RegexBlockFactory</pre>

Examples

The following code example shows how to use [RegexBlockFactory](#) class.

C#

Remarks

Constructors

--	--

Properties and Public Fields

--	--

Methods

<pre>Create(RegexSettings, ErrorHandler.OnErrorDelegate, Int32)</pre>	<p>Factory method that creates an instance of a class derived from RegexBlock.</p> <p>RegexSettings - Defines advanced options like regular expression itself, a list of groups, output format, etc.</p> <p>OnErrorDelegate - Is fired when error occurs.</p> <p>Int32 - Ordinal number of regular expression.</p>
---	--

MergeBlock Class

Namespace:	<code>dataaladder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Allows to merge data from several input fields into a new single output field using a custom delimiter.

C#
<code>public class MergeBlock : TransformationBlock</code>

Examples

The following code example shows how to use `MergeBlock` class.

C#

Remarks

Constructors

<code>MergeBlock(String)</code>	Creates a new instance of a merge block with specified name.
---------------------------------	--

Properties and Public Fields

<code>Name</code>	Gets or sets the name of a new merged output column..
<code>Delimiter</code>	Gets or sets a delimiter to split data from different input columns.
<code>OnlyFirstNotEmptyCount</code>	Gets or sets the value indicating the limit of values to merge. All empty fields will be skipped, and not empty values will be merged until this limit is reached. Other values will be ignored.

Methods

<code>Process()</code>	Runs merging process.
<code>AddInput(DataFlow)</code>	Adds an input field that will be merged during transformation.

MergeBlockSingleStorage Class

Namespace:	dataladder.Data.DataTransformation
Assemblies:	DataMatch.Transformation.dll

Describes a single merge blocks. Such a block involves several input fields and only one output.

C#
<pre>public class MergeBlockSingleStorage</pre>

Examples

The following code example shows how to use [MergeBlockSingleStorage](#) class.

C#

Remarks

Constructors

<pre>MergeBlockSingleStorage(String, List<String>)</pre>	Creates a new instance of single merge block. String – Merge block name. This name is used for output column name. List<String> – The list of input field names that are used for merge.
--	--

Properties and Public Fields

MergedFieldNames	Holds a list of names of input fields that are used for merging.
this[Int32]	Gets the name of a merge field with defined index.
Name	Gets or sets the merge block name. This name is used for output column name.
Delimiter	Gets or sets the delimiter that is used for input fields data separation.
DelimiterEnum	Gets one of the predefined delimiters.
OnlyFirstNotEmptyCount	Gets or sets the value indicating the limit of values to merge.

Methods

AddFieldName(String)	Adds a new input field with unique name into the list of fields to merge.
RemoveFieldName(String)	Removes a field with the name String form the merge list.

MergeBlocksStorage Class

Namespace:	<code>dataladder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Describes information about one or more single merge blocks.

C#
<code>public class MergeBlocksStorage : IContainer2CoordsMapper</code>

Examples

The following code example shows how to use `MergeBlocksStorage` class.

C#

Remarks

Constructors

<code>MergeBlocksStorage()</code>	Default constructor.
-----------------------------------	----------------------

Properties and Public Fields

<code>this[Int32]</code>	Gets a single merge block storage by its index.
<code>Count</code>	Gets the total amount of merge blocks defined.
<code>RecordCount</code>	Gets the total amount of merge blocks.
<code>ColumnCount</code>	Always returns 1.

Methods

<code>Add(MergeBlockSingleStorage)</code>	Adds a new merge block.
<code>Remove(MergeBlockSingleStorage)</code>	Removes an existing merge block from the list.
<code>ChangeNameIfExists(MergeBlockSingleStorage)</code>	Guarantees uniqueness of merge blocks adding digits if column name already exists.
<code>NameAlreadyExists(MergeBlockSingleStorage)</code>	Checks if merge block with defined name already exists.
<code>Copy(MergeBlocksStorage)</code>	Copies the information about merge blocks into current instance from another source.
<code>ToTable()</code>	Serializes the information about merge blocks into <code>DataTable</code> class instance.
<code>FromTable(DataTable)</code>	Loads information about merge blocks from a <code>DataTable</code> into current instance.

<code>GetData(Int32, Int32)</code>	Gets the name of a merge block with the specified index. Note that the second parameter is ignored.
<code>SetData(Object, Int32, Int32)</code>	Updates merge block name with <code>Int32</code> index (first <code>Int32</code> parameter). The last argument is ignored.
<code>GetColumnName(Int32)</code>	Returns <code>'Name'</code> . <code>Int32</code> is ignored.

WordSmithBlock Class

Namespace:	<code>dataadder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Allows to perform complex replacements using replacement mapping tables. Phrase to remove, replace, replace with, operation priority can be defined in this kind of transformation.

C#
<pre>public class WordSmithBlock : TransformationBlock</pre>

Examples

The following code example shows how to use `WordSmithBlock` class.

C#

Remarks

Constructors

<code>WordSmithBlock(Dictionary<String, ReplacingStep>, Dictionary<String, ReplacingStep>, Int32, String)</code>	Creates the new instance of Word Smith block. <code>Dictionary<String, ReplacingStep></code> - A list of replacements. <code>Dictionary<String, ReplacingStep></code> - A list of replacements for which new column is defined. <code>Int32</code> - Max amount of words that can represent collocation. <code>String</code> - Any characters that will separate different words from Word Smith block.
--	---

Properties and Public Fields

<code>MaxWordCount</code>	Gets maximum number of words that can represent collocation.
<code>AllSeparators</code>	Holds word delimiters.

Methods

<code>AddInput(DataFlow)</code>	Adds a new input column Word Smith should be defined for.
<code>SetInput(Int32, DataFlow)</code>	Assigns a data flow to input with index specified and adds current transformation block to the output block list.
<code>Process()</code>	Performs Word Smith transformation.

<code>GetSortedReplacingSteps(Dictionary<String, ReplacingStep>, String, Int32, ref Char[], Boolean)</code>	<p>Sorts input steps by priority and length.</p> <p><code>Dictionary<String, ReplacingStep></code> - The list of replacements.</p> <p><code>String</code> - Input text which replacements should be applied to.</p> <p><code>Int32</code> - Maximum combined words count.</p> <p><code>ref Char[]</code> - Word separator.</p> <p><code>Boolean</code> - Include full text.</p>
---	---

AddressBlock Class

Namespace:	<code>dataladder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Represents a block used for standard address verification.

C#
<pre>public class AddressBlock : TransformationBlock</pre>

Examples

The following code example shows how to use `AddressBlock` class.

C#

Remarks

Constructors

<code>AddressBlock(CassAddress, CassGeoCoder, AddressVerificationSettings AddressParser)</code>	Creates an instance of Address Block <u>and sets the parser to use in the block.</u>
---	--

Properties and Public Fields

--	--

Methods

<code>AddInput(DataFlow, CassInputTypes)</code>	Adds the new input field to transformation. <u>Adds a data flow to the inputs of the block.</u>
<code>Process()</code>	Performs Word Smith transformation. <u>Performs the address block transformation.</u>

AddressCassBlock Class

Namespace:	<code>dataadder.Data.DataTransformation</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

This block can be used for address verification that uses CASS address database.

C#
<code>public class AddressCassBlock : TransformationBlock</code>

Examples

The following code example shows how to use `AddressCassBlock` class.

C#

Remarks

Constructors

<code>AddressCassBlock(CassAddress, CassGeoCoder, AddressVerificationSettings)</code>	Creates an instance of Address Cass Block specifying corresponding address, geocoder and settings objects.
---	--

Properties and Public Fields

--	--

Methods

<code>AddInput(DataFlow, CassInputTypes)</code>	Adds a new input field to the transformation.
<code>Process()</code>	Verifies the address using the database.

AddressVerificationSettings Class

Namespace:	<code>dataladder.AddressVerification</code>
Assemblies:	<code>DataMatch.Transformation.dll</code>

Used for customization of Address Verification module output fields. Included fields and their order can be defined using this class.

C#
<code>public class AddressVerificationSettings : ITableConvertible</code>

Examples

The following code example shows how to use `AddressVerificationSettings` class.

C#

Remarks

Constructors

<code>AddressVerificationSettings()</code>	Creates a new instance of address verification settings and includes all output fields using their default order.
--	---

Properties and Public Fields

<code>OutputColumnSettings</code>	Gets the list of address verification settings for each output column.
-----------------------------------	--

Events

<code>OutputSettingsChanged</code>	Occurs after any output setting changed.
<code>OutputSettingsChanging</code>	Occurs before any output setting changed.
<code>SomethingChanged</code>	Occurs when any parameter changed.

Methods

<code>ToTable()</code>	Transforms output column settings into tabular representation to have a possibility to save them in a project file.
<code>FromTable(DataTable)</code>	Loads output column settings from <code>DataTable</code> .
<code>GetOutputColumnMapper()</code>	Returns included output columns sorted by ascending order and mapped to their absolute indexes.

OnOutputSettingsChanged()	Fires the OutputSettingsChanged event.
OnOutputSettingsChanging(SettingsChangingEventArgs)	Fires the OutputSettingsChanging event.
OnSomethingChanged()	Fires the SomethingChanged event.
SettingsChanged()	Returns a value indicating whether any output column setting changed.
ForceOnOutputSettingsChanged()	Forcibly invoke event that occurs after any output setting changed.
DoOnOutputSettingsChanging(SettingsChangingEventArgs)	Forcibly invoke event that occurs before any output setting changed.
DoOnSomethingChanged()	Forcibly invoke event that occurs when any parameter changed.
GetColumnName(CassOutputParts)	Gets column name by full output column description.
GetDefaultOutputColumnMapper()	Gets all CASS output columns mapped to their index.

CassAddress Class

Namespace:	DataMatch.AddressVerification.Cass
Assemblies:	DataMatch.AddressVerification.dll

This class is used for verification of US and Canadian addresses by Primary Address, Secondary Address, Country, City, State, Zip, Company Name inputs. Verified addresses can be enriched with additional information like building, suggested city, etc.

C#
<pre>public class CassAddress</pre>

Examples

The following code example shows how to use [CassAddress](#) class.

C#

Remarks

Constructors

CassAddress()	Default constructor.
-------------------------------	----------------------

Properties and Public Fields

InputCompanyName	Input field contains Company or Firm name.
InputPrimaryAddress	Input column contains primary address.
InputSecondaryAddress	Input column contains secondary address.
InputCityName	Input column contains city name.
InputStateName	Input column contains state name.
InputZipCode	Input column contains US postal code (Zip code).
InputUrbanization	Denotes an area, sector, or residential development within a geographic area.
InputCountry	Column contains country name. (Used for Canadian address verification).
InputCanadianPostalcode	Canadian postal code (if Country field is defined and equals 'Canada') or US Zip code for US addresses.
Status	Result return code. Possible values: V - address verified; M - multiple response; N - address not verified.
ResidentialDeliveryIndicator	Residential Delivery Indicator (RDI).

	Possible values: Y = Residential Delivery; N = Not Residential Delivery; Blank = Did not query RDI.
ResultErrorExplanation	Textual explanation of address verification error code.
ResultCompanyFirm	Firm/Company.
ResultPrimaryAddress	Standardized Primary Address.
ResultSecondaryAddress	Secondary Address.
ResultCity	Standardized Preferred City.
ResultState	Standardized State Abbreviation.
ResultZipCode	Standardized ZIP Code.
ResultPostalCode	Standardized Postalcode (Canada Only).
ResultPlus4	+4 Code.
ResultDeliveryPointCheckDigit	Delivery Point and Check Digit.
ResultPlus6	Zip + 6.
ResultDeliveryPoint	Delivery Point.
ResultCarrierRouteCode	Carrier Route Code.
ResultLineOfTravelNumber	Line of Travel Number.
ResultLineOfTravelCode	Line of Travel Asc/Desc Code.
ResultAddressRecordType	Urbanization.
ResultStateFips	State FIPS Code.
ResultCountyFips	County FIPS Number.
ResultCountyName	County Name.
ResultCongressDist	Congressional District Number.
ResultPreferredCity	Standardized Preferred City.
ResultAbreviatedCity	Standardized Abbreviated City Name (if available).
ResultLastLine	Complete Standardized Last Line.
ResultLacsMatch	LACS Match.
ResultLacsLink	LACSLink Return Code.
ResultLacsLinkIndicator	LACSLink Indicator.
ResultSuiteLink	SuiteLink Code.
ResultReturnCode	Return Code. Possible values: 10 = Invalid Address; 11 = Invalid ZIP code; 12 = Invalid State Code; 13 = Invalid City; 21 = Address not found; 22 = Multiple response; 31 = Single response (Exact Match); 32 = Default response (Missing information - Ste #, or Invalid Ste #).
ResultErrorAndWarning	Warnings or Errors. Possible values: A# ZIP; B# City/State Corrected C# Invalid city/state/zip D# No ZIP assigned E# ZIP assigned for multiple response

	F# No ZIP available G# Part of firm moved to address H# Secondary number missing I# Insufficient/incorrect data J# Dual input K# Multi caused by cardinal rule L# Deliver address component add/del/chg M# Street name spelling changed N# Delivery address was standardized O# Low +4 tie-breaker (multi-response) P# Better delivery address exists Q# Unique ZIP Code R# No match due to EWS (Early Warning System) S# Invalid secondary number T# Multiple caused by magnet rule U# Unofficial Post Office name V# Unverifiable city/state W# Small town default X# Unique ZIP code generated Y# Military match Z# ZIP move match
ResultBuildingNumber	Parsed Primary Number.
ResultPreDirection	Parsed Pre-direction.
ResultStreetName	Parsed Street Name.
ResultPostDirection	Parsed Post Direction.
ResultSuffix	Parsed Suffix.
ResultSecondaryName	Parsed Unit Description.
ResultSecondaryNumber	Parsed Secondary Number.
ResultPMBIndicator	Private Mail Box Description.
ResultPMBNumber	Private Mail Box Number.
ResultVacancyFlag	Delivery Point Confirmation Indicators - DPV Vacancy Indicator.
ResultCountyFIPSCode	County FIPS Number.
ResultStateFIPSCode	State FIPS Code.
ResultCongressDistNumber	Congressional District Number.
ResultDPV	Delivery Point Confirmation Indicators. Combines the next result values into a single string: DPV Confirmation Indicator; DPV CMRA Indicator; DPV False Positive Indicator; DPV Vacancy Indicator; DPV No Stats Indicator.
ResultDPVFlag	Delivery Point Confirmation Footnotes. A combination of results of the next variables: 1) AA Input Address Matched to the ZIP + 4 file; 2) BB Input Address Matched to DPV (all components). If parameter (1) returns - adds 'AA'; If parameter (2) returns - adds 'BB';

	Example: "", "AA", "BB", "AABB".
ResultLACSFlag	LACSLink Return Code.
ResultLACSReturnCode	LACSLink Return Code.
ResultStelLinkInd	SuiteLink Indicator.
ResultDPVFootAA	AA Input Address Matched to the ZIP + 4 file.
ResultDPVFootA1	A1 Input Address Not Matched to the ZIP + 4 file.
ResultDPVFootBB	BB Input Address Matched to DPV (all components).
ResultDPVFootCC	CC Input Address Primary Number Matched to DPV but Secondary Number not Matched (present but invalid).
ResultDPVFootNA	N1 Input Address Primary Number Matched to DPV but Address Missing Secondary Number.
ResultDPVFootM1	M1 Input Address Primary Number Missing.
ResultDpvFootM3	M3 Input Address Primary Number Invalid.
ResultDPVFootRR	RR Input Address Matched to CMRA and PMB designator present (PMB 123 or #123).
ResultDPVFootR1	R1 Input Address Matched to CMRA but PMB designator not present (PMB 123 or #123).
ResultDPVFootP1	P1 Input Address PO, RR, or HC Box number missing.
ResultDPVFootP3	P3 Input Address PO, RR, or HC Box number Invalid.
ResultDPVFootU1	U1 Input Address Matched to a Unique ZIP Code.
ResultDPVFootG1	G1 Input Address Matched to a General Delivery Address.
ResultDPVFootF1	F1 Input Address Matched to a Military Address.
ResultDPVFootN1	N1 Input Address Primary Number Matched to DPV but Address Missing Secondary Number.
ResultDPVA	DPV Confirmation Indicator. Possible values: Y =Address was DPV confirmed for both primary and (if present) secondary numbers. D =Address was DPV confirmed for the primary number only, and Secondary number information was missing. S = Address was DPV confirmed for the primary number only, and Secondary number information was present but unconfirmed. N =Both Primary and (if present) Secondary number information failed to DPV Confirm.
ResultDPVC	DPV CMRA Indicator.
ResultDPVF	DPV False Positive Indicator.
ResultDPVV	DPV Vacancy Indicator.
ResultDPVX	DPV No Stats Indicator.
Initialized	Indicates whether CASS module has been initialized to run initialization part once.
AddrCode	An instance of CASS address validation module.
DataExpirationDays	Gets CASS Database license days left.
DllExpirationDays	Gets CASS Library license days left.
Version	Gets CASS Module version.
ReleaseDate	Gets the Release date of the CASS module.

Methods

<code>CheckExpirationDates(String)</code>	Performs checking of expiration dates of Address Verification Database and API Library. <code>String</code> - Path to address verification database.
<code>Init(String)</code>	Initializes Address Verification module. This method should be called before using CASS address verification. <code>String</code> - Path to address verification database.
<code>CASSLookup()</code>	Verifies a single address.
<code>Clear()</code>	Clears input address fields.
<code>Close()</code>	Releases the resources used.

CassGeoCoder Class

Namespace:	DataMatch.AddressVerification.Cass
Assemblies:	DataMatch.AddressVerification.dll

Allows to retrieve the Census Tract, Census Block Group, Latitude and Longitude by Zip+4 value (9 digits).

C#
<pre>public class CassGeoCoder</pre>

Examples

The following code example shows how to use [CassGeoCoder](#) class.

C#

Remarks

Constructors

CassGeoCoder()	Creates an instance of CassGeoCoder class and initializes Database.
--------------------------------	---

Properties and Public Fields

CensusTract	Gets Census Tract value, received after the request.
CensusBlockGroup	Gets Census Block Group value, received after the request.
Latitude	Gets Latitude value that corresponds to Zip+4.
Longitude	Gets Longitude value that corresponds to Zip+4.

Methods

Init(String)	Initializes and opens the CASS Geo Database that is located by the path String .
Lookup(String)	Call Address Lookup for input Zip + 4 String without dashes and spaces (9 digits).
Close()	Closes CASS Geo Engine.

CassParser Class

Namespace:	<code>DataMatch.AddressVerification.Cass</code>
Assemblies:	<code>DataMatch.AddressVerification.dll</code>

Special class that allows to verify addresses by incomplete input data and retrieve Street, Building Number, Preferred City, Standardized Zip Code, Latitude, Longitude, etc.

C#
<code>public class CassParser</code>

Examples

The following code example shows how to use `CassParser` class.

C#

Remarks

Constructors

<code>CassParser()</code>	Default constructor.
---------------------------	----------------------

Properties and Public Fields

<code>CassAddress</code>	Gets CASS Address module.
<code>CassGeoCoder</code>	Gets CASS Geo module required for geographical coordinates determination.

Methods

<code>InitCassIfNeeeded(String, String)</code>	Initializes the CASS address and geo modules if they're not initialized yet. First <code>String</code> - Path to CASS Address DB. Second <code>String</code> - Path to CASS Geo DB.
--	---

ICassRequest Interface

Namespace:	DataMatch.AddressVerification.Contracts
Assemblies:	DataMatch.AddressVerification.dll

Declares parameters for a request to CASS Address Verification module.

C#
<pre>public interface ICassRequest</pre>

Examples

The following code example shows how to use [ICassRequest](#) interface.

C#

Remarks

Properties

Query	Gets or sets Primary address for which 5 suggestions will be provided.
-----------------------	--

Methods

--	--

ICassResponse Interface

Namespace:	DataMatch.AddressVerification.Contracts
Assemblies:	DataMatch.AddressVerification.dll

Declares fields that are provided as a result of request by CASS Suggestions module.

C#
<pre>public interface ICassResponse</pre>

Examples

The following code example shows how to use [ICassResponse](#) interface.

C#

Remarks

Properties

AddressMain	Gets or sets Full address value.
AddressPrimary	Gets or sets Primary address value.
AddressSecondary	Gets or sets Secondary address value.
Country	Gets or sets Country value.
State	Gets or sets State value.
City	Gets or sets City value.

Methods

--	--

IAddressRequest Interface

Namespace:	DataMatch.AddressVerification.Contracts
Assemblies:	DataMatch.AddressVerification.dll

Provides address verification input data interface.

C#
public interface IAddressRequest

Examples

The following code example shows how to use [IAddressRequest](#) interface.

C#

Remarks

Properties

Address1	Gets or sets Main Address value.
Address2	Gets or sets Secondary Address value.
CompanyName	Gets or sets Company Name value.
City	Gets or sets City value.
State	Gets or sets State value.
PostalCode	Gets or sets Postal Code value.
Urban	Gets or sets Urban value.
Country	Gets or sets Country value.

Methods

--	--

AddressRequest Class

Namespace:	DataMatch.AddressVerification.Contracts
Assemblies:	DataMatch.AddressVerification.dll

Provides address verification input data interface.

C#
<pre>public class AddressRequest</pre>

Examples

The following code example shows how to use [AddressRequest](#) class.

C#

Remarks

Default `IAddressRequest` interface implementation.

Properties

Address1	Gets or sets Main Address value.
Address2	Gets or sets Secondary Address value.
CompanyName	Gets or sets Company Name value.
City	Gets or sets City value.
State	Gets or sets State value.
PostalCode	Gets or sets Postal Code value.
Urban	Gets or sets Urban value.
Country	Gets or sets Country value.

Methods

--	--

IAddressResponse Interface

Namespace:	DataMatch.AddressVerification.Contracts
Assemblies:	DataMatch.AddressVerification.dll

Provides Zip suggestion for input address data.

C#
<pre>public interface IAddressResponse</pre>

Examples

The following code example shows how to use [IAddressResponse](#) interface.

C#

Remarks

Properties

Zip	Gets or sets Suggested Zip code received by input parts of an address.
---------------------	--

Methods

--	--

AddressResponse Class

Namespace:	DataMatch.AddressVerification.Contracts
Assemblies:	DataMatch.AddressVerification.dll

Provides Zip suggestion for input address data.

C#
<code>public class AddressResponse</code>

Examples

The following code example shows how to use [AddressResponse](#) class.

C#

Remarks

Default IAddressResponse interface implementation.

Properties

Zip	Gets or sets Suggested Zip code received by input parts of an address.
---------------------	--

Methods

--	--

CassRequest ClassNamespace:	DataMatch.AddressVerification.Entities
Assemblies:	DataMatch.AddressVerification.dll

Declares parameters for a request to CASS Address Verification module.

C#
<code>public class CassRequest : ICassRequest</code>

Examples

The following code example shows how to use `CassRequest` class.

C#

Remarks

Constructors

<code>CassRequest(String)</code>	Creates a new instance of CASS request with defined Primary Address <code>String</code> .
<code>CassRequest()</code>	Creates a new instance of CASS request with default parameters.

Properties and Public Fields

<code>Query</code>	Gets or sets the parameter for Primary Address.
--------------------	---

Methods

--	--

CassManagerFactory Class

Namespace:	DataMatch.AddressVerification
Assemblies:	DataMatch.AddressVerification.dll

Used for creation of `CassManager` class instances. There is only one `CassManager` type now, but the list can be extended in the future.

C#
<code>public static class CassManagerFactory</code>

Examples

The following code example shows how to use `CassManagerFactory` class.

C#
<code>ICassManager cassManager = CassManagerFactory.Create(CassManagerTypes.Default);</code>

Remarks

Constructors

--	--

Properties and Public Fields

--	--

Methods

<code>Create(CassManagerTypes)</code>	Crates a new instance of <code>CassManager</code> . <code>Custom = Default</code> at the moment.
---------------------------------------	--

ICassManager Interface

Namespace:	DataMatch.AddressVerification.Entities
Assemblies:	DataMatch.AddressVerification.dll

Interface declaring of address autocomplete functionality. Provides suggestions for uncompleted full address, can define zip code by input Primary Address, City, State, Company Name, etc.

C#
<code>public interface ICassManager</code>

Examples

The following code example shows how to use [ICassManager](#) interface.

C#

Remarks

Constructors

--	--

Properties and Public Fields

--	--

Methods

GetAddressSuggestions (ICassRequest)	Returns an array of first 5 address suggestions by incomplete primary address.
GetAddressSuggestions (String)	Returns first 5 address suggestions by incomplete primary address as JSON.
ValidateAddress (IAddressRequest)	Validates single address by input data and provides missing Zip code.

CassManager Class

Namespace:	DataMatch.AddressVerification.Entities
Assemblies:	DataMatch.AddressVerification.dll

Default implementation of [ICassManager](#) interface.

C#
<pre>public class CassManager : ICassManager</pre>

Examples

The following code example shows how to use [CassManager](#) class.

C#

Remarks

Constructors

--	--

Properties and Public Fields

--	--

Methods

GetAddressSuggestions (ICassRequest)	Returns an array of first 5 address suggestions by incomplete primary address.
GetAddressSuggestions (String)	Returns first 5 address suggestions by incomplete primary address as JSON.
ValidateAddress (IAddressRequest)	Validates single address by input data and provides missing Zip code.