**TransFort – Common Match Library**

|  |  |
| --- | --- |
| Requirements | Abhishek Jain |
| Design | Surendra Duggirala |
| Review | Sandip Bose |

Introduction

Comparison of user verification data in TransFort is currently done by each callout. This results in additional effort spending in each callout development as well as makes the comparison error prone in case callout developer overlooks any aspects of comparison logic.

Current comparison logic also does not include facility to perform change in comparison behavior like less strict name comparison based on configuration, differently formatted date comparison. In addition, current comparison logic also does not include configuration based what if comparisons which though will not affect the end result but can provide an analysis of the reason for comparison failures, helping in improving issuer’s success rates.

Performing all these changes in numerous callout will be time consuming and error prone without a central infrastructure. Hence this document proposes a common match library to provide match function for most commonly used verification fields that can be used by callouts to perform verification matches. This will also help in reducing callout development time for future callouts and add to quality by being confident about the comparison logic.

Requirements

Common match library should address generic and field level specific requirements as mentioned below. List of most commonly used verification fields is provided as part of specific requirements section:

Generic Requirements:

1. All match functions should be available for encrypted as well as hashed data (PII project).
2. All match functions should be configurable for different format matches based on the field data type. It should cover all type of possible match criterions for specific match field it deals with.
3. All match functions should be configurable for “what if” match analysis providing result for actual configured match but suggestive result for configured “what if” match scenario (logging and analysis purposes).
4. Possibility of configuring Regex like match rules.
5. Generic match functions for string/alpha-numeric field comparison e.g. address, driving licence, Mother’s maiden name (MMN), city, state, country, Hint Answer etc.
6. Generic match functions for purely numeric field comparison e.g. account number, credit limit, OTP, registration code, ATM pin etc.
7. Generic match function for Boolean match values e.g. yes, true, 1 TRUE, YES, should all be treated as boolean true
8. We also need to consider the possibility of achieving all such combinations in hashed data scenario. **Please note that this might require change in upstream upload process (client/server)**

List of most common verification fields and Specific Requirements:

1. **Cardholder name**:

This is the most complex verification field in terms of match logic and the function design should be carefully contemplated to address all kind of match scenarios including (but not limited to):

* 1. Exact match
  2. Match without spaces
  3. Match without special characters and/or spaces
  4. Inexact match e.g. by dropping middle name, last name
  5. Case insensitive match
  6. Match with/without titles like Jr./Sr./Mr.
  7. Partial name match
  8. All the permutations should be achievable through configuration
  9. Possibility of configuring a combination of match scenarios and marking few of them as “what if” only results

1. **Date of Birth**:
   1. Exact match in configured format, say dd/mm/yyyy
   2. Match without century part of year, i.e. dd/mm/yy (cardholder will still be entering all four digits of year
   3. Match with dd and mm reversed (this can be very useful for “what if”)
   4. Detection of separator mismatch
2. **Postal Code**:
   1. Exact match
   2. Match without spaces
   3. Match without separators e.g. – (hyphen), / (slash)
   4. Country/region specific regex/partial match possibility, e.g. a part of the code might be widely considered optional or a particular character may be considered as good as some other character.
3. **SSN**:
   1. Last 4 and last 6 match assuming the server returned/DB stored value is either complete or just last 4/6.
   2. “What if” match within the DB value to see if user value could have matched at a different position.
   3. Picking only last 4/6 from user input if the user input value is longer than configured.
4. **Email**:
   1. Case insensitive match only.
   2. Correction of minor errors like two consecutive @ signs or two dots in user entered value before comparing.
5. **Home/Mobile Phone**:
   1. Match with/without country code.
   2. Match with/without Area code.
   3. Match without spaces, hyphens.

Design/Usage

Match library provides options for matching different types of user entered field values with the stored data or data from backend interfaces. User entered field value is referred as user input data and the data which is stored in the database or coming from backend interfaces is referred as stored data. All methods return a result with a Boolean result as well as a description about the Boolean result.

MatchLibrary has been designed keeping in mind the most common usage while creating callouts. For that purpose the basic flow is as simple as it gets. The developer just creates the Fields that are to matched and then uses the common match() method to compare them and fetch all the results in one go. In case of fields not present in the Library, the developer can extend the Field class (or one of the subclasses) and create a Custom field.

Developer Usage

Prerequisites

* Identify all the fields that are to be matched using MatchLibrary. Ideally all the fields where matching is done by the callout should qualify.
* For every Field add the appropriate Callout Config. This callout config will tell the Library on what parameters the match is to be done. For e.g. L4 means “Last 4 digits” for SSN. For details, check the Field Details section at the end of the document.
* In case you are editing an existing callout with old match logic, add MATCHLIB\_ENABLED to Callout and set its value to YES. This provides an easy toggle for the customer if he wants to disable the MatchLibrary due to some issue.

## Callout Code

* If it’s an existing Callout and Match logic has been written, the developer needs to incorporate the new code by adding a condition. The condition should check for MATCHLIB\_ENABLED and use the new code block (written using MatchLibrary) if it’s enabled and the old match logic if it’s not enabled.
* Fetch the user data from PIN and stored data from DB or alternatively from a service as required.
* Create a Field corresponding to each field to be matched. An ArrayList or an Array of Fields can be used if you have multiple fields. For creating a Field you need to call the createField() method which expects Field type, Field name, user data and stored data. Field Type has to be a standard type from the Enum FieldType(See details ahead). The Field name can be any name. Note that this name will be used in the logs, in callout config as well as in the Callout status.
* Once field(s) are created, call MatchLibrary.match(fields) to compare each of the field with the provided Matching logic. The function will result a MatchResult object.
* The MatchResult object gives your matchCount (getMatchCount()) which can help your determine whether the match succeeded or not. If you need specific Field results getMatchResults() returns a Boolean array of all the Match results corresponding to each field.
* The MatchResult also contains the CalloutStatus (getCalloutStatus()) that can be used to update the Callout status.
* Surround your code with try/catch and handle the exceptions that the MatchLibrary does not handle. This can be as simple as logging and sending back a error status.
* When you are modifying an existing callout make sure that you do not edit the existing logic and keep it as an else condition as much as possible.

Functions to be used:

**Method for matching multiple fields in one shot :**

**public static MatchResult match(Field[] fields) throws MatchException**

Returns a MatchResult object after matching all fields passed as an array in a sequence.

**Parameters:**

|  |  |
| --- | --- |
| **Parameter Name** | **Description** |
| **Field[] fields** | An array of Field objects. A factory method **createField** of Field can be used for creating Fields. |

**Returns:**

**MatchResult** : An object of MatchResult class which contains an int value which is the number of fields matched and a boolean array for getting individual field result and callout status.

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| **getMatchCount()** | * Returns the number of fields matched. * If all fields passed are mandatory, the value returned by this method should be equal to the number of fields passed for any callout to return success. * If the callout has a minMatch configured, the value returned by this method should be greater than or equal to the minMatch value for the callout to return success. * For example, if 5 fields are passed to the match method and minMatch is 4, the getMatchCount() method should return 4 or 5. |
| **getCalloutStatus()** | * This should be used as callout status. * The callout status for any field will have the field name (which is used while creating the field) appended with s or f in square brackets. For example : DOB[s] * If multiple formats are configured, the format number will be added in round brackets beside s (only if the match is success). For example : DOB[s(3)] * Format number is optional if the first format matches. DOB[s] means the first format matched. * The WhatIf configurations will be added in curve brackets. For example : DOB[f]{DOB[s(2)] which means the actual format match failed and the 2nd what if format matched. * Example : DOB[s(4)]ED[f]{ED[s(2)]}SSN[s]POSTALCODE[s]CN[s]MOBILE[s]LL[s] |
| **getMatchResults()** | * This returns a boolean array of the individual boolean result of each field in the same order. * This can be used when individual result of any field is required. |

**Throws:**

**MatchException** - If the input fields array is null or contains any null elements.

**Method for creating Field object :**

**public static Field createField(CORequest request, COConfig config, FieldType fieldType, String fieldName, String userData, String storedData) throws MatchException**

**Parameters:**

|  |  |
| --- | --- |
| **Parameter Name** | **Description** |
| **CORequest request** | CORequest. Will be used only for logging. |
| **COConfig config** | COConfig object. Configuration will be taken from this object. |
| **FieldType fieldType** | FieldType enum value which denotes the type of field. Library as a few pre defined fields like Numeric, Date, AlphaNumeric, Name etc. |
| **String fieldName** | Name of the field. Can be anything like DOB, ED, SSN etc. Regularly used fields are defined in the library as constants. Using the already defined constants makes the callout status consistent across banks. |
| **String userData** | User input value. |
| **String storedData** | The data stored in the data base or data coming from backend interface. |

**Returns:**

**Field** : An object of one of the pre defined Field classes based on the fieldType parameter. For example if the field type is Field.FieldType.DATE an object of DateField class will be returned.

**Throws:**

**MatchException** – Exception will be thrown in the below mentioned cases

1. if the fieldName is blank or null.
2. If UserDataFormat is not configured in few cases if UDF is mandatory. Configuration will be taken from the COConfig object.
3. If StoredDataFormat is not configured in few cases if SDF is mandatory. Configuration will be taken from the COConfig object.

**Field Constructor :**

**public Field(CORequest request, COConfig config, String fieldName, String userData, String storedData) throws MatchException**

Different types of fields like Date, Numeric, Card holder name etc are pre defined in the library. Every field to be matched should either use one of these pre defined types or a new type can be defined by overriding the Field class and adding it to the class path. Constructor can be used for creating the object. Parameters, returns and throws are same as createField method.

Configuration values:

Values which should be configured in the callout configuration. [FIELD\_NAME] is the name of field which can be anything like DOB, SSN etc. Regularly used field names are listed in the MatchLibrary class as constants. New fields other than the listed ones can be used without any issue.

**Generic field types :** These are the generic fields which can be used for any type of field. These will be used if the product specific fields (listed in another table) don’t meet the requirement or if match library doesn’t provide a field class for the field to be matched (for new type of fields).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TYPE** | **Enum name** | **Description** | **Class name** | **Configuration** |
| **DATE** | Field.FieldType.DATE | Date type  Can be used for fields like DOB, Expiry date or ValidFrom etc. | com.arcot.callout.matchlibrary.field.impl.DateField | **UDF :** Required (Any date format like ddMMyyyy)  **SDF :** Required (Any date format)  **MF :** Optional (Partial format can be configured like MMyy when SDF/UDF format is ddMMyyyy) |
| **EMAIL** | Field.FieldType.EMAIL | Can be used for fields like email, primary email, secondary email etc. | com.arcot.callout.matchlibrary.field.impl.EmailField | **UDF :** Not Required  **SDF :** Not Required  **MF :** Optional (Flags like ignoreCase, IgnoreSplChars etc. can be configured.) |
| **NAME** | Field.FieldType.NAME | Will be used for all name fields like MotherMaiden name, father name or cardholder name. | com.arcot.callout.matchlibrary.field.impl.NameField | **UDF :** Not Required  **SDF :** Not Required  **MF :** Optional (Flags like ignoreCase, IgnoreSplChars etc. can be configured.) |
| **REGEX** | Field.FieldType.REGEX | For all input fields which should be in a predefined regex format. | com.arcot.callout.matchlibrary.field.impl.RegExField | **UDF :** Not Required  **SDF :** Not Required  **MF :** Optional (Flags like ignoreCase, IgnoreSplChars etc. can be configured.) |
| **NUMERIC** | Field.FieldType.NUMERIC | For any integer, long type fields. | com.arcot.callout.matchlibrary.field.impl.NumericField | **UDF :** Optional (Length of User data can be specified)  **SDF :** Optional (Length of stored data can be specified)  **MF :** Optional (Flags like ignoreNonNumerics, IgnoreSplChars etc. can be configured.) |
| **NUMDOUBLE** | Field.FieldType.NUMERICDOUBLE | For any double type fields like amount. | com.arcot.callout.matchlibrary.field.impl.NumericDoubleField | **UDF :** Optional (Length of User data can be specified)  **SDF :** Optional (Length of stored data can be specified)  **MF :** Optional (Flags like ignoreNonNumerics, IgnoreSplChars etc. can be configured.) |
| **ALPHANUM** | Field.FieldType.ALPHANUMERIC | Alpha numeric like city name  All string fields with only digits and letters. | com.arcot.callout.matchlibrary.field.impl.AlphaNumericField | **UDF :** Optional (Length of User data can be specified)  **SDF :** Optional (Length of stored data can be specified)  **MF :** Optional (Flags like ignoreNonNumerics, IgnoreSplChars etc. can be configured.) |
| **DEFAULT** | Field.FieldType.DEFAULT | Default type  Default field can be used if the field is a general string type. | com.arcot.callout.matchlibrary.field.impl.DefaultField | **UDF :** Optional (Length of User data can be specified)  **SDF :** Optional (Length of stored data can be specified)  **MF :** Optional (Flags like ignoreNonNumerics, IgnoreSplChars etc. can be configured.) |

* Not Required : MatchException with errorCode INVALID\_SDF or INVALID\_UDF will be thrown if added.
* Required : MatchException with errorCode INVALID\_SDF or INVALID\_UDF will be thrown if not added.
* Optional : Can be added based on requirement. No error will be thrown if not configured.

# Field Types

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **Enum name** | **Class name** | **Configuration** |
| **CREDITCARDLIMIT** | Field.FieldType.CREDITCARDLIMIT | com.arcot.callout.matchlibrary.field.impl. CreditCardLimitField (Derived from NumericField) | **UDF :** Not Required  **SDF :** Not Required  **MF :** Optional (Only partial percentage match is allowed) |
| **DrivingLicenseField** | Field.FieldType. DRIVINGLICENCE | com.arcot.callout.matchlibrary.field.impl. DrivingLicenseField  (Derived from AlphaNumericField) | **UDF :** Not Required  **SDF :** Not Required  **MF :** Optional (Partial first, last and any are allowed) |
| **EmailField** | Field.FieldType. EMAIL | com.arcot.callout.matchlibrary.field.impl.EmailField | **UDF :** Not Required  **SDF :** Not Required  **MF :** Optional. No partial match. IGNORECASE, IGNORESPACES turned on by default. |
| **HintAnswerField** | Field.FieldType. HINTANSWER | com.arcot.callout.matchlibrary.field.impl. HintAnswerField  (Derived from AlphaNumericField) | **UDF :** Not Required  **SDF :** Not Required  **MF :** Optional. Partial match restricted. IGNORECASE, IGNORESPACES turned on by default |
| **LastBillAmoutField** | Field.FieldType. LASTBILLAMOUNT | com.arcot.callout.matchlibrary.field.impl. LastBillAmoutField (Derived from NumericDoubleField) | **UDF :** Not Required  **SDF :** Not Required  **MF :** Optional (Only partial percentage match allowed) |
| **OTPField** | Field.FieldType.OTP | com.arcot.callout.matchlibrary.field.impl. OTPField (Derived from NumericField) | UDF : Not Required  SDF : Not Required  MF : Optional (Only partial First match allowed) IGNORESPLCHARS, IGNORESPACES turned on by default |
| **PhoneField** | Field.FieldType.PHONE | com.arcot.callout.matchlibrary.field.impl. OTPField (Derived from NumericField) | UDF : Not Required  SDF : Not Required  MF : Optional (Only partial First match allowed) IGNORESPLCHARS, IGNORESPACES turned on by default |
| **PostalCodeField** | Field.FieldType. POSTALCODE | com.arcot.callout.matchlibrary.field.impl. PostalCodeField (Derived from AlphaNumericField) | UDF : Not Required  SDF : Not Required  MF : Optional (Only partial First, Last Any match allowed) IGNORECASE, IGNORESPACES turned on by default |
| **RegCodeField** | Field.FieldType. REGCODE | com.arcot.callout.matchlibrary.field.impl. RegCodeField (Derived from AlphaNumericField) | UDF : Not Required  SDF : Not Required  MF : Optional (Only partial First, Last Any match allowed) IGNORECASE, IGNORESPACES turned on by default |
| **SSNField** | Field.FieldType.SSN | com.arcot.callout.matchlibrary.field.impl. SSNField (Derived from NumericField) | UDF : Not Required  SDF : Not Required  MF : Optional (Only partial First match allowed) IGNORESPLCHARS, IGNORESPACES turned on by default |
| **NameField** | Field.FieldType.NAME | com.arcot.callout.matchlibrary.field.NameField | UDF : Not Required  SDF : Not Required  MF : Optional (Only partial percentage match allowed) IGNORECASE, IGNORESPACES and IGNORETITLE turned on by default |

# Valid Fields

**[FIELD\_NAME]\_MF :**

* This defines the actual match logic. Match logic with different flags can be set using this config value.
* Match Logic can be a partial date for date field, or partial match (like n digits from start or end) for numeric and alpha numeric fields along with different flags enabled or disabled.
* A similar configuration can be added as WhatIf configuration with a separator ‘~’.

for analysis purpose. This should be encloused in square brackets.

* Examples :

1. POSTALCODE\_MF=L6|IGNORESPACES-y|IGNORESPLCHARS-y; L6 stands for Last 6. All spaces and special characters will be ignored.
2. DOB\_MF=MMyyyy~[MMyy]; Matching only month and year of DOB, match will be true if month and 4digit year matches and MMyy is the WhatIf format. PostalCode is a AlphaNumeric string, Match can be done for last 6 characters (L6) or first 6 characters (F6) or any 6 chracters(A6) or complete string (default).
3. REGCODE\_MF=A6|IGNORESPACES-n ~[ A4|IGNORESPACES-y]; A6 stands for Any 6 characters. Above config returns true if any 6 characters match with exact case. WhatIf matches any 4 characters ignoring case.
4. LASTBILLAMOUNT\_MF=P15; 15% deviation is allowed for LastBillAmount which means if the last bill amount is 1000, the match returns true for any value greater than 850 and less than 1150.

* Different flags like IGNORECASE, IGNORESPLCHARS, IGNORETITLE (only for name field), IGNORENONALPHANUM, IGNORENONNUM etc can be set using this config value.
* Match can be done for Last ‘n’ characters (Ln) or First ‘n’ characters (Fn) or Any ‘n’ characters (An) for Numeric/AlphaNumeric fields. Percentage deviation is also allowed for Numeric fields.
* Can take multiple values separated by ‘~’.

**[FIELD\_NAME]\_UDF : (Mandatory for Date field, optional for all other generic fields and not required for product specific fields)**

* Format of the user data. Ex. DOB\_UDF=ddMMyyyy;
* Mandatory for Date fields and optional for all other fields. Not required for email field.
* This config value will be used to check if the user data is in correct format.
* If the user input doesn’t match the format configured, the result will have the error code ‘INVALID\_UD’. Ex. DOB[INVALID\_UD]
* If an invalid format is configured, callout status will have the error code ‘INVALID\_UDF’. Ex. DOB[INVALID\_UDF]
* CAP and ES are expected to send correct data, hence only single value is allowed here. If the developer finds INVALID\_UD or INVALID\_UDF he should correct the CAP/ES pages.
* The match will not proceed further (like checking WhatIf formats) if any of the above error occurs.
* This config value defines the input data format based on the field type, it would mean the length of the user input data for Numeric/Alphanumeric fields, the data format for a date field etc.
* This field is optional for few fields like Numeric/Alphanumeric fields, not required for fields like Email, RegEx etc and is mandatory for DateField.

**[FIELD\_NAME]\_SDF : (Mandatory for Date field, optional for all other generic fields and not required for product specific fields)**

* Format of stored data or data coming from backend. Ex. DOB\_SDF=MMddyyyy;
* This config value will be used to check if the stored data is in correct format.
* If the stored data doesn’t match the format configured, the result will have the error code ‘INVALID\_SD’. Ex. DOB[INVALID\_SD]
* If an invalid format is configured, callout status will have the error code ‘INVALID\_DF’. Ex. DOB[INVALID\_SDF]
* Can have multiple values configured separated with a ‘~’.
* This config value defines the stored data format based on the field type, it would mean the length of the stored data for Numeric/Alphanumeric fields, the data format for a date field etc.
* This field is optional for few fields like Numeric/Alphanumeric fields, not required for fields like Email, RegEx etc and is mandatory for DateField.

**[FIELD\_NAME]\_ HASHED**

* This will be set to true if this field is hashed.
* This field is not required if all the fields are hashed. Another parameter ALL\_HASHED can be defined if all the fields are hashed.
* For example, if only one field (DOB) is hashed and all other fields are encrypted, then a config property DOB\_HASHED can be added with value true.

**MATCHLIB\_LOG\_LEVEL**

* Can be used for enabling extra logging for all classes of match library. Ex. MATCHLIB\_LOG\_LEVEL=DEBUG;
* Default value is info.
* Valid values are INFO, DEBUG.
* This config param with DEBUG value will be used only in development and testing environments.
* This is MatchLibrary logging and will be common for all fields.
* For debugging purpose this flag can be enabled, which would log a few extra details about the match logic.
* Generic Callout logging framework will be used for match library also. CORequest object is passed as parameter to match methods for this purpose only.
* Sensitive data will be logged if this is enabled in production. This should be strictly disabled in production. The value should be INFO or blank or the property can be removed completely.

**[FIELD\_NAME]\_ SPLCHARSLIST**

* This is to specify the list of characters to be removed from any format or user input value or stored value. Ex. DOB\_SPLCHARLIST=\-.+\_;/
* This will be used only for fields like DOB, Name etc.
* If this is not specified the default set of special characters which is specified in Constants file will be used.

public static final char[] SPECIAL\_CHARS\_ARRAY = new char[]{'\\', '/', '-', '.'};

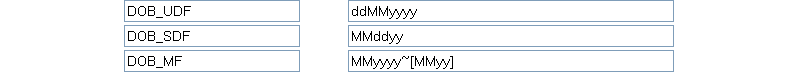
**[FIELD\_NAME]\_ IGNORECASE**

* Flag to configure case sensitive and insensitive match.
* True/false are allowed values.
* True for case sensitive match and false for insensitive match.
* Will be used only for a few fields.

Pre-Defined Fields:

**Date Field:**

* User input data and stored data will be validated with the formats defined. DataFormat (UDF or SDF) will be validated before this. INVALID\_UDF or INVALID\_SDF will be thrown if the format is not fine.
* False with the error description will be returned if one or both user input (INVALID\_UD) and stored data (INVALID\_SD) are not in format configured.
* If MatchFormat (MF) is not configured, user input data will be converted to the StoredDateFormat (SDF) and the returned string will be matched with the stored data. Multiple values can beconfigured for SDF and each format will be validated in a loop.
* If MatchFormat (MF) is configured, both the dates (user input and stored data) will be converted to the match format. True or false will be returned based on these strings.
* Below mentioned DOB matching configuration with input data format ddMMyyyy and stored data format MMddyy. MatchFormat is provided in this case hence only MMyyyy will be matched (after converting both the dates to MMyyyy format). If this match fails, MMyy will be checked but just for analsys purpose.



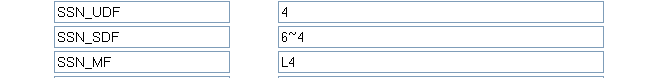
* Another example for validating the expiry date. CAP/ES sends the expiry date in MMyyyy format and the stored data can be in MMyyyy or MMyy formats.



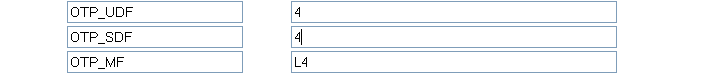
* Special characters can be configured using the SPLCHARLIST config value. If SPLCHARLIST config value is not configured default special characters {'\\', '/', '-', '.'} will be removed.

**Numeric Field:**

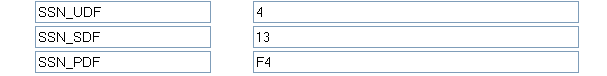
* User input data format and stored data format will be validated. Only integers are allowed as data formats. False will be returned with error description if one or both are not integers. UDF/SDF/MF are not mandatory config values. Can be added only if data length validation is required.
* MatchFormat (MF) can be configured to match the First ‘n’ digits, or Last ‘n’ digits or Any ‘n’ digits or a percentage deviation can be specified.
* If both user data format and stored data formats are configured properly, user input data and stored data are validated. Validation is to check if the data is numeric or not and with defined length (if configured).
* If both input and stored data are valid numeric strings, both are validated and true or false is returned.
* If MatchFormat is configured with first or last ‘n’ number of characters, both user input and stored data will be truncated and will be matched.
* SSN configuration example with 4 digit numeric user input and stored data can be of 6 or 4 digits. Only last 4 digits will be matched.



* OTP configuration where all the lengths are 4.



* Below mentioned SSN configuration with user input 4 numeric digits and stored data 13 digits and only first 4 are to be matched.

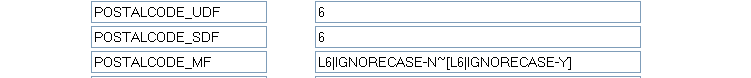


* Last month bill amount configuration. UDF and SDF are not defined here as this is an amount and the number of digits is not standard. Only validation will be to check if the entered data is numeric or not. 15% deviation is added which means a value greater than 15% of the stored value and less than 15% of the stored value will be matched.



**Alphanumeric Field:**

* Same as the Numeric field, the only difference is in validation logic of the input and stored data.
* Percentage match is not allowed in this case.
* Postal code example with 6 characters input and stored data, last 6 characters to be matched with the same case. WhatIf is configured to check if the case insensitive match results in true.



**Email Field:**

* Email doesn’t need any special configuration.
* Special characters will be ignored by default. MF can be configured if the special characters are to be retained.
* User input data will be validated with the hard coded regex format.
* Default regex is "^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,6}$".
* Sample email configuration which turns off the IGNORESPLCHARS flag. Turning it on for WhatIf configuration.



**Name Field:**

* This should be used for matching card holder name or father name or mother maiden name.
* Ex NAME\_DF=EXACTMATCH-N|IGNORESPACES-Y|SHUFFLE-Y|IGNORETITLE-Y|IGNORECASE-Y
* Default values of each config value are EXACTMATCH = true, IGNORESPACES = false, SHUFFLE = false, IGNORETITLE = false and IGNORECASE = true.
* The first word will be treated as title if it is among the below list

String[] titles = new String[]{"MR", "MR.", "DR", "DR.", "SR", "SR.", "JR", "JR.", "MISS", "MISS.", "MRS", "MRS.", "PROF", "PROF."};

* Exact match will be after ignoring case and special characters based on the flags configured.
* Ignore spaces, shuffle, ignore title, ignore spl characters config values can be configured as a single value for name as below. Example of card holder name field.



* ‘|’ will be used as a separator for two config values and ‘-‘ will be used to separate the key and value pair of a config value.

**RegEx Field:**

* A common reg-ex format can be configured for both input and stored data format.
* False will be returned with error description if one or both of them are not in configured format.
* DF should be configured instead of SDF and UDF or both should have same value. False will be returned with invalid config error if UDF and SDF are different.
* Below mentioned SSN field configured as regex.



**Default Field:**

* Ignore case, ignore spaces, ignore special characters, ignore non alpha numerics, ignore numerics flags can be configured as a single config value similar to name field.
* IGNORECASE-y|IGNORESPACES-y| IGNORESPLCHARS-y|IGNORENONNUM-y| IGNORENUM-| IGNORENONALPHANUM-y|
* Default values are for the above flags are as below.

IGNORECASE = true, IGNORESPACES = true, IGNORESPLCHARS = true, IGNORENONNUM = false, IGNORENUM = false, IGNORENONALPHANUM = false

* Example callout configuration for default field.



* If none of the config values are added (including XYZ\_DF) the default behaviour would be matching the exact strings, then changing the case of both strings to upper case and matching, then ignoring special characters and matching (as default value for IGNORECASE and IGNORESPLCHARS is true).

**Result : Result should be taken from the MatchResult object which is returned by the MatchLibrary.**

Code snippet for matching different fields:

String fieldName[] = **new** String[11];

String storedData[] = **new** String[11];

String userData[] = **new** String[11];

fieldName[0] = MatchLibrary.*FIELD\_DOB*;

fieldName[1] = MatchLibrary.*FIELD\_EXPIRY\_DATE*;

fieldName[2] = MatchLibrary.*FIELD\_SSN*;

fieldName[3] = MatchLibrary.*FIELD\_POSTALCODE*;

fieldName[4] = MatchLibrary.*FIELD\_CARDHOLDERNAME*;

fieldName[5] = MatchLibrary.*FIELD\_MOBILE*;

fieldName[6] = MatchLibrary.*FIELD\_LANDLINE*;

fieldName[7] = MatchLibrary.*FIELD\_EMAIL*;

fieldName[8] = MatchLibrary.*FIELD\_SECONDARYEMAIL*;

fieldName[9] = "FAVOURITECITY";

fieldName[10] = MatchLibrary.*FIELD\_LASTBILLAMOUNT*;

userData[0] = "22072011";

userData[1] = "1208";

userData[2] = "9876";

userData[3] = "543-210";

userData[4] = "Mr. M S DHONI";

userData[5] = "919880123456";

userData[6] = "08023456789";

userData[7] = "asfd. lkj@gmail.com";

userData[8] = "asfd.\\ lkj@gmail.com";

userData[9] = "Hyderabad-17";

userData[10] = "100.10";

storedData[0] = "02011.JULY.22";

storedData[1] = "0812";

storedData[2] = "9876";

storedData[3] = "543-210";

storedData[4] = "Mr. M S Dhoni";

storedData[5] = "9880123456";

storedData[6] = "08023456789";

storedData[7] = "asfd.lkj@gmail.com";

storedData[8] = "asfd.lkj@gmail.com";

storedData[9] = "HYDERABAD";

storedData[10] = "100.10";

Field fields[] = **new** Field[11];

MatchResult matchResult = **null**;

**try** {

fields[0] = Field.*createField*(request, config, Field.FieldType.*DATE*, fieldName[0], userData[0], storedData[0]);

fields[1] = Field.*createField*(request, config, Field.FieldType.*DATE*, fieldName[1], userData[1], storedData[1]);

fields[2] = Field.*createField*(request, config, Field.FieldType.*NUMERIC*, fieldName[2], userData[2], storedData[2]);

fields[3] = Field.*createField*(request, config, Field.FieldType.*REGEX*, fieldName[3], userData[3], storedData[3]);

fields[4] = Field.*createField*(request, config, Field.FieldType.*NAME*, fieldName[4], userData[4], storedData[4]);

fields[5] = Field.*createField*(request, config, Field.FieldType.*NUMERIC*, fieldName[5], userData[5], storedData[5]);

fields[6] = Field.*createField*(request, config, Field.FieldType.*NUMERIC*, fieldName[6], userData[6], storedData[6]);

fields[7] = Field.*createField*(request, config, Field.FieldType.*EMAIL*, fieldName[7], userData[7], storedData[7]);

fields[8] = Field.*createField*(request, config, Field.FieldType.*REGEX*, fieldName[8], userData[8], storedData[8]);

fields[9] = Field.*createField*(request, config, Field.FieldType.*DEFAULT*, fieldName[9], userData[9], storedData[9]);

fields[10] = Field.*createField*(request, config, Field.FieldType.*NUMERICDOUBLE*, fieldName[10], userData[10], storedData[10]);

matchResult = MatchLibrary.*match*(fields);

} **catch** (MatchException e) {

e.printStackTrace();

}

System.*out*.println("MatchResult :"+ matchResult);

System.*out*.println("Number of fields matched :"+ matchResult.getMatchCount());

System.*out*.println("Callout status :"+ matchResult.getCalloutStatus());

Match Library Check in Location:

\\profsvc\transfort\PSLibs\Java\src\MatchLibrary

Field level development effort for hashed scenario (assuming engg. team is ready with ES/DUT changes)

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Activity** | **Owner** | **Effort (Person Days)** |
| 1 | Cardholder name | Apps | 8 |
| 2 | Date of Birth | Apps | 1 |
| 3 | Email | Apps | 1 |
| 4 | Home/Mobile Phone | Apps | 1 |
| 5 | Driving license + Hint answer + Registration code | Apps | 2 |
| 6 | Credit limit + Last month statement + SSN + Postal Code | Apps | 3 |