matchIT API Guide

Table of Contents

Introduction

The matchIT API is a component written by helpIT systems for state of the art fuzzy matching, data formatting and data cleansing – amongst its most common uses are duplicate prevention, inquiry, deduplication and merge/purge. The matchIT API splits and cases names and addresses, generates match keys, and grades matching records. The component provides a compact and efficient solution to the problems of data quality and duplication on any Windows based system. The matchIT API is also available in common variants of Unix and Linux, using shared libraries.

This is the help file for the COM version of the matchIT API. This document assumes that you have familiarity with at least one Windows-based programming language. Experience with the utilization of COM components from within programs would be an advantage, but not essential. If you have any questions, please contact us and we will be glad to help you.

Overview

There are two fundamental parts to the matchIT API:

- record generation
- record comparison.

These can be utilized in five different scenarios:

- data entry not connected to the target database e.g. in web forms
- on-line lookup e.g. customer inquiry
- data capture incorporating duplicate prevention
- single file matching e.g. deduplication
- cross file matching e.g. data load.

Looking at these scenarios, it is obvious that the first scenario, data entry not connected to the target database, does not require record comparison. Therefore, for formatting, standardization and screening of new data independent of data that you already hold, you only need to incorporate the record generation aspect in your application. This will allow you to e.g. parse names into their component parts, relocate floating data to fixed fields, standardize abbreviations and expansions and allow you to screen for garbage and abusive entries.

The remaining scenarios require both record generation and record comparison to be effective. Record generation allows you to group records intelligently for comparison, by generating phonetic and non-phonetic match keys for each record. Record comparison grades each pair of matching records with a match score, which allows you to fine tune the matching and give the user control of the level of matching that they use.

Because understanding the use of *match keys* and *match scores* is fundamental to effective use of the matchIT API for the last four scenarios listed above, we will next describe how and why they are used.

Match Keys

The matchIT API allows you to generate phonetic “match keys” from names and addresses, as well as non-phonetic standardized data. In this context, a match key is something that groups of records within the database have in common, which indicates that detailed comparison of the records is worthwhile to see how well the records match each other in other respects.

In an ideal world, a user would be able to compare every single record in a database with every inquiry, or with data entered on a data capture screen, or with every other existing record in the database. For internal matching on a 10,000 record database, this would mean you would do roughly 10,000*5,000 i.e. 50 million comparisons - even on this fairly small database, the processing time would be far too long. For inquiries or data entry, looking at every record on the existing database would take far too long for a real time application.

To reduce the number of comparisons, we can look at a field and say that only records that have the same value in that field are potential matches. For example, we could select the Lastname field (if the last name is held separately from the title and first names or initials), and only compare records where people have the same last name. This would dramatically reduce the number of comparisons we would have to do because, for example, we would not compare a record for Mr Smith to a record for Mr Jones. We would compare all the Smiths, because two records for Mr Smith might be for the same person. In this approach, we have used the Lastname field as a match key.

This approach is an improvement on our original idea, but it has its limitations. For example, take the name Deighton – people may expect it to be spelt Dayton, as this spelling is far more common than Deighton. The two spellings sound the same, but our solution above would not compare Mr Dayton and Mr Deighton, because the Lastname field is different. The obvious solution to this is to use a "sounds like" version of the Lastname field as a phonetic match key. The matchIT API takes important fields (such as name, address, company), and generates phonetic versions of the key elements in those fields.

However, even this approach still gives us too many records to compare, most of the time. For example, if you are comparing records across the whole of the US, there are a lot of Smiths – in a file of 100,000 records there would on average be nearly 750 Smiths, which would involve over a quarter of a million comparisons if processing internal matching of the database, or 750 real time comparisons for inquiry or data entry. To get round this, we use combinations of fields to narrow the search – Mr Smith in Boston is obviously not the same person as Mr Smith in Dallas. In the example above, we may choose something like phonetic key of last name plus part or all of the postal code – so now we only compare two records for Smith if they live in the same area. This is a more explicit match key than just phonetic last name.

The last point to consider is that when looking for matches is that you can never rely on any single item of data always being the same in order to find all the matching records there are - you need to do at least two scans of the database using different match keys. If, for example, two records were identical except that one record did not have a postal code, you would miss the match if you relied just on phonetic key of last name plus the postal code - because the records would not be compared unless the postal code as well as the phonetic last name were the same. However, if the street in both the records is the same (which is quite likely, if they are really matching records), then a second match key of phonetic last name plus the phonetic key of the most significant word(s) in the street will find the match on the second scan of the database. For data entry, this may mean that a potential duplicate of a new record would not be detected until the address has

been entered, if the postal code held on the record on the database is different from that being entered – but most duplicates would be detected before entering the address.

For most data files, we recommend three match keys - for US data, our third default key is a phonetic key of city and street together with the street or apartment number. For UK data, it is the postal code on its own. These keys usually give reasonably small groups of records to compare and allow non-phonetic last name matches to be detected. For example, you could pick up a match of Wilson and Wislon, as long as the postal code in the two records is identical and the address is effectively the same.

Match Scores

Having established that two records have the same match key, or that a record being entered or sought has the same match key as an existing record on the database, you can use the matchIT API to go through the data, field by field, and work out how similar they are. Each field can contribute to a match score, depending on how similar those fields are between the two records, or between the new data and the existing record. At the end, we have an overall score that tells us how alike two records are – by default, the higher the score, the more similar the records are.

For batch processing, when flagging duplicates or merging files in a batch process, you can ask the user to enter a threshold score, above which your system will automatically flag one record from all matching pairs. With most data files, all pairs scoring above a particular score will be genuine duplicates ("true matches") and anything below a lower score will be "false matches" – this leaves a "gray area" between these two scores where most of the pairs are true matches but some are false. For a cautious approach (“underkill”), you can therefore enter a higher score as a threshold score for deletion. For overkill, you can enter a lower score. Alternatively, you can go for underkill and then visually inspect the matching pairs in the gray area - there are usually relatively few in this area.

This process works well, but it is essential that system designers can tune it themselves. This is because matching requirements can vary from company to company, file to file and even job to job. For example, sometimes you want to match at individual level, sometimes at company or family or address. In addition, data files vary widely in their structure and overall "shape" of the data" – sometimes postal codes are reliable and low level, sometimes they are unreliable, or they only indicate the town/city and not the street.

Because we know that everyone's data is different, we have allowed the way that two records are compared to be customized, using a parameter table that tells us how much each field contributes to the overall matching score. Using this table, we can tell the matchIT API how important each field is in the matching process. We call this the Weights table, as it reflects the relative weighting that each field has towards the total match score. For name and address matching, the matchIT API compares the elements of the name (or address) as a whole, rather than just comparing them element by element - this allows it to match names where some of the components are omitted or in a different order in one record (e.g. John Michael Smith and Mike Smith) or addresses which have a house or building name in one record but not in another.

For more information about matching weights and a list of the default weights, please refer to the matchIT API FAQ’s.

System Design

The first point to consider when designing a database that uses the matchIT API for data cleansing and/or matching, is if and where you want to store match key fields on your system, and (if so) which ones. Note that you can also use key fields of your own as match keys e.g. date of birth, account number. Whether you are generating match keys or not, the matchIT API will clean and tidy up data you supply to it – you may need to provide additional fields for e.g. salutation, company name, etc. if you want to take advantage of the name parsing and reformatting functions available in the API.

Record Generation

The matchIT API should usually be run and the results stored for every record in your database before you can perform on-line duplicate prevention and/or inquiry. If you have existing data, then this record generation process should be performed in a batch run - you will need to write a batch processing routine that calls the matchIT API to perform the task. Notes on which fields should be included and which may need to be supplied can be found in the Record Class section of this document. The name keys returned by the matchIT API are loose forms of the generated phonetic keys. However, there are some circumstances e.g. well structured consumer data files with good postal code quality and last name stored in a separate field, where storing phonetic keys on the database may not be absolutely necessary. For more advice on this, please contact us.

Record Comparison

The developer using the matchIT API for comparing records must supply the two records that constitute the candidate pair, as this operation is very specific to the database system and platform being used, but the record generation process helps the developer filter candidate pairs. Comparing records involves more work for the calling program than generating keys, because the calling program must retrieve records from the source database. There are four distinct ways of using the matchIT API for comparing records:

- Batch matching within a single file
- Comparing across files
- Online data capture
- Online inquiry

Batch matching within a single file

This is the simplest use of the matchIT API – the steps you need to perform are as follows:

1. Generate match keys for the table – using Engine.Generate( ).
2. Choose a match key.
3. Create an index on your table based on the match key (or use e.g. a SQL Select statement to return groups of records with the same key).
4. Read the table in index order (or read through a group of records with the same key) and for each and every pair of records with the same value of match key:
    - Compare the two records – using Engine.Compare( ) – to find a matching score;
    - If the two records are good enough matches, store their reference numbers and match score.

For example, if there are 4 records with the same match key value, you must compare records 1 and 2, 1 and 3, 1 and 4, then go back to record 2 and compare records 2 and 3, 2 and 4, then go back to record 3 and compare records 3 and 4 – thus there are 6 comparisons for this group of 4 records.

NB: If there are an unexpectedly large number of records with the same match key value, the process will take a long time for that key – so you may want to program in a limit above which the calling program reports the number of records for that key value and proceeds with the next key. This can happen with bad or skeletal data e.g. data with missing address information. We have found that to allow for this type of data, employing an approach using indexes gives more control and visibility than using SQL Select.

5. Move to the next record with a different match key value and repeat the process until end of file.

Repeat the steps above for each match key, then report the results and allow the user to e.g. interactively delete duplicates or delete all duplicates that score above a given threshold.

Comparing across files

For two files, this process is essentially the same as batch matching within a single file, but you are more likely to have different requirements after the matching e.g. merging files, transferring of data between matching records etc. – this is made easy if you have stored the reference numbers of the matching pairs. The steps you need to perform are as follows:

1. Generate match keys for both tables – using Engine.Generate( ).
2. Choose a match key.
3. Create an index on the larger table (or both tables), based on the match key. Although you can use e.g. a SQL Select statement if preferred, the rest of this section assumes that you are using an index.
4. Read the smaller table either in physical order, or in order of the match key. The advantage of reading it in order of the match key is that you can display progress to the user in terms of match key values as well as numbers of records, so they can see what kinds of match key cause the system to run more slowly.
5. For each record read, seek each record from the larger table which has the same value of match key as the current record from the small table. For each pair thus retrieved:
    - Compare the records from the two files – using Engine.Compare( ) – to find a matching score;
    - If the two records are good enough matches, store their reference numbers and match score.
6. Move to the next record with a different match key value and repeat the process until end of file.

Repeat the steps above for each match key, then report the results and allow the user to merge or purge, transfer data, etc.

Online data capture

To encourage the user to use the duplicate prevention features of any data capture system, it must:

- save time;
- only display records that are good matches.

The matchIT API ensures the latter objective and (if the record exists on the system) can achieve the first objective as well. The best way to ensure that the first objective is achieved is to structure the data capture screen so that you ask the user to enter the most useful lookup data first e.g. name or company name and postal code. Then you can do a lookup on keys drawn from these data items and if matching records are found, allow the

user to select one – which saves them time entering the address and other information, as well as stopping them from unwittingly creating a duplicate record.

The matchIT API also allows you to perform a "failsafe" check for duplicates after the address etc. has been entered, because you now have more data to match on and can stop the user from creating a duplicate even if the zip or postal code was significantly different – as long as the name and address match well.

The steps in each case are as follows:

1. Choose a match key or match keys (as described for Online inquiry);
2. Set an index on your existing data table that corresponds to the match key;
3. For the data entered, generate match key(s) using Engine.Generate( );
4. Look for all records in the existing data table whose fields match your chosen match key and for each of these:
    - Compare with the new record using Engine.Compare( );
    - If any pairs score highly enough, save their reference numbers and match score;
    - Repeat until no more records match the new one.
5. If more than on match key is possible, repeat steps 2-4 for each match key;
6. If any matching records are found, display them to the user with the highest scoring records first, so that they can choose whether to use an existing record, or to add the new record.

Online inquiry

This is different from data capture, because the user expects to find a record and he/she will enter only the minimum information that they expect to need to identify the right record. The steps are as follows:

1. Choose a match key according to the information entered. Depending on what the user searches on and the size of the database, it may be possible to use more than one match key e.g. use name on its own in a smaller file or where the name is unusual, but then look up name and company (or name and town/city) as well, to allow for non-phonetic keying errors in the name. In this way, you could return John Dell at Guthries Inc when the user enters John Bell at The Guthrey Group;
2. Set an index on your existing data table that corresponds to the match key;
3. For the search criteria entered generate match key(s) using Engine.Generate( );
4. Look for all records in the existing data table whose fields match your chosen match key and for each of these:
    - Compare with the new record using Engine.Compare( );

- If any pairs score highly enough, save their reference numbers and match score;
- Repeat until no more records match the new one.

5. If more than one match key is possible, repeat steps 2-3 for each match key;
6. Display the matching records (and scores, if you wish) that reach a required minimum score to the user, with the highest scoring records first. If no matching records meet this minimum, you can lower the threshold score so that you only show the fuzzier matches when there are no good matches – this avoids displaying records that don't match very well, if a good match is found.

Classes, Properties, and Methods

The matchIT API consists of a number of classes. These classes are listed and described here along with their properties and/or methods.

Properties are listed according to hierarchical organization. For example, the EngineSettings class has a property Jobtitle, accessible using Engine.Settings.Generate.Organization.Extract.Jobtitle.

Class	*Description*
Engine	Provides the core interface for using and configuring the matchIT API.
EngineSettings	Contains *all* settings used by the Engine class.
MatchingSettings (*)	Contains settings used when calculating final matching scores.
NameMatchingMatrix (*)	Contains the lookup matrix used when translating raw name matching results (e.g. sounds-approx) into processed name matching results (e.g. 0.333).
OrganizationMatchingMatrix (*)	Contains the lookup matrix used when translating raw organization matching results into processed organization matching results.
NamesAndWords	Contains the data lookup tables. See Names and Words Table for more information.
InputFields	Contains the input data when generating match keys.
Record	Contains either the generated output data, or the input data when comparing two records.
CompareResult	Contains the results obtained from comparing two records.
MultiLevelMatchingResults	Contains the comparison results after processing with the relevant matching matrix.
Scores	Contains the final matching scores as products of the processed results and the engine's weight values.

* Note that these are classes that cannot be instantiated, as they're contained entirely within the EngineSettings class and are only accessible via properties of it. The matchIT API also contains a number of other classes that cannot be instantiated, but these are only used to provide the hierarchical property structures and can therefore be ignored.

Engine class

This class provides the core interface for using and configuring the matchIT API.

Before the engine can be used, an instance must first be created and then the Initialize( ) method called.

The engine provides methods including Generate( ) – for generating match keys for searching databases – and Compare( ) – for comparing two records.

The engine's configuration settings are structured in a hierarchical organization accessible via the Settings property (which is an instance of the EngineSettings class).

Engine Properties

Properties	*Type*
Settings	(see EngineSettings class)
Tables	(see NamesAndWords class)
LastError	Text indicating any licencing or deprecation error.

Engine Methods

Initialize(NamesAndWords)

The engine must be initialized before using any other method or property. A NamesAndWords object must be created and initialized before being passed into Engine.Initialize( ).

Phoneticize(String, PhoneticAlgorithm) = String

Returns a string containing the phonetic representation of the word passed in.

ProperCase(String) = String

Returns a string containing the proper cased version of the passed string.

TypeOf(String) = FieldType

The string passed into this method is analyzed and its type (e.g. name, address, postal code, etc.) is determined and returned (FieldType is an enumeration, see Appendix A Enumeration C).

Generate(InputFields, Record)

This method processes the input fields and outputs a record object consisting of generated match keys and normalized input data, which is used for finding records in a database.

GenerateEx(InputFields, Record, EngineSettings)

See the Generate( ) method above.

Note that this method will use the supplied settings instead of the Engine's current settings (which are not affected).

Compare(Record, Record, CompareResult)

This method compares two records. The results are output in the CompareResult object that must first be created and then passed into this method. The results can then be passed to Engine.ApplyMatchingRules( ) for further processing.

CompareEx(Record, Record, CompareResult, EngineSettings)

See the Compare( ) method above.

Note that this method will use the supplied settings instead of the Engine's current settings (which are not affected).

ApplyMatchingRules(CompareResult, MultiLevelMatchingResults)

This method takes the raw comparison results obtained from Engine.Compare( ), and outputs a series of results calculated in conjunction with the matching matrices. These individual results are real numbers in the range 0 to 1, indicating the likelihood that components within the compared results are a sure match.

Score(MultiLevelMatchingResults, Scores)

This method takes the processed results from Engine.ApplyMatchingRules( ) and outputs a populated Scores class. Each resultant score is the product of each individual multi-level matching result and the relevant weight (as set in e.g. Engine.Settings.MatchingRules.IndividualLevel.Weights.Name.Weight).

EngineSettings class

This class provides the core interface for configuring the matchIT API.

Usually, the matchIT API is configured by using the properties of the Engine.Settings object.

Alternatively, an EngineSettings object can be created and passed into the Engine.GenerateEx( ) and Engine.CompareEx( ) methods to use different settings without affecting the Engine's current settings.

EngineSettings Properties

Properties	*Type*
NationalityOfData	Country, see Appendix A: Enumeration A
Generate	
.Address.AbbreviateRegion	Boolean
.Address.DefaultThoroughfareLine	Integer
.Address.Extract.Country	Extraction, see Appendix A: Enumeration B:
.Address.Extract.Postcode	Extraction, see Appendix A: Enumeration B:
.Address.Extract.Premise	Extraction, see Appendix A: Enumeration B:
.Address.Extract.Region	Extraction, see Appendix A: Enumeration B:
.Address.Extract.Thoroughfare	Extraction, see Appendix A: Enumeration B:
.Address.Extract.Town	Extraction, see Appendix A: Enumeration B:
.Address.Extract.PostTownsOnly	Boolean
.Address.NumOfLinesToScan	Integer
.Address.PremiseFirst	Boolean
.Address.UpperCaseTown	Boolean
.Address.VerifyPostcode	Boolean
.ConsiderCasing	Boolean
.DropExcludedWords	Boolean
.Name.ContactFullName	Boolean

Properties	*Type*
.Name.DefaultGender	Gender, see Appendix A: Enumeration D
.Name.DefaultSalutation	String
.Name.DetectInverseNames	Boolean
.Name.EnhancedDoubleBarrelledLookup	Boolean
.Name.GenerateContact	Boolean
.Name.JoinMarriedPrefixes	Boolean
.Name.ParseAsNormalizedName	Boolean
.Name.ParseNameElements	Boolean
.Name.ProcessBlankLastName	Boolean
.Name.ReplaceAndWithAmpersand	Boolean
.Name.UseEquivalentNames	Boolean
.NormalizationDelimiter	Char
.Organization.Extract.Jobtitle	Extraction, see Appendix A: Enumeration B:
.Organization.Extract.Name	Extraction, see Appendix A: Enumeration B:
.Organization.IgnoreParentheses	Boolean
.Organization.IgnoreTrailingPostTown	Boolean
.Organization.JoinInitials	Boolean
.Organization.NormalizationTruncation	Integer
.Organization.UseEquivalentName	Boolean
.ProperCase	Boolean
.Quality.Enabled	Boolean
.Quality.Address.AllowBlankPostcode	Boolean
.Quality.Address.Country	Boolean
.Quality.Address.MaxRepetition	Double
.Quality.Address.Premise	Boolean
.Quality.Address.Region	Boolean
.Quality.Email.MaxRepetition	Double
.Quality.Email.WebmailFiltering	Boolean
.Quality.Name.MaxRepetition	Double
.ReportUnrecognisedWords	Callback
.SpecialCaseMac	Boolean

Properties	*Type*
.VariableKeysMaxLength	Integer
Compare	
.Address.DefaultDeliveryPoints	String
.Address.DefaultThoroughfareLine	Integer
.Address.IgnorePremiseSuffix	Boolean
.Address.LooseFuzzyPremiseMatch	Boolean
.Address.MatchBoxNumberAndPostcode	Boolean
.Address.MatchDeliveryPoints	Boolean
.Address.MatchDeliveryPointsThreshold	double
.Address.UsePremiseRange	Boolean
.Name.FuzzyMatchNonNormalizedName	Boolean
.Name.OrganizationMatchingOnBlankNames	Boolean
.Name.PreventMrsMatchingMiss	Boolean
.Phonetic.Algorithm	PhoneticAlgorithm, see Appendix A: Enumeration F
.Phonetic.AlgorithmForFirstNames	PhoneticAlgorithm, see Appendix A: Enumeration F
.LooseThresholdForDynamicSoundIT	Integer
MatchingRules	
.IndividualLevel	(see MatchingSettings class)
.FamilyLevel	(see MatchingSettings class)
.HouseholdLevel	(see MatchingSettings class)
.BusinessLevel	(see MatchingSettings class)
.CustomLevel	(see MatchingSettings class)

EngineSettings Property Usage

EngineSettings.NationalityOfData

This property primarily influences processing of the POSTCODE field:

- it works in conjunction with the Settings.Generate.Address.Extract.Postcode property to determine whether a string in the address is determined to be a postal code and should be moved to the Postcode field.
- it works in conjunction with the Settings.Generate.Address.VerifyPostcode property to determine whether a string in the address or Postcode field has an alphanumeric error e.g. 5 when it should be S.
- it works in conjunction with the Postcode.Weight matching property (e.g. Settings.MatchingRules.IndividualLevel.Weights.Postcode.Weight) for matching: different countries have different standards for how specific a postcode is e.g. one per town, one per street, several per street; hence two full UK postcodes that match will get a higher score than two 5 digit US zip codes that match.

EngineSettings.Generate…

The following settings are used when the matchIT API *generates* a record (i.e. for key generation, and data standardisation and casing):

EngineSettings.Generate.Address.AbbreviateRegion

Set this property to True if you want the matchIT API to abbreviate States or Provinces when processing address lines e.g. to change “Pennsylvania” to “PA”.

EngineSettings.Generate.Address.DefaultThoroughfareLine

This property is used when the matchIT API is generating a phonetic address key, for which it needs to know the thoroughfare (e.g. street) and the town in the address. If it cannot locate a thoroughfare in the address, usually because it cannot find a word to indicate one, such as “Street”, then the API will assume that the thoroughfare is the contents of the address line indicated by this property (if it is greater than zero). For example, if this property is set to 2, then the API will take the contents of address line 2 as the thoroughfare if it cannot find a thoroughfare word in the address. This property should only be used if the addresses in your data are very rigidly structured.

EngineSettings.Generate.Address.Extract

This group of properties applies to Country, Postcode, Premise, Region, Thoroughfare and Town. It enables different types of data to be moved or copied from specific fields (in most cases the address lines) to designated fields that were explicitly added to store that type of data. These properties can be used to greatly improve the structure and consistency of the data, which can inherently improve results obtained during the Compare stage.

Each of these properties can either be set to “MoveExtract”, “CopyExtract” or “Leave”. If the property is set to “MoveExtract”, the corresponding data with will be moved from its original field into the designated field. When set to “CopyExtract”, the data will be copied into the designated field, and will still remain in the original field. When set to “Leave”, the corresponding data will not be copied or moved.

Each of these properties will be ignored if the relevant Record property required to copy or move the data into does not exist.

If the *Settings.Generate.ProperCase* property is set to True, moved and copied data (with the exception of premise (i.e. building) numbers and postcodes) will be correctly cased.

The different properties in this group are listed below with a description of data that each corresponds to, and any other information specific to that property…

EngineSettings.Generate.Address.Extract.Country

This will move or copy valid countries found in the address lines (based on Country type entries found in the NAMES.DAT file) into a field labeled 'COUNTRY'.

EngineSettings.Generate.Address.Extract.Postcode

This will move or copy UK postcodes, or US zip codes found in the address lines into a field labelled POSTCODE.

Only UK postcodes with an outward half that is valid according to the MAILSORT.DAT file will be extracted. It is advisable to set this property to "MoveExtract" rather than "CopyExtract" if using this data for mailing or updating a database.

EngineSettings.Generate.Address.Extract.Premise

This will move or copy premise numbers found in the address lines into a field labelled PREMISE.

It is not advisable to Extract the premise if you want to output the updated address later, as the API will not know which address line the premise number came from. It can however be useful to copy premise numbers, as this enables the inclusion of the PREMISE field as part of the match keys used during the compare stage, which can increase efficiency and accuracy when working on large files, or files containing very localized data.

EngineSettings.Generate.Address.Extract.Region

This will move or copy US, Canadian or Australian states or provinces, or valid UK counties (or other regions found in the NAMES.DAT file), that are found in the address lines into a field labelled REGION.

EngineSettings.Generate.Address.Extract.Thoroughfare

This will move or copy address data recognized as the thoroughfare of the address (based on Address type entries found in the NAMES.DAT file) into a field labelled THOROUGHFARE.

EngineSettings.Generate.Address.Extract.Town

This will move or copy address data recognized as the town or city from the address lines to a field labelled TOWN.

EngineSettings.Generate.Address.Extract.PostTownsOnly

If this is enabled, together with Settings.Generate.Address.Extract.Town, then only post towns (i.e. any towns found in the TOWNS.DAT file) will be moved or copied.

EngineSettings.Generate.Address.NumOfLinesToScan

This property enables personal names to be extracted from address lines. It can be set to 1 or 2. If set to 1, only the first address line will be scanned for names. If set to 2, both the first and second address lines will be scanned and have names extracted from them if found. Any personal names found can then be used for the generation of Contacts and Salutations.

If either or both of the *Settings.Generate.Organization.Extract.Jobtitle* and *Settings.Generate.Organization.ExtractName* properties are used in conjunction with this one, the matchIT API will not only scan the ADDRESSEE field for job titles and business names, but will also scan the corresponding number of address lines

EngineSettings.Generate.Address.PremiseFirst

When parsing an address, this Boolean property indicates whether to expect the premise or flat number to come first in address lines when the flat is not explicitly specified (e.g. “Flat 5”).

EngineSettings.Generate.Address.UpperCaseTown

This applies to UK addresses only. Set this property to True to convert the post town in the address to capitals. Note that, if the *Settings.Generate.ProperCase* property is set to False, then this property is ignored.

EngineSettings.Generate.Address.VerifyPostcode

If set to True, this property verifies and corrects the format of the postcode. Numerics are changed to alphas and vice versa where appropriate. This feature makes use of the rules concerning the alphanumeric structure of the postcode. e.g. it changes “KT22 BDN” to “KT22 8DN” – it will change 0, 1, 5 and 8 to O, I, S and B, or vice versa, if that makes the postcode alphanumerically correct. The matchIT API will not verify or correct the format of postcodes that are not in the postcode field.

EngineSettings.Generate.ConsiderCasing

If this property is set to True, then the matchIT API will consider the casing of the incoming data when it is splitting the data up for extracting keys, proper casing, and so forth. This is mainly used for the extraction of name data. For instance, consider the name:

EngineSettings.Generate.DropExcludedWords

With this property set to True, during the generate step the matchIT API will flag any records that contain exclusion words in any of the key fields (fields such as addressee, company or the address lines). Such exclusion words include “Deceased”, “Addressee” (indicating a record may be a header record) and any other Exclusion type entries in the NAMES.DAT file. Records are flagged by setting the first character of the *Record.DataFlags* property to “X”.

EngineSettings.Generate.Name.ContactFullName

Set this property to True to include the full first name of any incoming name in the CONTACT field; just the initial will be used if the property is False. For example, if the property is True, and the incoming name is “John Smith”, then the generated contact will be “Mr John Smith”, if it is False, then the contact will be “Mr J Smith”.

EngineSettings.Generate.Name.DefaultGender

The Default Gender property is the gender to assume when the matchIT API can’t determine whether the name is male or female e.g. Chris Smith, C Smith. If you set this property to Male or Female, the API will assume it to be male or female accordingly, and develop a salutation using Mr or Ms as the prefix.

EngineSettings.Generate.Name.DefaultSalutation

This property determines the default salutation, either where the API can't determine one (for example, C Smith or Chris Smith, which could be either Mr or Ms), or where the Prefix supplied doesn’t have a salutation rule. If you include the word ”Dear” as at the start of the default salutation (i.e. actually specify "Dear Customer" and not just "Customer", then all the salutations derived by the matchIT API will start with the word "Dear" unless the salutation for the type of title (or prefix) specifies "Title" only. For example, Mr J Smith will result in a salutation of "Dear Mr Smith" whereas The Bishop of Liverpool will result in a salutation of "My Lord".

EngineSettings.Generate.Name.DetectInverseNames

With this property enabled, the matchIT API will attempt to identify addressee names that have been specified with the lastname preceding the firstnames, provided a comma delimiter follows the lastname (for example, “Smith, John” where Smith is the lastname). Without a comma, a name is assumed to be in standard left-to-right format, with the firstnames preceding the lastname.

This setting is disabled by default.

EngineSettings.Generate.Name.EnhancedDoubleBarrelledLookup

When enabled, this property will cause an unrecognised middle name to be considered part of a non-hyphenated double-barrelled last name (for example, where the full name is John Harrington Jones, the last name will be considered Harrington -Jones because Harrington is not a recognised first name).

EngineSettings.Generate.Name.GenerateContact

With this property set to True, the matchIT API will generate a contact for the input name. The contact will be structured in same way as you would expect to find its corresponding input name on e.g. the front of an envelope. For example, the input name of “John Smith” or “Mr Smith” would result in a generated contact of “Mr J Smith”.

An accurate contact value cannot be generated when the matchIT API is unable to determine the gender of a input name. In this situation, the generated contact would be equal to the input name. e.g. “J Smith” as an input name would result in a generated contact of “J Smith”.

EngineSettings.Generate.Name.JoinMarriedPrefixes

With this property set to True, multiple addressees with the same last name will be treated as married e.g. input names of “Mr. John Smith and Ms. Mary Smith” or “Mr John Smith & Mary Smith” would have a Salutation generated of “Mr and Mrs Smith” and a Contact generated of “Mr and Mrs J Smith” or “Mr and Mrs John Smith”.

EngineSettings.Generate.Name.ParseAsNormalizedName

When enabled, addressee names are assumed to be in a delimited normalized format similar to the Record.MatchingFields.NormalizedName value that’s output by Engine.Generate( ). Currently supported delimiters are spaces, commas, semicolons, and pipes (‘|’).

This property is disabled by default.

EngineSettings.Generate.Name.ParseNameElements

When enabled, this will cause input name elements (including prefix, firstnames, and lastname) to be parsed. If the matchIT API deems any values to have been entered into an incorrect field (for example, suffixes and qualifications in the lastname field), it will reassign these values into the correct fields.

This property is disabled by default, so that any such incorrect values are not reassigned.

EngineSettings.Generate.Name.ProcessBlankLastName

With this property enabled, a blank lastname will cause extra processing to be performed on other input data to help detect typographical errors. For example, if a firstname was entered but not a lastname, then it’ll be assumed that the firstname is in fact the lastname and match keys will be generated rather than being left blank.

This property is disabled by default.

EngineSettings.Generate.Name.ReplaceAndWithAmpersand

On return from Engine.Generate( ) the matchIT API will, by default, convert ‘and’ to an ampersand when outputting InputFields.Name.Addressee. Disabling this property will prevent this behaviour.

EngineSettings.Generate.Name.UseEquivalentNames

If you set the Use Equivalent Name property to True, in the generated the matchIT API replaces the first name with its equivalent from the NAMES.DAT file, if there is an entry for the input first name. This enables, for example, “Tony Smith” and “Anthony Smith” to be picked up as a match. The initial of the original first name is stored in the *Record.DataFlags* property to enable, for example, “Tony Smith” and “T Smith” to still be matched.

EngineSettings.Generate.NormalizationDelimiter

This setting (default is a comma, ‘,’) contains the delimiter that’s used when generating the Record.MatchingFields.NormalizedName and Record.MatchingFields.NormalizedOrganization values. This could be useful when

outputting values to a comma-delimited file, for example, so that an alternative character (such as a pipe ‘|’) could instead be used.

EngineSettings.Generate.Organization.Extract.Jobtitle

This will copy or extract job titles contained within the ADDRESSEE field into a field labeled JOB_TITLE.

Job Titles are recognized by having a word or string defined as a Job Title in the NAMES.DAT file e.g. Director.

EngineSettings.Generate.Organization.ExtractName

This will copy or extract any business names contained within the ADDRESSEE field into a field labeled COMPANY.

Business names are recognized by having a word or string defined as a Business word in the NAMES.DAT file e.g. Ltd. Care should be taken when using this property, as words like "Bank" can be taken to indicate a Business when this isn't the case (e.g. it may be a last name or part of an address line).

If you want Extract Company Name processing to be applied also to the first one or two lines of the address, you must Set the property *Settings.Generate.Address.NumOfLinesToScan* to either 1 or 2.

EngineSettings.Generate.Organization.IgnoreParentheses

With this property enabled, any words that are enclosed with parentheses within an organization name will be excluded from the phonetic organization keys. This can be useful for records such as Remnel Ltd and Remnel (UK) Ltd, to ensure records with these company names are compared if the phonetic organization keys are being used as part of composite match keys.

EngineSettings.Generate.Organization.IgnoreTrailingPostTown

This property, when enabled, will exclude from the phonetic organization keys any trailing post town (defined in towns.dat, see Towns Table) or UK county that appears at the end of a company name. For example, the phonetic organization keys for Handso Ltd and Handso Essex Ltd will be the same to help ensure such records will be compared.

EngineSettings.Generate.Organization.JoinInitials

Set this property to True if you want a group of initials separated by spaces or dots in a company name to be concatenated. For example, if this property is True, then “I B M” and “I.B.M.” will be replaced by “IBM”. Note that, if the *Settings.Generate.ProperCase* property is set to False, then this property will have no effect.

EngineSettings.Generate.Organization.NormalizationTruncation

Disabled by default (i.e. set to 0) If this setting is enabled, and the organization consists of more than four words, then the third element of Record.MatchingFields.NormalizedOrganization will be truncated to the first N characters of each word after the first two (where N is the value of this setting).

EngineSettings.Generate.Organization.UseEquivalentName

If this property is set to True, then the equivalent (according to the NAMES.DAT file) of words indicating a business name, such as “Motors” or “Services” are included in the *Record.NormalizedOrganization* property and the corresponding phonetic keys. This enables, for example, “Wood Green Cars” to match “Wood Green Motors” well (because “Cars” has an equivalent of “Motors”), but ensures that neither of them match “Wood Green Carpets” well.

If you set this property to True, you should change any words in the NAMES.DAT file that you do want ignored, such as “Ltd” and “Inc” to Noise type so that they are not included in the *Record.NormalizedOrganization* property. As a rule of thumb, if you are doing business matching on a file that is very geographically concentrated, that is, contains records mostly from the same immediate area, then set the *Settings.Generate.Organization.UseEquivalentName* property to True, otherwise set it to False.

EngineSettings.Generate.ProperCase

If this property is set to True, the Generate step will convert the address lines in your records (labeled ADDRESS1, ADDRESS2... ADDRESSn) to their proper case. This proper casing will handle punctuation, apostrophes and abbreviations. It will also convert ADDRESSEE, JOB_TITLE, DEPARTMENT, and COMPANY to the correct case. Exceptions to the default casing rules are held in the NAMES.DAT file.

The API's default rules for casing data are as follows: letters following an apostrophe are capitalized (e.g. “Mr O’Reilly”), as are letters following “Mc” or (subject to one of the Advanced Input Options) “Mac” at the start of a name (see "Mac Name Treatment"). Double-barreled names have a capital letter after the hyphen. If the name or other word has a proper casing entry in the NAMES.DAT file, it is cased as shown there e.g. BSc, helpIT, IBM, plc. If not in the NAMES.DAT file, words are all capitals if they contain no vowels, otherwise they are changed to initial capital followed by lower case letters.

EngineSettings.Generate.Quality.Enabled

By default, quality scoring is disabled and all quality scores are 0.

EngineSettings.Generate.Quality.Address.AllowBlankPostcode

If disabled (enabled by default) then addresses without a postal code are restricted to a maximum quality score of 1.

EngineSettings.Generate.Quality.Address.Country

With this enabled, address quality scores will receive an extra one point if the address contains a recognised country. This is disabled by default.

EngineSettings.Generate.Quality.Address.MaxRepetition

This setting (the default is 0.7) is used when calculating the repetition level of characters within a string (in this case, all the concatenated address lines and elements). For example, the string “Heheheb” contains seven characters, six of which are involved in repeated sequences (he: hehehe); the repetition level is thus calculated as 6/7=0.857,

which exceeds the max repetition level and will therefore be flagged as *nonsense* and achieve a quality score of 0.

EngineSettings.Generate.Quality.Address.Premise

With this enabled, address quality scores will receive an extra one point if the address contains a premise number. This is disabled by default.

EngineSettings.Generate.Quality.Address.Region

With this enabled (the default), address quality scores will receive an extra one point if the address contains a recognised region (e.g. county or state).

EngineSettings.Generate.Quality.Email.MaxRepetition

(See Settings.Generate.Quality.Address.MaxRepetition, above.)

EngineSettings.Generate.Quality.Email.WebmailFiltering

If enabled (default) then email addresses that use webmail provider (such as Hotmail, Yahoo, & mail.com) domains are restricted to a maximum quality score of 7.

EngineSettings.Generate.Quality.Name.MaxRepetition

(See Settings.Generate.Quality.Address.MaxRepetition, above.)

EngineSettings.Generate.ReportUnrecognisedWords

This can specify a *callback* function that is used to notify the calling application of any unrecognised words (i.e. not found within names.dat) that are encountered when parsing records. The callback function is implemented as an object of a user-defined class that derives from matchIT.IReportUnrecognisedWords.

EngineSettings.Generate.SpecialCaseMac

Where a last name begins with Mac, when formatting salutations, the matchIT API follows this with a small letter or a capital letter, depending on this property. A value of True will mean that MACLEAN will be formatted as MacLean. You can add exceptions to the rule (e.g. Maccabee, Macclesfield, MacKay, Mackie) to the NAMES.DAT file. If you invariably want to use a lower case letter following Mac, set this property to False.

NB: Names beginning Mach are always formatted with a lower case H, e.g. Machin, Machinery. Names beginning Mc are formatted with a capital letter following, if they are greater than 3 characters long.

EngineSettings.Generate.VariableKeysMaxLength

This specifies the maximum length of various variable-length phonetic keys created when a record is generated. Such keys are PhoneticLastName, PhoneticFirstName, PhoneticMiddleName, PhoneticOrganizationName1, PhoneticOrganizationName2,

PhoneticOrganizationName3, PhoneticStreet, and PhoneticTown. The default is eight characters.

EngineSettings.Compare…

The following settings are used when the matchIT API compares two records:

EngineSettings.Compare.Address.DefaultThoroughfareLine

(See Settings.Generate.Address.DefaultThoroughfareLine, above.)

EngineSettings.Compare.Address.LooseFuzzyPremiseMatch

When enabled, this will cause two premises to match if one premise starts with the other but contains extra trailing characters (for example, 88 and 88/2 will match, but 88 and 887 will *not* match).

EngineSettings.Compare.Address.MatchBoxNumberAndPostcode

If this setting is enabled, then two compared addresses score Sure if they contain matching postal box numbers and postcodes (i.e. the remainder of the addresses are ignored).

EngineSettings.Compare.Address.MatchDeliveryPoints

When enabled, this will prevent two addresses from matching when both contain two postal codes but different delivery point codes (for example, DPS codes in the UK, DPV codes in the US) and the addresses score below the minimum threshold.

If either record is missing a delivery point, or either is a default, then the addresses will match regardless.

EngineSettings.Compare.Address.MatchDeliveryPointsThreshold

(See Settings.Compare.Address.MatchDeliveryPoints, above.)

EngineSettings.Compare.Address.DefaultDeliveryPoints

(See Settings.Compare.Address.MatchDeliveryPoints, above.)

EngineSettings.Compare.Address.IgnorePremiseSuffix

When enabled, this will cause two premises to be matched regardless of whether one or both has an apartment- or flat-type suffix (for example, 12 and 12a). Normally, such premises will cause the address score to be reduced because the addresses are considered different, which could prevent the two records being flagged as a match.

EngineSettings.Compare.Address.UsePremiseRange

When this setting is enabled, this will allow addresses to contain premise *ranges*. For example, if one record contains an address line of "11-15 Main Street" and the other "13 Main Street", then the premises are considered a match with this setting enabled; otherwise, the premises will not be matched and, depending on constraints and weights, the addresses might not score a high enough score to be considered matching records.

EngineSettings.Compare.Name.FuzzyMatchNonNormalizedNames

When enabled (the default), this will cause additional matching checks to be performed on names using the non-normalized name matching fields. This can be useful when Settings.Generate.Name.UseEquivalentNames is enabled, which will allow Elizabeth and Lisa to match, but will not allow for some misspellings and typos such as Lsia to match.

EngineSettings.Compare.Name.OrganizationMatchingOnBlankNames

When two records contain no addressee names, this setting will allow the names to achieve a score depending on what's available in the job title and company name fields. For example, if the two records contain job titles of Managing Director and company name of helpIT systems, then a positive name score will be given even though the records don't contain an addressee.

EngineSettings.Compare.Name.PreventMrsMatchingMiss

If this setting is enabled, then two compared names will not match if one has a title of Mrs and the other a title of Miss. For example, "Mrs J Smith" will *not* match "Miss J Smith" with the setting enabled (the default).

EngineSettings.Compare.Phonetic.Algorithm

There are two stages to the matching process that the matchIT API uses; the key stage and the scoring stage. The first stage creates standardized and phonetic keys based on the input data, which allows potential matches to be identified. The second stage scores each pair of potential matches, using phonetic and fuzzy matching. This property governs the phonetic algorithm that the API uses when scoring.

There are four choices available:

soundIT

The API provides a unique phonetic algorithm for name matching, called soundIT. soundIT takes account of vowel sounds and syllables in the name, and, more importantly, determines the stressed syllable in the word. This means that "Batten" and "Batton" sound the same according to soundIT, as the different letters fall in the unstressed syllable, whilst "Batton" and "Button" sound different, as it is the stressed syllable which differs. Another advantage of soundIT is that it can recognize groups of vowels and consonants that form vowel sounds – thus it can equate "Shaw" and "Shore", "Wight" and "White", "Naughton" and "Norton", and "Leighton" and "Layton" (which are all reasonably common English last names).

This algorithm was developed with extensive testing on a large table of the most common last names in the UK. Therefore, it is specifically designed to be used with English names. If a file with mostly non-English names is processed through the

matchIT API, then you may want to try the 'Loose' soundIT or Soundex algorithms instead. For US data we recommend that you use soundIT, because it is proven to work well also with Spanish, German and other names that occur commonly in the US. soundIT has been designed with foreign language versions in mind (i.e. for data collected in countries where foreign languages are spoken). These could quite easily be developed, according to demand. Please contact your supplier if you are interested in this.

Note that the keys that the matchIT API generates are 'Loose' soundIT keys, where all vowel sounds are equated, together with some consonants, such as 'm' and 'n', 'd' and 't', 's' and 'f'. This is so that potential matches are not missed from candidate match groups based on the phonetic keys; The API uses the 'full' soundIT algorithm at the scoring stage, for matching accuracy.

Loose soundIT

This option is effectively the same as the soundIT option, except that the API uses the 'Loose' soundIT algorithm as described above at the scoring stage. This is for use mainly with non-English names, on which soundIT works less effectively, and can miss True matches. This option should not be used on files with mainly English names, as it can potentially lead to more false matches.

Dynamic soundIT

This is a hybrid of the soundIT and Loose soundIT phonetic algorithms. Firstly, the loose algorithm is used to generate the phonetic form of a word. By default, if it contains only one vowel sound, then the standard soundIT algorithm is used instead. This can improve accuracy when matching mono-syllabic words and can help to reduce the number of false matches.

See EngineSettings.Compare.Phonetic.LooseThresholdForDynamicSoundIT below.

Soundex

Soundex is a widely-used algorithm (patented just after the First World War!), which constructs a crude non-phonetic key by keeping the initial letter of the name, then removing all vowels, plus the letters H, W and Y, and translating the remaining letters to numbers. It gives the same number to letters that can be confused e.g. 'm' and 'n' both become 5. It also drops repeated consonants and consecutive letters that give the same number e.g. S and C. It only takes the first four characters of the result, or pads it out with zeroes if it is less than four long. Thus all the common spellings and misspellings of the name "Tootill" equate to the same Soundex key: Tootill, Toothill, Tootil, Tootal, Tootle, Tuthill, Totill are all translated to "T340".

The algorithm that the matchIT API uses is an enhanced version of Soundex, and is for use mainly with non-English names. This option should not be used on files with mainly English names, as it can lead to False matches e.g. Brady, Beard and Broad get the same Soundex key.

Non-phonetic

This option constructs a non-phonetic version of the supplied name fields as match keys and allows only non-phonetic name matching.

EngineSettings.Compare.Phonetic.AlgorithmForFirstNames

By default, this property is set to PhoneticAlgorithm.None which simply means that EngineSettings.Compare.Phonetic.Algorithm will be used.

Otherwise, all first names will be phoneticised using this setting.

This can be useful to impose a 'tighter' level of matching for first names than for last names, where firstnames are often abbreviated to short forms.

EngineSettings.Compare.Phonetic.LooseThresholdForDynamicSoundIT

When Dynamic soundIT is in use, this property controls the threshold at which soundIT is switched to Loose soundIT. The default is 2, which means that words containing less than two syllables are phoneticised using soundIT instead of Loose soundIT.

EngineSettings Methods

ToXML(String)

This method returns the engine's current settings as an XML-formatted string. Note that the XML schema follows the layout of the engine's COM hierarchy (for example, the settings/compare/phonetic/algorithm node in the XML can be accessed via the Engine.Settings.Compare.Phonetic.Algorithm property in code).

FromXML(String)

This method applies the settings in the passed XML-formatted string to the engine. Note that any number of settings can be specified in the XML, not all need be specified. If the method returns an error (i.e. the XML is invalid), an error message string can be retrieved using the Engine.LastError property.

Copy(EngineSettings)

This method copies the settings from the specified EngineSettings object.

For example, to make a copy of the engine's current settings, use:

```
EngineSettings temp = new EngineSettings();
temp.Copy( engine.Settings );
```

MatchingSettings class

Note that it is not possible to create an instance of this class – it is only accessible via EngineSettings.MatchingRules.*Level (where * is one of Individual, Family, Household, Business, or Custom). See Matching Levels for more info.

This class exists only within the Engine class, as one of five levels (i.e. individual, family, household, business, and custom). Each MatchingSettings instance contains several matching constraints, a number of weight values used when processing raw comparison results via the Engine.ApplyMatchingScores( ) method, a name matching matrix, and an organization matching matrix.

MatchingSettings Properties and Methods

Properties	*Type*
Constraints.MustMatchGender	Boolean
Constraints.MustMatchLocation	Boolean
Constraints.MustMatchPremise	Boolean
Constraints.NoOneEmptyPremise	Boolean
Constraints.AllowFuzzyPremiseMatch	Boolean
Constraints.MustMatchSuffix	Boolean
Constraints.MustMatchDirectional	Boolean
Constraints.MustMatchNumericStreetName	Boolean
Constraints.MustMatchJointNames	Boolean
Constraints.MustMatchBuilding	Boolean
Constraints.NoOneEmptyBuilding	Boolean
Weights.Name.SetMatrixWeights( )	double, double, double
Weights.Name.GetMatrixWeights( )	double, double, double
Weights.Name.BothEmptyScore	double
Weights.Name.OneEmptyScore	double
Weights.Organization…	as Weights.Name
Weights.Address.SetWeights( )	double, double, double
Weights.Address.GetWeights( )	double, double, double
Weights.Address.BothEmptyScore	double
Weights.Address.OneEmptyScore	double

Weights.Postcode…	as Weights.Address
Weights.Telephone…	as Weights.Address
Weights.Email…	as Weights.Address
Weights.DateOfBirth…	as Weights.Address
Thresholds.Name	double
Thresholds.Organization	double
Thresholds.Address	double
Thresholds.Postcode	double
Thresholds.Telephone	double
Thresholds.Email	double
Thresholds.DateOfBirth	double
NameMatchingMatrix	(see NameMatchingMatrix class)
OrganizationMatchingMatrix	(see OrganizationMatchingMatrix class)

MatchingSettings Property Usage

EngineSettings.MatchingRules.*Level.Constraints…

There are different copies of each of the properties below; one for each of the different matching levels (Business, Family, Household and Individual). For Example, *Settings.MatchingRules.IndividualLevel.Constraints.MustMatchGender* would be set differently to *Settings.MatchingRules.FamilyLevel.Constraints.MustMatchGender*.

MustMatchGender

When this property is set to True, potential matches will be disregarded if their genders differ. If however the gender is unknown in one or both of the records, the records will potentially be classed as a match.

MustMatchLocation

When this property is set to True, potential matches will be disregarded if their address locations differ. In detail, this means that the postcodes in the two records (if present) must achieve at least a probable match with the address score at least a Possible match, or the address score must be at least a Likely match irrespective of the postcodes, or the postcodes must achieve a Sure match irrespective of the address. This is to prevent false matches where there is some match on address, but where the addresses are clearly not the same, for example "10 High Street, Bookham", and "10 High Street, Alford". Switch this constraint off if you want to match people or companies in different locations; you may want to match on items of data that are independent of location, such as date of birth or bank account. See also the topic "What matching weights should I use for my data?" in the matchIT API Frequently Asked Questions.

MustMatchPremise

When this property is set to True, potential matches will be disregarded if their premise numbers differ. If however the premise number is unknown (e.g. one record or both records may contain a premise name), the records will potentially be classed as a match.

NoOneEmptyPremise

When this property is set to True, potential matches will be disregarded if one of the addresses is missing a premise number.

AllowFuzzyPremiseMatch

When *both* this and MustMatchPremise are set to True, then potential matches will be disregarded if the premises are not exact matches (for example, 71 and 71) or if they're not *fuzzy* matches (for example 71 and 71A, 45 and 54, or 71 and 7). Note that this property has no effect if MustMatchPremise is set to False because, in that case, fuzzy premises are always allowed.

MustMatchSuffix

When this property is set to True, potential matches will be disregarded if their suffixes differ. If however the suffix is unknown in one or both of the records, the records will potentially be classed as a match.

MustMatchDirectional

When this property is set to True, potential matches will be disregarded if both addresses (i.e. typically US) have a pre- or post-directional (e.g. N, North, E, etc.) but they don't match. For example, with this constraint enabled, "N Washington Ave" and "S Washington Ave" will not be matched.

MustMatchNumericStreetName

When this property is set to True, potential matches will be disregarded if both addresses (i.e. typically US) have a numeric street name but they don't match. For example, with this constraint enabled, "5th Ave" and "15th Ave" will not be matched.

MustMatchJointNames

When this property is set to True, potential matches will be disregarded if one record has a joint name but the other doesn't. For example, normal behaviour will match "Mr & Mrs J Smith" with "Mr J Smith"; setting this property to True will prevent such matches.

MustMatchBuilding

When this property is set to True, potential matches will be disregarded if their building names differ. If however one or both addresses do not contain a building name, the records will potentially be classed as a match.

NoOneEmptyBuilding

When this property is set to True, potential matches will be disregarded if one of the addresses is missing a building name.

EngineSettings.MatchingRules.*Level.Weights.#...

Where * is one of Individual, Family, Household, Business, or Custom and # is one of Name, Organization, Address, Postcode, Telephone, Email, or DateOfBirth.

For each of the four following methods/properties, there is a different copy of the method/property for each of the five different matching levels.

SetMatrixWeights( )

Regenerates the current name or organization matching matrix using the supplied sure, likely, and possible weight values.

SetWeights( )

Sets the sure, likely, and possible weights that are used to score compared items (NB: *not* names and organizations, use SetMatrixWeights( ) instead).

Sure specifies the maximum score to be returned when the two compared items match. If they're not identical, but similar, then either likely or possible will be scored depending on the field being compared and the data being compared.

BothEmptyScore

This property specifies the score to be returned when both compared items are empty.

OneEmptyScore

This property specifies the score to be returned when one of the two compared items are empty.

EngineSettings.MatchingRules.*Level.Thresholds.#

Where * is one of Individual, Family, Household, Business, or Custom and # is one of Name, Organization, Address, Postcode, Telephone, Email, or DateOfBirth.

Thresholds are applied in this order, to prevent two records from matching when the cumulative total score fails to reach any threshold (all default to 0).

For example, suppose the postcode threshold is set at 100, the weights for name, address, and postcode are set at 60, 30, and 35 respectively, and that the three components are each scoring Sure. The name threshold (0) will be checked with a cumulative score of 60, the address threshold (0) will be checked with a cumulative score of 90, and the postcode threshold (100) will be checked with a cumulative score of 125. All three thresholds are reached. However, should the names score 25 (Possible), then the cumulative score will be 25, 55, and 90 at the same three points; the postcode threshold (100) will therefore not be reached, thus the two records will score 0.

NameMatchingMatrix class

Note that it is not possible to create an instance of this class – it is only accessible via EngineSettings.MatchingRules.*Level.NameMatchingMatrix (where * is one of Individual, Family, Household, Business, or Custom). See Matching Levels for more info.

The name matching matrix – of which there is one for each of the five matching levels (individual, family, household, business, custom) – is a three-dimensional matrix used to process the raw matching results that are output from the Engine.Compare( ) method, using the Engine.ApplyMatchingResults( ) method.

The raw matching results indicate the comparison result between fields of the two records – for example, the names could be sounds-equal. Using the name matching matrix as a lookup, the raw results are then transformed using the Engine.Score( ) method into final matching scores.

These scores are the product of the relevant cell value from the matrix and the relevant level's component weight (e.g. cell * weight, where cell is Level.LastNamesEqual.FirstNamesEqual.MiddleNamesBothEmpty, and weight is Level.Weights.Name.Weight, assuming Level is Engine.Settings.MatchingRules.IndividualLevel).

NameMatchingMatrix Properties

Properties	*Type*
LastNamesEqual.FirstNamesEqual.MiddleNamesEqual	double
LastNamesEqual.FirstNamesEqual.MiddleNamesBothEmpty	double
LastNamesEqual.FirstNamesEqual.MiddleNamesOneEmpty	double
LastNamesEqual.FirstNamesEqual.MiddleNamesApprox	double
LastNamesEqual.FirstNamesEqual.MiddleNamesContain	double
LastNamesEqual.FirstNamesEqual.MiddleNamesUnequal	double
LastNamesEqual.FirstNamesSoundEqual…	double
LastNamesEqual.FirstNamesBothEmpty…	double
LastNamesEqual.FirstNamesOneEmpty…	double
LastNamesEqual.FirstNamesApprox…	double
LastNamesEqual.FirstNamesSoundApprox…	double
LastNamesEqual.FirstNamesContain…	double
LastNamesEqual.FirstNamesUnequal…	double
LastNamesSoundEqual…	double

LastNamesApprox…	double
LastNamesSoundApprox…	double
LastNamesContains…	double
LastNamesUnequal…	double

NameMatchingMatrix Methods

Set(NameMatchingMatrix)

This method is simply used for copying the specified name matching matrix, overwriting that which is currently stored in an object of this class.

OrganizationMatchingMatrix class

Note that it is not possible to create an instance of this class – it is only accessible via EngineSettings.MatchingRules.*Level.OrganizationMatchingMatrix (where * is one of Individual, Family, Household, Business, or Custom). See Matching Levels for more info.

An organization matching matrix is effectively identical to a name matching matrix (see NameMatchingMatrix class), but used specifically when comparing organizations instead of names.

OrganizationMatchingMatrix Properties

Properties	*Type*
Name1Equal.Name2Equal.Name3Equal	double
Name1Equal.Name2Equal.Name3BothEmpty	double
Name1Equal.Name2Equal.Name3OneEmpty	double
Name1Equal.Name2Equal.Name3Approx	double
Name1Equal.Name2Equal.Name3Contain	double
Name1Equal.Name2Equal.Name3Unequal	double
Name1Equal.Name2SoundEqual…	double
Name1Equal.Name2BothEmpty…	double
Name1Equal.Name2OneEmpty…	double
Name1Equal.Name2Approx…	double
Name1Equal.Name2SoundApprox…	double
Name1Equal.Name2Contain…	double
Name1Equal.Name2Unequal…	double
Name1SoundEqual…	double
Name1Approx…	double
Name1SoundApprox…	double
Name1Contains…	double
Name1Unequal…	double

OrganizationMatchingMatrix Methods

Set(OrganizationMatchingMatrix)

This method is simply used for copying the specified organization matching matrix, overwriting that which is currently stored in an object of this class.

NamesAndWords class

This class contains the data lookup tables. These must be loaded – by calling the Initialize( ) method – prior to initializing the matchIT engine.

NamesAndWords Methods

Initialize(String, Country)

This method requires both a string specifying the folder containing the data lookup tables, and an enumeration value specifying the nationality of the tables (see Appendix A Enumeration A for a list of countries).

InputFields class

This class specifies the data that is input to the Engine.Generate( ) method (the output of which is a populated Record object).

InputFields Properties

Properties	*Type*	*Description*	*Example*	*Requirement*
Addressee.Contact	string	Standardized form of personal name in one field, as used in addressing mail. The matchIT API generates this field from input *Prefix*, *FirstNames/Initials* and *LastName* or *Addressee It is a Read Only property.*	Mr Geoff Smith Mr G. Smith	Not Required
Addressee.FullName	string	Freeform personal name in one field, as used in addressing mail. The matchIT API can generate this field from input *Prefix, FirstNames/Initials* and *LastName*.	MR G.C. SMITH Geoff Smith Esq. Mr G. Smith & Miss S. Brown	Required for matching at Contact, Individual or Family level, if separate Prefix, FirstNames/Initials and LastName fields are not present.
Addressee.Salutation	string	Standardized form of personal name in one field, as a salutation for a letter. The matchIT API generates this field from input Prefix, FirstNames/Initials and LastName or Addressee It is a Read Only property.	Dear Mr Smith	Not Required

Properties	*Type*	*Description*	*Example*	*Requirement*
Addressee.NameElements.LastName	string	The last name, typically when the name is split into component parts of *Prefix, FirstNames/Initials* and *LastName*.	Smith van der Valk	Required for matching at Contact, Individual or Family level, if a *FullName* field is not present.
Addressee.NameElements.FirstNames	string	First names (also called *F*irstNames) or initials in one field, typically when the name is split into component parts of *Prefix, FirstNames/Initials* and *LastName.* The matchIT API will generate *FirstNames* from information contained in a *FullName* field if present.	J JR J R John Robert, John R	Required for matching at Contact, Individual or Family level, if a *FullName* field is not present.
Addressee.NameElements.Initials	string	Initials of either just the middle name(s), or of all the first names.	Where the name of the person is “John R Smith”: JR or J R if *FirstNames* is not present or empty. R if *FirstNames* contains John	Required for matching at Contact, Individual or Family level, if a *FullName* field is not present.
Addressee.NameElements.Prefix	string	Personal title	Mr Mrs Dr Professor	Required for matching at Contact, Individual or Family level, if a *FullName* field is not present.
Addressee.NameElements.Suffix	string	A title in a separate field following the last name. The matchIT API will	Esq Jr	Not Required

Properties	*Type*	*Description*	*Example*	*Requirement*
		generate Suffix from information contained in an *Addressee* or *LastName* field if present.		
Addressee.NameElements.Qualification	string	A Qualification. The matchIT API will generate the Qualification from information contained in an *Addressee* or *LastName* field if present.	B.Sc. ARCS BCom	Not Required
Addressee.SecondNameElements.LastName	string	The last name of a second individual. The matchIT API can generate this from information contained in an *Addressee* field, if the *Addressee* field is identified as containing more than one name. E.g. “Mr John M Smith and Ms Sarah D Jones”, or “Mr and Mrs Jones”.	Jones	Not Required
Addressee.SecondNameElements.FirstNames	string	The forename(s) of a second individual. The matchIT API can generate this from information contained in an *Addressee* field, if the *Addressee* field is identified as containing more than one name. E.g. “Mr John M Smith and Ms	Sarah D Sarah S D S	Not Required

Properties	*Type*	*Description*	*Example*	*Requirement*
		Sarah D Jones", or "Mr and Mrs Jones".		
Addressee.SecondNameElements.Initials	string	The initials of a second individual. The matchIT API can generate this from information contained in an *Addressee* field, if the *Addressee* field is identified as containing more than one name. E.g. "Mr John M Smith and Ms Sarah D Jones", or "Mr and Mrs Jones".	Where the name of the person is "Mrs Sarah D Jones": S D if *FirstNames* is not present or empty. D if *FirstNames* contains Sarah	Not Required
Addressee.SecondNameElements.Prefix	string	The prefix of a second individual. The matchIT API can generate this from information contained in an *Addressee* field, if the *Addressee* field is identified as containing more than one name. E.g. "Mr John M Smith and Ms Sarah D Jones", or "Mr and Mrs Jones".	Ms Mrs	Not Required
Address.Lines.Line1 Address.Lines.Line2 Address.Lines.Line3 Address.Lines.Line4	string	The only line of address, or one of multiple address lines. The address line(s) may also contain postcodes or zip codes, as the matchIT API can extract	560 So Winchester Blvd, Fl 5 San Jose, CA 95128 560 So Winchester Blvd Fl 5 San Jose CA 95128 9 North Street	You must have at least one address line for address matching.

Properties	*Type*	*Description*	*Example*	*Requirement*
Address.Lines.Line5 Address.Lines.Line6 Address.Lines.Line7 Address.Lines.Line8 Address.Lines.Line9		these into a designated *Postcode* field.	Leatherhead Surrey KT22 8DY Leatherhead KT22 8DY	
Address.Elements.Postcode	string	Should contain either full or partial UK postcodes or zip codes. The matchIT API can extract valid postcodes or zip codes from address lines into the *Postcode* field.	GU14 7BQ KT228DY KT22 95128 95128-2519	Required in order to include postcodes and zip codes in the matching process.
Address.Elements.Premise	string	The premise number part of the address – this is the building number, not the apartment or unit number if there is one The matchIT API can move or copy some premise numbers from an address line to the *Premise* field.	260 12-14	Not required, however this can be used within the match keys in order to enhance the accuracy and efficiency when performing matching processes within large files or files containing very localized data.
Address.Elements.FlatNo	string	The apartment or unit number, as opposed to *Premise*, which is the building number. The matchIT API can	A 3 3a	Not Required

Properties	*Type*	*Description*	*Example*	*Requirement*
		extract or copy apartment or flat numbers from the address lines into the *FlatNo* field.		
Address.Elements.Thoroughfare	string	The thoroughfare element of the address.. The matchIT API can extract or copy what it recognizes as the thoroughfare, from the address lines into the *Thoroughfare* field.	North Street The Crescent	Not Required
Address.Elements.Town	string	The town element of the address. The matchIT API can extract or copy any valid UK post towns, from the address lines into the *Town* field	Kingston Upon Thames FARNHAM	Not Required
Address.Elements.Region	string	The US or Australian state or Canadian province, or UK county element of the address. The matchIT API can extract or copy states or provinces or UK counties, defined as such in the NAMES.DAT file, from the address lines into the *Region* field.	California NY Surrey	Not Required
Address.Elements.Country	string	The country element of the	England	Not Required

Properties	*Type*	*Description*	*Example*	*Requirement*
		address. The matchIT API can extract or copy any valid country names from the address lines into the *Country* field.	UK Azerbaijan	
Email.Address	string	An email address.	Support@helpIT.com Sales@helpIT.com JohnD@helpIT.com	Not Required
Email.Username	string	Contains the username extracted from the email address if its format is correct.	Support Sales JohnD	Not Required
Email.Domain	string	Contains the domain extracted from the emailaddress if its format is correct.	helpIT.com	Not Required
Organization.Name	string	A company or business name. The matchIT API can move some company names from the *Addressee* or an address line to the *Company* field.	helpIT systems ltd The ABC Company	Not Required
Organization.Department	string	A department name.	Sales Dept. Marketing	Not Required
Organization.Jobtitle	string	A job title. The matchIT API can	Managing Director MD	Not Required

Properties	*Type*	*Description*	*Example*	*Requirement*
		move some job titles or department names from the *Addressee* or an address line to the *JobTitle* field.		
Telephone.Number	string	A telephone number. The matchIT API can split telephone numbers into two parts: *TelAreaCode* for the area code and *TelLocalNumber* for the local part of the number. The user can sometimes more effectively use *TelLocalNumber* for matching.	01372 360070 (01372) 360 070 01372-360070	Not Required
Telephone.Fax	string	A fax number. The matchIT API can split fax numbers into two parts: *FaxAreaCode* for the area code and *FaxLocalNumber* for the local part of the number.	As for *Telephone.Number*	Not Required

Record class

The record class has two uses:

- Firstly, on returning from the Engine.Generate( ) method a Record object is populated with generated data; this is subsequently used for finding matching records in a database.
- And secondly, when calling the Engine.Compare( ) method two Record objects are first initialized with the data to be compared, then these two records are passed in to the method (and a populated CompareResult object is output, see CompareResult class).

In the Properties table below, the Match Key column indicates fields that are recommended for using as match keys (although you can of course use any fields as match keys). These fields will give you the most effective results.

The Matching Field column indicates fields that must be supplied as part of the matching process. These are fields that contain additional, useful information to help out with the matching process.

If you are storing generated fields in a database (to save time by not repeating the Engine.Generate( ) method for each record every time it is involved in a comparison), it is a good idea to store those match keys that you want to use from the list below, and it is essential to store the matching fields.

Record Properties

Properties	*Type*	*Match Key?*	*Match-ing Field?*	*Description*	*Example*	*Main Purpose*
InputFields	(see InputFields class)	Yes				
KeyFields.NameKey	string	Yes		Phonetic name key of last name plus first initial	For “Bill Dayton”, “William Deighton Jr” and “Bill Dayten” by default the matchIT API generates “dytymW”	Finding records to compare
KeyFields.OrganizationKey	string	Yes		Phonetic business name key of first significant word in the company name.	For “The A.B.C. Co.”, “ABC Ltd” and “A B C Company Ltd” by default the matchIT API generates “ABC”. “The” is ignored in the first example, and “ABC” is treated as the first word whether or not it has spaces, periods or nothing separating the letters ABC. For “Sanford Services Inc”, the matchIT API generates “symfy” if *Settings.Generate.Organization.UseEquivalentName* is False or “symfysyvys” if it is True	Finding records to compare
KeyFields.AddressKey	string	Yes		Phonetic town/city (first 5 characters) and phonetic street name (remaining 5 characters).	For “North Street, Leatherhead” and “North St, LEATHERHEAD”, by default the matchIT API generates “lyTynyT”	Finding records to compare
KeyFields.PhoneticStreet	string	Yes		Phoneticization of the	For “North Street, Leatherhead” and “North St, LEATHERHEAD”, by default the matchIT API	Finding records to

Properties	*Type*	*Match Key?*	*Match-ing Field?*	*Description*	*Example*	*Main Purpose*
				thoroughfare found in the address lines. Combines with the PhoneticTown to form the AddressKey (last five characters)	generates “nyT”	compare
KeyFields.PhoneticTown	string	Yes		Phoneticization of the town/city found in the address lines. Combines with the PhoneticStreet to form the AddressKey (first five characters)	For “North Street, Leatherhead” and “North St, LEATHERHEAD”, by default the matchIT API generates “lyTy”	Finding records to compare
KeyFields.PhoneticLastName	string	Yes		Phonetic last name	For “Bill Dayton”, “William Deighton Jr” and “Bill Dayten” the matchIT API generates “dytym”	Finding records to compare
KeyFields.PhoneticFirstName	string			Phonetic forename	For “Bill Dayton”, “William Deighton Jr” and “Will Dayten” by default the matchIT API generates “wyl”	Finding records to compare
KeyFields.PhoneticMiddleName	string			Phonetic middle name if present	For “John Mark Smith”, the matchIT API generates “myk”.	Finding records to compare

Properties	*Type*	*Match Key?*	*Match-ing Field?*	*Description*	*Example*	*Main Purpose*
					For “Mr J M Smith”, the matchIT API generates “m”.	
KeyFields.PhoneticOrganizationName1	string			Phonetic version of first word in business name	For “The A.B.C. Co.”, “ABC Ltd” and “A B C Company Ltd” by default the matchIT API generates “ABC”, as described above.	Finding records to compare
KeyFields.PhoneticOrganizationName2	string			Phonetic version of second word in business name if present	For “The A.B.C. Co.”, “ABC Ltd” and “A B C Company Ltd” the matchIT API does not generate a value if *Settings.Generate.Organization.UseEquivalentName* is False as the second significant word in each of these examples is a business word and is therefore ignored. If you set *Settings.Generate.Organization.UseEquivalentName* to True, it is recommended that you change the type of words like Co, Ltd and Inc in the NAMES.DAT file from type “B” for Business to type “N” for Noise. “The” is ignored by default in the first example as a Noise word, and “ABC” is treated as the first word.	Finding records to compare
KeyFields.PhoneticOrganizationName3	string			Phonetic version of third word in business name if present	Similar to the *Name2* property above. If there are more than three significant elements to the company name, those after the third significant element are ignored.	Finding records to compare
MatchingFields.FirstNameFound	string			Indicates the presence of a recognized first name (from the NAMES.DAT	For “John Smith,” or “Mr Chris Smith” or “Miss Chris Smith” by default the matchIT API generates “Y”. It leaves this property blank for “Mr Smith”. For “Chris Smith” the matchIT API generates “E” meaning “Either Gender”.	Comparing records

Properties	*Type*	*Match Key?*	*Match-ing Field?*	*Description*	*Example*	*Main Purpose*
				file) in the Addressee or FirstNames fields.	For “Mr Christine Smith” the matchIT API generates “X” meaning that the prefix and first name are inconsistent.	
MatchingFields.NormalizedName	string		Yes	Normalized version of full name if present	For “Bill Dayton”, “William Dayton Jr” and “Will Dayton” by default the matchIT API generates “DAYTON,WILL,”, whereas for “William Deighton” it generates “DEIGHTON,WILL,”.	Comparing records
MatchingFields.NormalizedOrganization	string		Yes	Normalized version of full business name if present	For “Sanford Services Inc”, the matchIT API generates “SANFORD,,” if *Settings.Generate.Organization.UseEquivalentName* is False or “SANFORD,SERVICES” if it is True	Comparing records
MatchingFields.Gender	Gender, see Appendix A Enumeration D: Gender		Yes	Gender if individual name supplied	For “John Smith,” or “Mr Chris Smith” by default the matchIT API generates “Male”. For “Christine Smith” or “Miss Chris Smith” by default the matchIT API generates “Female”. For “Dr Chris Smith”, “Chris Smith” or “Mr Christine Smith” the matchIT API generates “UnknownGender” by default.	Comparing records
MatchingFields.PostIn	string	Yes		Inward part of UK postcode, if full valid UK postcode supplied. Plus 4 part of US Zip, if supplied.	For a UK *Postcode* “KT22 8DY”, the matchIT API populates this with “8DY”. For a US zip in the Postcode field of “10536-1423”, the matchIT API populates this with “1423”.	Finding records to compare
MatchingFields.PostOut	string	Yes		Outward part of UK	For a UK Postcode “KT22 8DY”, the matchIT API populates this with “KT22”.	Finding records to

Properties	*Type*	*Match Key?*	*Match-ing Field?*	*Description*	*Example*	*Main Purpose*
				postcode if full or partial valid UK postcode supplied. First 5 digits of US Zip, if supplied.	For a US zip in the *Postcode* field of "10536-1423" or "10536", the matchIT API populates this with "10536".	compare
MatchingFields.TelAreaCode	string			Area code from telephone number if supplied	For "01372 225 904" or "01372225904" the matchIT API populates this with "01372".	Finding records to compare
MatchingFields.TelLocalNumber	string			Local part of telephone number if supplied. Obtained by removing the area code	For "01372 225 904" or "01372225904" the matchIT API populates this with "225904".	Finding records to compare
MatchingFields.FaxAreaCode	string			Area code from fax number if supplied	As for MatchingFields.TelAreaCode	Finding records to compare
MatchingFields.FaxLocalNumber	string			Local part of fax number if supplied. Obtained by removing the area code	As for MatchingFields.TelLocalNumber	Finding records to compare
MatchingFields.DataFlags	string		Yes	Each position specifies a	If for example, a record was excluded, this would contain an "X" in position 1. See the next table for a	Comparing records

Properties	*Type*	*Match Key?*	*Match-ing Field?*	*Description*	*Example*	*Main Purpose*
				different property for that record	full breakdown	
QualityFields,Name	integer			The name quality is calculated according to which name elements have been specified, or parsed from the addressee	0 = empty or nonsense 1 = no surname 2 = surname only 3 = initial + surname 4 = prefix + surname 5 = prefix + initial + surname 6 = unrecognized firstname + surname 7 = firstname + surname 8 = firstname + middlename(s) + surname 9 = prefix + firstname + surname 10 = prefix + firsname + middlename(s) + surname	Quality analysis
QualityFields.Address	integer			The address quality score calculated from the address lines and fields.	0 = empty or nonsense 1 = no town and/or street 1 = no postcode if Settings.Generate.Quality.Address.AllowBlankPostcode is disabled 3 = no town and no postcode 4 = no street but postcode 5 = no town but postcode *otherwise score1 for region plus 2 for each street, town, and postcode (maximum of 7).	Quality analysis

Properties	*Type*	*Match Key?*	*Match-ing Field?*	*Description*	*Example*	*Main Purpose*
QualityFields.Email	integer			The email quality score calculated from the email address.	0 = empty, nonsense 1 = invalid format 2 = invalid top-level domain (com, org, uk, fr etc.) 5 = generic username (sales, support, postmaster etc.) 6 = username doesn't match the firstname & lastname from the Input fields 7 = webmail domain (eg. Hotmail.com, mail.com) if Settings.Generate.Quality.Email.WebmailFiltering is enabled 9 = neither of the above apply	Quality analysis

Data Flags Usage

Position	*DataFlags Usage*	*Values*
1	Exclusion Status	**X** if exclusion words are found, **blank** otherwise
2	Company Extracted	**C** if company name has been extracted
3	Company Acronym	This is set to **1**,**2** or **3** if any part of the company name is an acronym
4	Job Title/ Department	**J** if job title extracted, **D** if department extracted
5	Verified Postcode	**V** if postcode was verified OK
6	Extracted Postcode	**E** if postcode was extracted OK
7	Not used by the matchIT API	
8	Generated Prefix status	**P** - generated prefix, **Q** - changed prefix, **S** - used supplied prefix **blank** - no personal name processing
9	Salutation status	**S** - default salutation generated **G** - non-default (success!) salutation generated **blank** - no salutation field
10	Original Initial	Set to the first letter of the input forename field
11	Original Initial #2	First letter of second input forename, if any
12	Second Name Sex	Set to sex of second name if supplied & possible to calculate

13	Foreign status	**F** if record is foreign
14	Not used by the matchIT API	
15	Not used by the matchIT API	
16	Premise status	**X** if premise extracted, **C** if premise copied
17	Zip status	**X** if zip extracted, **C** if zip copied
18	Town status	**X** if town extracted, **C** if town copied
19	County status	**X** if county extracted, **C** if county copied
20	Country status	**X** if country extracted, **C** if country copied.

Record Methods

Clear( )

This method simply resets all the data in the record object back to default (empty) values.

Copy(Record)

This method makes a copy of the supplied record.

CompareResult class

These are the results obtained when two records are compared using Engine.Compare( ).

The results can be further processed by running them through Engine.ApplyMatchingRules( ) to obtain results in the MultiLevelMatchingResults format (see MultiLevelMatchingResults).

Each result from the MatchingMatrixIndex enumeration can be one of Equal, Approx, Unequal, etc. For example, if Name.LastName is Approx, then the last names from the two compared records are approximate, according to the matchIT comparison algorithm.

CompareResult Properties

Properties	*Type*
Name.LastName	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex
Name.FirstName	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex
Name.MiddleName	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex
Name.SuffixMatch	Boolean
Name.GenderMatch	Boolean
Organization.Name1	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex
Organization.Name2	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex
Organization.Name3	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex
Address.LikelyLinesMatch	Boolean
Address.LinesSimilarity	double
Address.NonEmptyAddressArguments	integer
Address.NonEmptyPremiseArguments	integer
Address.Postcode	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex

Address.PremiseMatch	Boolean
Address.PremiseSimilarity	double
Postcode.Postcode	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex
Postcode.Similarity	double
Telephone.Telephone	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex
Email.Email	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex
DateOfBirth.DateOfBirth	MatchingMatrixIndex, see Enumeration E: MatchingMatrixIndex

MultiLevelMatchingResults class

These are the results obtained after calling Engine.ApplyMatchingRules( ), obtained when the comparison results from Engine.Compare( ) are processed in conjunction with the relevant matching matrix.

Here there are five levels – individual, family, household, business, and custom – each of which contains four components – name, organization, address, and postal code. The four components in turn contain two properties, NonEmptyFields and Similarity.

NonEmptyFields can contain either 0, 1, or 2, and specifies the number of fields in the two records that have non-empty values. For example, if Name.NonEmptyFields is 2, then no name has been set in either of the two compared records; if 1, then only one name has been set in one of the two compared records and the other is blank; or if 0, then both names have been set.

Similarity is a real number in the range 0 to 1, and is the value from the relevant lookup matching matrix. A value of 1 is defined as a sure match between fields of the two compared records (e.g. two names are identical or determined to be identical) and 0 is defined as no match, with intermediate values indicating the likelihood that two fields are a match or not. Similarity is multiplied by the level's relevant weight to produce a final score (see the Scores class) in the range 0 to the actual weight (e.g. by default the final name score is a real number between 0 and 60 inclusive).

MultiLevelMatchingResults Properties

Properties	*Type*
IndividualLevel.Name.NonEmptyFields	integer
IndividualLevel.Name.Similarity	double
IndividualLevel.Organization.NonEmptyFields	integer
IndividualLevel.Organization.Similarity	double
IndividualLevel.Address.NonEmptyFields	integer
IndividualLevel.Address.Similarity	double
IndividualLevel.Postcode.NonEmptyFields	integer
IndividualLevel.Postcode.Similarity	double
IndividualLevel.Telephone.NonEmptyFields	integer
IndividualLevel.Telephone.Similarity	double
IndividualLevel.Email.NonEmptyFields	integer
IndividualLevel.Email.Similarity	double
IndividualLevel.DateOfBirth.NonEmptyFields	integer
IndividualLevel.DateOfBirth.Similarity	double
FamilyLevel…	(as

	IndividualLevel)
HouseholdLevel…	(as IndividualLevel)
BusinessLevel…	(as IndividualLevel)
CustomLevel…	(as IndividualLevel)

MultiLevelMatchingResults Usage

Individual Level

This will find matches at an individual level e.g. John Smith and Mary Smith living at the same address will not be matched, nor will John Smith and James Smith. However, John Smith and Mr J E Smith will be matched and (by default) John Smith and E J Smith will be regarded as a possible match, as perhaps E J Smith is known by his middle name.

Family Level

This finds matches on surname at the same address e.g. John and Mary Smith at the same address will be matched, as will John Smith and James Smith, and all but one record will be flagged.

Household Level

This matches records with the same address, regardless of surname e.g. John Smith and Lucy Jones living at the same address will be matched.

Business Level

This level is used to produce one record per company or business. Therefore, two different employees working for the same company will be matched, as long as the addresses and postcodes match well enough.

Scores class

The final scores are obtained when Engine.Score( ) is run with the processed multi-level matching results (see MultiLevelMatchingResults).

Here there are also five levels – individual, family, household, business, and custom – each of which contains five score components – name, organization, address, postal code, and total. Total is the sum of the other four components, and is a value ranging from 0 to a maximum; this maximum possible matching score is the sum of four weights (e.g. the name weight is specified by using Engine.Settings.MatchingRules.IndividualLevel.Weights.Name.SetMatrixWeights).

For example, the default US matching weights for IndividualLevel are 60, 0, 40, and 30 (name, organization, address, and postal code respectively), so the maximum possible matching score for IndividualLevel.Total is therefore 130. NB: for US zip codes, the postal code score is restricted by design to a Likely match even if identical, unless the zip codes include the "Plus 4" element and are identical. For UK data, the weights are the same as for US data except for address, where the maximum score is set to 30 – this is because UK postcodes are usually full postcodes which are at a much lower level than a US zip code, making them worth more in relation to the other components of the data. So, the maximum possible matching score for IndividualLevel.Total for UK data using the default weights is therefore 120.

Scores Properties

Properties	*Type*
IndividualLevel.Name	double
IndividualLevel.Organization	double
IndividualLevel.Address	double
IndividualLevel.Postcode	double
IndividualLevel.Telephone	double
IndividualLevel.Email	double
IndividualLevel.DateOfBirth	double
IndividualLevel.Total	double
FamilyLevel	(as IndividualLevel)
HouseholdLevel	(as IndividualLevel)
BusinessLevel	(as IndividualLevel)
CustomLevel	(as IndividualLevel)

Names and Words Table

The various names and words are contained in the DAT files that are provided with the matchIT API and must be installed on any system that the matchIT API is installed on. The Names and Names2 DAT files delivered with the matchIT API are the only ones that the user will usually want to configure. These tables control:

- the matching equivalent of words e.g. Tony = Anthony
- the gender of forenames e.g. John = Male, Susan = Female, Chris = Either
- casing rules e.g. PO Box, IBM, helpIT
- expansion/contraction of abbreviations and correction of typing errors e.g. Svcs = Services, Finacial = Financial
- attributing type to these and other words e.g. Mr = Prefix, Ltd = Business, FL = State, The = Noise.

These are fixed-width text files, by default in the DAT sub-directory; the layout of the NAMES.DAT file is as follows:

Property	Width	Description
TYPE	1	Type of entry – see below
NAME	25	Matching equivalent of the entry (e.g. 'Anthony' has a matching equivalent of 'Tony', enabling these two names to be matched)
EQUIVALENT	10	The word which is actually looked up
GENDER	1	Indicates the gender of the forename or prefix
SALUTATION	2	Indicates the type of salutation to be generated for a particular prefix – see below
PROPER CASE	30	Proper case value for the entry
SWITCH	1	Indicates whether this entry is the first part of a two word lookup

Note: matchIT will only look up the word in the Equivalent column, not the Name column. This means that all names must have an entry with name equal to Equivalent.

The different types that can be entered in the table are as follows:

'A' Address Word, such as "Rd" or "Street"

'B' Business word, such as "Ltd" or "Printers"

'C' UK county, such as "Kent" or "Glos"

'E' Exclusion word, such as "Deceased" or "Moved"

'F' Female forename (note the gender has to be set for these entries too)

'I' Initials, such as "E" or "W"; these entries are in the table as they may be the first part of a two word phrase, such as "E Midlands"

'J' Job title word, such as "Manager"

'M' Male forename (note the gender has to be set for these entries too)

'N' Noise word (i.e. ignored when generating keys or address matching), such as “The” or “House”

'O' Overseas i.e. foreign country

'L' Local country, such as "UK" or "Scotland"; this enables local countries to be identified as countries, without the record being marked as foreign

'P' Prefix, such as "Mr" or "Captain" (note the gender has to be set for these entries too, also the SALUTATION TYPE – see below)

'Q' Qualification word, such as "PhD" or "ARICS"; these entries typically always need a proper case entry as casing of qualifications can be unusual

'S' Special casing word, i.e. a word that is cased unusually but doesn't fall into any of the above categories, such as "PhotoMe"

T State or province, such as "Pennsylvania" or "PA"

'U' Unknown word; this is for the first word of a two word phrase, which, on their own, have no special meaning, such as the "Hong" in "Hong Kong"

Each prefix entry must have a salutation type associated with it. The following list shows the salutation types, along with an example of the type of salutation that will be generated:

Type	Rule	Example
S	Dear Prefix Surname	Dear Mr Smith
C	Dear Prefix Surname	Dear Mr Smith
FS	Dear Prefix Forename Surname	Dear Mr John Smith
FF	Dear Forename	Dear John
F	Dear Prefix Forename	Dear Sir John
B	Dear Prefix	Dear Sir
T	Prefix	My Lord

Salutation type C is different from type S in that it is treated as a name even if it is found in address lines 1 or 2 with **Scan Address Lines for Names** set. This means that if the option is switched on and e.g. MR has salutation type C, then Mr J Smith would be identified as a name in address line 1 or 2, whereas if MR has salutation type S, then it would not be identified.

Additionally, each prefix, male forename and female forename must have a gender associated with it, taking a value of either ‘M’ (Male), ‘F’ (Female), or ‘E’ (Either).

These tables are in a standard fixed width format that you can edit, but a maintenance program can be supplied on CD, which allows you to easily edit and maintain the NAMES.DAT and NAMES2.DAT files.

Note: if you inadvertently change the record length or field positions of these files, it may cause a failure in the matchIT API.

Surnames Table

SURNAMES.DAT - used for casing surname prefixes such as "de" in Charles de Gaulle. To add or modify entries, follow the layout of the existing entries.

Towns Table

TOWNS.DAT - used for extracting towns from address lines to a specific Town field, also for upper casing Towns. This file is available for UK "post towns" only i.e. defined as such by Royal Mail.

Coding Example (Visual Basic)

Scalable sample applications are available in the SDKs which accompany the matchIT API, written in VB.Net and C++. The example shown below is simplified, in that it has no screen or database interaction. The properties returned by the Engine method would usually be stored in a database table for efficiency.

Comparing two records

```
'Load the data lookup tables...
Dim tables As matchIT.NamesAndWords
Set tables = New matchIT.NamesAndWords
tables.Initialize "C:\Tables", UK

'Initialize the matchIT engine...
Dim engine As matchIT.Engine
Set engine = New matchIT.Engine
engine.Initialize tables

'Configure engine settings here

'Create and set the first record...
Dim rec1 As matchIT.Record
Set rec1 = New matchIT.Record
rec1.InputFields.Addressee.FullName = "Bill Dayton"
rec1.InputFields.Address.Lines.Line1 = "12 Greenridge Ave"
rec1.InputFields.Address.Lines.Line2 = "White Plains"
rec1.InputFields.Address.Elements.Region = "NY"
rec1.InputFields.Address.Elements.Postcode = "10605-1423"
engine.Generate rec1.InputFields, rec1

'Create and set the second record...
Dim rec2 As matchIT.Record
Set rec2 = New matchIT.Record
rec2.InputFields.Addressee.FullName = "w r deighton jr"
rec2.InputFields.Address.Lines.Line1 = "12 green ridge
avenue"
```

```
rec2.InputFields.Address.Lines.Line2 = "white planes"
rec2.InputFields.Address.Elements.Postcode = "10605"
engine.Generate rec2.InputFields, rec2

'Compare the two records...
Dim result As matchIT.CompareResult
Set result = New matchIT.CompareResult
engine.Compare rec1, rec2, result

'Apply the matching rules to the results...
Dim mlmresult As matchIT.MultiLevelMatchingResults
Set mlmresult = New matchIT.MultiLevelMatchingResults
engine.ApplyMatchingRules result, mlmresult

'Retrieve the final scores...
Dim scores As matchIT.Scores
Set scores = New matchIT.Scores
engine.Score mlmresult, scores

'Do the records match well enough?
If scores.IndividualLevel.Name >= 80 Then
      'Yes, the records do match well enough
      '(by default the total IndividualLevel score using US
weights will be 98, with 40 for the name, 38 for the
address, and 20 for the postal code)
End If
```

Sample Record

Sample of matchIT API Generate processing

This sample record is presented as a guide to the functionality of the matchIT API. It shows in detail the field contents before and after processing.

Field Name	*Before (Input)*	*After (Output)*
Addressee.FullName	TONY R MACKAY JR	Tony R MacKay Jr
Addressee.NameElements.Prefix		Mr
Addressee.NameElements.FirstNames		Tony R
Addressee.NameElements.LastName		MacKay
Addressee.NameElements.Suffix		Jr
Addressee.Contact		Mr Tony R MacKay Jr
Addressee.Salutation		Dear Mr MacKay
Gender		Male
NormalizedName		MACKAY,ANTHONY,R
NameKey		mykyA
PhoneticLastName		myky
PhoneticFirstName		ymTymy
PhoneticMiddleName		r
Address.Lines.Line1	12 GREENRIDGE AVE	12 Greenridge Ave

Field Name	*Before (Input)*	*After (Output)*
Address.Lines.Line2	WHITE PLAINS	White Plains
Address.Lines.Line3	NY 10605	
Address.Lines.Line4	USA	
Address.Lines.Line5		
Address.Elements.Postcode		10605
Address.Elements.Region		NY
Address.Elements.Country		USA
Address.Elements.Premise		12
Address.Elements.Thoroughfare		Greenridge Ave
AddressKey		wytpgrym
Telephone.Number	(914) 232-0908	(914) 232-0908
TelAreaCode		914
TelLocalCode		2320908

Note: To achieve these exact results, the following matchIT API settings must be set before calling the Engine.Generate( ) method:

- EngineSettings.Generate.Address.Extract.Premise = CopyExtract
- EngineSettings.Generate.Address.Extract.Thoroughfare = CopyExtract
- EngineSettings.Generate.Address.Extract.Region = MoveExtract
- EngineSettings.Generate.Address.Extract.Country = MoveExtract
- EngineSettings.Generate.Address.Extract.Postcode = MoveExtract

Appendix A: Enumerations

Enumeration A: Country

0	UnknownCountry
1	UK
2	USA
3	Ireland
4	France
5	Germany
6	Spain
7	Portugal
8	Sweden
9	Denmark
10	Norway
11	Australia
12	NewZealand
13	Austria
14	Switzerland
15	Netherlands

Enumeration B: Extraction

0	Leave
1	CopyExtract
2	MoveExtract

Enumeration C: FieldType

0	UnknownField
1	NameField
2	TitleField
3	FirstNamesField
4	InitialsField
5	LastNameField
6	OrganizationField
7	AddressField
8	CountryField
9	PostcodeField
10	DPS_Field
11	JobTitleField
12	DepartmentField
13	TelephoneField
14	NoiseField
15	RegionField
16	RegionAndPostcodeField
17	TownAndRegionField
18	TownRegionAndPostcodeField
19	QualificationField
20	EmailField
21	SalutationField

Enumeration D: Gender

0	Male
1	Female
2	Either
3	Inconsistent
4	Unknown

Enumeration E: MatchingMatrixIndex

0	Equal
1	SoundsEqual
2	BothEmpty
3	OneEmpty
4	Approx
5	SoundsApprox
6	Contains
7	Unequal

Enumeration F: PhoneticAlgorithm

0	None
1	Soundex
2	SoundIT
3	LooseSoundIT
4	DynamicSoundIT

Contact Details

For help on using the matchIT API, please contact:

USA

helpIT systems inc.
560 So. Winchester Blvd, 5th Floor
San Jose, CA 95128
United States

Toll Free: 866 matchIT (866 628 2448)
International: +1 408 236 7489
Fax: +1 408 236 7491
Email: support.us@helpit.com

UK

helpIT systems ltd.
Swan House
24 Bridge Street
Leatherhead
Surrey
KT22 8BX
United Kingdom

Tel: +44 (0)1372 225904
Fax: +44 (0)1372 360081
Email: support@helpit.com

Website

www.helpIT.com