Oracle® Healthcare Master Person Index

Match Engine Reference

Release 2.0.11

E25254-04

April 2016

ORACLE

Oracle Healthcare Master Person Index Match Engine Reference, Release 2.0.11

E25254-04

Contents

4 Setting Match Field Variations and Agreement/Disagreement

5 OHMPI Match Engine Comparison Functions

6 Creating Custom Comparators for the OHMPI Match Engine

Preface

This user's guide provides conceptual and procedural information for configuring matching extensions in an Oracle Healthcare Master Person Index (OHMPI) project. This includes conceptual information about possible configurations used when creating custom comparators for the OHMPI Match Engine, as well as setting up variations in sets of match fields and agreement / disagreement weights at runtime which enable the OHMPI Match Engine to compute more accurate matches when a specific behavior is desired.

Audience

This document is intended for users of OHMPI applications that require data comparison to evaluate and confirm the possibility of data matches.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information and instructions for implementing and using a master person index application, see the following documents in the Oracle Healthcare Master Person Index Release 2.0 documentation set:

- *Oracle Healthcare Master Person Index Installation Guide*
- *Oracle Healthcare Master Person Index Release Notes*
- *Oracle Healthcare Master Person Index User's Guide*
- *Oracle Healthcare Master Person Index Working With IHE Profiles User's Guide*
- *Oracle Healthcare Master Person Index Message Processing Reference*
- *Oracle Healthcare Master Person Index Configuration Guide*
- *Oracle Healthcare Master Person Index Configuration Reference*

- *Oracle Healthcare Master Person Index Data Manager User's Guide*
- *Oracle Healthcare Master Person Index Standardization Engine Reference*
- *Oracle Healthcare Master Person Index Analyzing and Cleansing Data User's Guide*
- *Oracle Healthcare Master Person Index Command Line Reports and Database Management User's Guide*
- *Oracle Healthcare Master Person Index Loading the Initial Data Set User's Guide*
- *Oracle Healthcare Master Person Index WebLogic User's Guide*
- *Oracle Healthcare Master Person Index Provider Index User's Guide*
- *Oracle Healthcare Master Person Index Australia Patient Solution User's Guide*
- *Oracle Healthcare Master Person Index United Kingdom Patient Solution User's Guide*
- *Oracle Healthcare Master Person Index United States Patient Solution User's Guide*

Note: These documents are designed to be used together when implementing a master index application.

Finding Information and Patches on My Oracle Support

Your source for the latest information about Oracle Clinical is Oracle Support's self-service Web site My Oracle Support (formerly MetaLink).

Before you install and use Oracle Clinical, always visit the My Oracle Support Web site for the latest information, including alerts, White Papers, installation verification (smoke) tests, bulletins, and patches.

Creating a My Oracle Support Account

You must register at My Oracle Support to obtain a user name and password account before you can enter the Web site.

To register for My Oracle Support:

1. Open a Web browser to `https://support.oracle.com`.
2. Click the **Register here** link to create a My Oracle Support account. The registration page opens.
3. Follow the instructions on the registration page.

Signing In to My Oracle Support

To sign in to My Oracle Support:

1. Open a Web browser to `https://support.oracle.com`.
2. Click **Sign In.**
3. Enter your user name and password.
4. Click **Go** to open the My Oracle Support home page.

Finding Information on My Oracle Support

There are many ways to find information on My Oracle Support.

Searching by Article ID

The fastest way to search for information, including alerts, White Papers, installation verification (smoke) tests, and bulletins is by the article ID number, if you know it.

To search by article ID:

1. Sign in to My Oracle Support at https://support.oracle.com.
2. Locate the Search box in the upper right corner of the My Oracle Support page.
3. Click the sources icon to the left of the search box, and then select **Article ID** from the list.
4. Enter the article ID number in the text box.
5. Click the magnifying glass icon to the right of the search box (or press the Enter key) to execute your search.

   The Knowledge page displays the results of your search. If the article is found, click the link to view the abstract, text, attachments, and related products.

Searching by Product and Topic

You can use the following My Oracle Support tools to browse and search the knowledge base:

- Product Focus — On the Knowledge page under Select Product, type part of the product name and the system immediately filters the product list by the letters you have typed. (You do not need to type "Oracle.") Select the product you want from the filtered list and then use other search or browse tools to find the information you need.
- Advanced Search — You can specify one or more search criteria, such as source, exact phrase, and related product, to find information. This option is available from the **Advanced** link on almost all pages.

Finding Patches on My Oracle Support

Be sure to check My Oracle Support for the latest patches, if any, for your product. You can search for patches by patch ID or number, or by product or family.

To locate and download a patch:

1. Sign in to My Oracle Support at https://support.oracle.com.
2. Click the **Patches & Updates** tab. The Patches & Updates page opens and displays the Patch Search region. You have the following options:
   - In the **Patch ID or Number is** field, enter the number of the patch you want. (This number is the same as the primary bug number fixed by the patch.) This option is useful if you already know the patch number.
   - To find a patch by product name, release, and platform, click the **Product or Family** link to enter one or more search criteria.
3. Click **Search** to execute your query. The Patch Search Results page opens.
4. Click the patch ID number. The system displays details about the patch. In addition, you can view the Read Me file before downloading the patch.
5. Click **Download.** Follow the instructions on the screen to download, save, and install the patch files.

Finding Oracle Documentation

The Oracle Web site contains links to all Oracle user and reference documentation. You can view or download a single document or an entire product library.

Finding Oracle Health Sciences Documentation

To get user documentation for Oracle Health Sciences applications, go to the Oracle Health Sciences documentation page at:

http://www.oracle.com/technetwork/documentation/hsgbu-154445.html

---

Note: Always check the Oracle Health Sciences Documentation page to ensure you have the latest updates to the documentation.

Finding Other Oracle Documentation

To get user documentation for other Oracle products:

1. Go to the following Web page:

   http://www.oracle.com/technology/documentation/index.html

   Alternatively, you can go to http://www.oracle.com, point to the Support tab, and then click **Documentation**.

2. Scroll to the product you need and click the link.
3. Click the link for the documentation you need.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Oracle Healthcare Master Person Index Match Engine Reference

This chapter introduces you conceptual information about the Oracle Healthcare Master Person Index (OHMPI) Match Engine and how it matches data in a master person index application. It also introduces you to the OHMPI Standardization Engine, with which the OHMPI Match Engine works closely. For more information about the standardization engine, see *Oracle Healthcare Master Person Index Standardization Engine Reference*.

This chapter includes the following sections:

Learning About the OHMPI Match Engine

The OHMPI Match Engine provides record matching capabilities for external applications, such as master person index applications. It works best along with the OHMPI Standardization Engine, which provides the preprocessing of data that is required for accurate matching, such as data parsing, data standardization, and also the OHMPI phonetic encoders. Before records can be compared to evaluate the possibility of a match, the data contained in those records must be standardized and in certain cases phonetically encoded. Once the data is conditioned, the match engine determines a match weight for each field defined for matching. The match weight is based on the fields on which matching is performed and how the matching logic is configured. The composite weight is usually the sum of weights generated for all match fields in the records (but could also be a function of the match field weights). The composite weight indicates how closely two records match.

The OHMPI Match Engine is the standard match engine designed to work with the master person index applications created by the Oracle Healthcare Master Person Index. The match engine can also be called from other applications. It is highly configurable in the Oracle Healthcare Master Person Index environment and can be used to match on various types of data. The OHMPI Match Engine works in conjunction with the OHMPI Standardization Engine to improve the quality of your data.

The following sections provide information about matching concepts, the match process, and how the OHMPI Match Engine matches data.

Data Matching Concepts

Data matching compares data stored in disparate systems in and across organizations, helping you reduce data duplication and improve data accuracy. Matching involves comparing specific fields in two standardized records and returning a weight that indicates the likelihood of a match between the two records. A higher weight between two records indicates a greater likelihood of a match. Data matching is based on proven algorithms that are designed to compare different types of data, such as strings, dates, integers, and so on. Matching is a key step in managing data quality, and the algorithms are typically quite complex. Some algorithms are configured to compare more specialized types of data, including first and last names, social security numbers, and dates of various formats.

The following topics provide additional information about standard data matching concepts:

- Deterministic and Probabilistic Data Matching on page 2
- Weighting Thresholds on page 2
- Probabilities and Direct Weights on page 2

Deterministic and Probabilistic Data Matching

Data matching can be either deterministic or probabilistic. In deterministic matching, either unique identifiers for each record are compared to determine a match or an exact comparison is used between fields. Unique identifiers can include national IDs, system IDs, and so on. This can include system IDs, national IDs, and so on. Deterministic matching is generally not completely reliable since in some cases no single field can provide a reliable match between two records. This is where probabilistic, or **fuzzy**, matching comes in. In probabilistic matching, several field values are compared between two records and each field is assigned a weight that indicates how closely the two field values match. The sum of the individual fields weights indicates the likelihood of a match between two records.

Weighting Thresholds

In a data management system, you can set duplicate and match threshold weights. The duplicate threshold is the weight above which two records potentially represent the same entity. The match threshold is the weight above which two records are considered to represent the same entity. Any records below the duplicate threshold are considered to represent completely separate and different entities.

Probabilities and Direct Weights

Optimum (or **ceiling**) matching weights can be assigned to field values using matching (m) and unmatching (u) probabilities or using agreement and disagreements weights in an equivalent way. Both types are based on a logarithmic function. Optimum agreement and disagreement weights are an equivalent logarithmic expression of the matching and unmatching probabilities, but for an end user, defining agreement and disagreement weight ranges is a more direct way to implement m-probabilities and u-probabilities.

Matching and Unmatching Probabilities

When matching and unmatching conditional probabilities are used, the match engine uses a logarithmic formula to determine agreement and disagreement weights between fields. The m-probabilities and u-probabilities you specify determine the maximum agreement weight and minimum disagreement weight for each field, and so define the agreement and disagreement weight ranges for each field and for the entire

record. These probabilities allow you to specify which fields provide the most reliable matching information and which provide the least. For example, in person matching, the gender field is not as reliable as the SSN field for determining a match since a person's SSN is more specific. Therefore, the SSN field should have a higher m-probability than the gender field. The more reliable the field, the greater the m-probability for that field should be.

If a field matches between two records, an agreement weight, determined by the logarithmic formula using the m-probability and u-probability, is added to the composite match weight for the record. If the fields disagree, the logarithmic formula using the m-probability and u-probability is negative, and a disagreement weight is subtracted from the composite match weight.

Agreement and Disagreement Weight Ranges

Like probabilities, the maximum agreement and minimum disagreement weights you define for each field allow you to specify the relative reliability of each field; however, the match weight has a more linear relationship with the numbers you specify. When you use agreement and disagreement weight ranges to determine the match weight, you define a maximum weight for each field when they are in complete agreement and a minimum weight for when they are in complete disagreement. The value assigned to a field is somewhere between the two numbers based on an underlying logarithmic formula. This provides a more convenient and intuitive representation of conditional probabilities.

Using the SSN and gender field example above, the SSN field is assigned a higher maximum agreement weight and a lower minimum disagreement weight than the gender field because it is more reliable. If you assign a maximum agreement weight of "10" and two SSNs match, the match weight for that field is "10". If you assign a minimum disagreement weight of "-10" and two SSNs are in complete disagreement, the match weight for that field is "-10".

Understanding How the OHMPI Match Engine Works

The OHMPI Match Engine compares records containing similar data types by calculating how closely certain fields in the records match. The resulting comparison weight is either a positive or negative numeric value that represents the degree to which the two sets of data are similar. The match engine relies on probabilistic algorithms to compare data of a given type using a comparison function specific to the type of data being compared. The comparison functions for each matching field are defined in a match configuration file that you can customize for the type of data you are indexing. You can also define custom comparison functions to plug in to the match engine. The formula used to determine the matching weight is based on either matching and unmatching probabilities or on agreement and disagreement weight ranges (described in Probabilities and Direct Weights).

Match Cache

Match Cache improves processing, especially in cases where there are multiple child objects and the match fields are from both the primary object and child objects. This means that performance impact due to new a Match Set will be minimal since there is no extra I/O. However, there might be some extra invocation to Match Engine but only if new the Match Sets have new match fields. In such cases some CPU processing will be used by this layer but the performance impact for such cases should be minimal.

The following sections provide additional information about how the OHMPI Match Engine works:

OHMPI Match Engine Structure

The OHMPI Match Engine was designed to be very flexible and generic, allowing you to customize existing matching rules and to define additional rules using Java. The match engine framework allows you to create and plug in custom matching comparison functions, or **comparators**, to the match engine to enable matching against any type of data. The OHMPI Match Engine framework includes two main modules. The real-time module stores the predefined and user-defined Java classes that define the matching comparator logic. The design-time modules stores the configuration and validation classes for the comparators.

The OHMPI Match Engine provides a wide variety of customizable comparators for you to choose from. You can also create comparators in the real-time module, and create new validation and configuration rules in the design-time module. The structure of the design-time module support validations, weighting curves, and class dependencies. There is also an option that allows you load information from a data file and use that information to calculate a matching weight.

OHMPI Match Engine Configuration Files

The OHMPI Match Engine compares two records and returns a match weight indicating the likelihood of a match between the two records based on information provided in configuration files. In a master person index application, the match engine is configured by these two files in the Match Engine node of the master person index project: the matching configuration file (`matchConfigFile.cfg`) and the comparators list (`comparatorsList.xml`). The matching configuration file defines the configuration and parameters for the matching comparator functions and the comparators list defines each comparator available to the match engine.

Matching criteria and logic are defined in the match configuration file in the master person index project (`matchConfigFile.cfg`). The data fields that are sent to the OHMPI Match Engine for matching, known as the **match string**, are defined in the MatchingConfig section of `mefa.xml` in the master person index project. The match engine configuration files define which matching rules to use to process each match field. The match engine provides a comprehensive set of comparator functions, and you can create custom comparators if needed.

OHMPI Match Engine Matching Weight Formulation

The OHMPI Match Engine determines the matching weight between two records by comparing the match string fields between the two records using the rules defined in the match configuration file and taking into account the matching logic specified for each field. The OHMPI Match Engine can use either matching (m) and unmatching (u) conditional probabilities or agreement and disagreement weight ranges to fine-tune the match process. It uses the underlying algorithm to arrive at a match weight for each match string field. The weight generated for each field in the match string indicates the level of match between each field. The weights assigned to each field are then summed together for a total, composite matching weight between the two records. Agreement and disagreement weight ranges or m-probabilities and u-probabilities are defined in the match configuration file.

The m-probabilities and u-probabilities are expressed as double values between one and zero (excluding one and zero) and can have up to 16 decimal points. Agreement and disagreement weights are expressed as double values and can have up to 16 decimal points. When using agreement and disagreement weights, the OHMPI Match Engine assigns a matching weight to each field that falls between the agreement and disagreement weights specified for the field. Thus, the maximum agreement weight between two records is the sum of the defined agreement weights for each field. The minimum disagreement weight is the sum of the defined disagreement weights for each field. For more information about weight calculation, see Determining the Weight Range.

OHMPI Match Engine Data Types

The OHMPI Match Engine is built on a flexible framework that allows you to customize and create matching rules for various types of data. The match engine provides an extensive set of comparison functions for matching on various types of fields, such as numbers, dates, single characters, and so on. The match engine also provides more specialized comparison functions for searching on specific types of data, such as person names, address fields, social security numbers, genders. You can define custom comparison functions and custom standardization logic for different data types or variants on data types. These customizations are easily incorporated into a master person index application, allowing you to completely customize the match and standardization process for your specific data format.

The OHMPI Match Engine and the OHMPI Standardization Engine

The OHMPI Match Engine works with the OHMPI Standardization Engine to provide an accurate comparison of two records. The standardization engine reads input data and determines how to parse, normalize, and standardize the data in order to create a standard set of values to use for match comparison. The standardization engine can standardize free-form text fields, such as street address fields or business names, and separate them into their individual parts, such as house numbers, street names, and so on, allowing the match engine to generate a more accurate weight for free-form data.

Understanding the OHMPI Standardization and Matching Process

In a default Oracle Healthcare Master Person Index implementation, the master person index application uses the OHMPI Match Engine and the OHMPI Standardization Engine to cleanse data in real time. The standardization engine uses configurable pattern-matching logic to identify data and reformat it into a standardized form. The match engine uses a matching algorithm with a proven methodology to process and weight records in the master person index database. By incorporating both standardization and matching capabilities, you can condition data prior to matching. You can also use these capabilities to review legacy data prior to loading it into the database. This review helps you determine data anomalies, invalid or default values, and missing fields.

In a master person index application, both matching and standardization occur when two records are analyzed for the probability of a match. Before matching, certain fields are normalized, parsed, or converted into their phonetic values if necessary. The match fields are then analyzed and weighted according to the rules defined in a match configuration file. The weights for each field are combined to determine the overall matching weight for the two records. After these steps are complete, survivorship is determined by the master person index application based on how the overall matching weight compares to the duplicate and match thresholds of the master person index application.

In a master person index application, the standardization and matching process includes the following steps:

1. The master person index application receives an incoming record.
2. The OHMPI Standardization Engine standardizes the fields specified for parsing, normalization. Phonetic encoding is also performed. These fields are defined in mefa.xml and the rules for standardization are defined in the standardization engine configuration files.
3. The master person index application queries the database for a candidate selection pool (records that are possible matches) using the blocking query specified in master.xml. If the blocking query uses standardized or phonetic fields, the criteria values are obtained from the database.
4. For each possible match, the master person index application creates a match string (based on the match columns in mefa.xml) and sends the string to the OHMPI Match Engine.
5. The OHMPI Match Engine checks the incoming record against each possible match, producing a matching weight for each. Matching is performed using the weighting rules defined in the match configuration file.

2

Match Engine Matching Configuration

This chapter introduces you to the matching configuration files for the Oracle Healthcare Master Person Index (OHMPI) Match Engine, including certain rules for formatting and interdependencies that must be followed. The following sections provide an overview of the two matching configuration files provided, the architecture of those files, and formatting descriptions. They also include an overview of comparison functions used in the match configuration file.

This chapter includes the following sections:

Understanding the OHMPI Match Engine Match Configuration File

The matching configuration files define how the OHMPI Match Engine processes records to assign matching probability weights, allowing the master person index application to identify matches, potential duplicates, and non-matches. The match engine includes two configurable files, the match configuration file and the comparators list. Together these files define additional logic for the OHMPI Match Engine to use when determining the matching probability between two records.

The matching configuration is very flexible, allowing you to customize the matching logic according to the type of data being matched and for the record matching requirements of your business. In a master person index application, the matching configuration files are stored in the master person index project and are located in the Match Engine node of the project. The OHMPI Standardization Engine typically standardizes the data prior to matching, so the match process is performed against the standardized data.

The match configuration file, `matchConfigFile.cfg`, contains the matching logic for each field on which matching is performed. By default, this file defines the matching logic for the three primary data types (person names, business names, and addresses), and can also handle generic data types, such as dates, numbers, social security numbers, and characters.

The match configuration file defines matching logic for each field on which matching is performed. The OHMPI Match Engine provides several comparison functions that you can call in this file to fine-tune the match process. Comparison functions contain the logic to compare different types of data in very specific ways in order to arrive at a match weight for each field. These functions allow you to define how matching is performed for different data types and can be used in conjunction with either matching and unmatching probabilities or agreement and disagreement weight ranges for each field. This file also defines how to handle missing fields.

The following sections describe the format of the configuration file and provide an overview of the predefined comparison functions:

These sections describe the format of the files so you can modify them directly. You can also modify the match configuration file using the OHMPI Configuration Editor, which provides an easy, graphical way to configure matching rules.

OHMPI Match Engine Match Configuration File Format

The match configuration file is divided into two sections. The first section consists of one line that indicates the matching probability type. The second section consists of the matching rules to use for each match field. In a master person index application, this file can be modified from the Matching tab of the Master Person Index Configuration Editor. For more information, see the *Oracle Healthcare Master Person Index Configuration Guide.*

Match Configuration File Sample

Following is an excerpt from the default match configuration file. This excerpt illustrates the components that are described in the following sections.

```
ProbabilityType            1

FirstName              15  0   uf    0.99  0.001   10  -8
LastName               15  0   ul    0.99  0.001   10  -10
String                 25  0   ua    0.99  0.001   8   -8
DateDays               20  0   dD    0.99  0.001   10  -10 y 15      30
DateMonths             20  0   dM    0.99  0.001   10  -10 n
DateHours              20  0   dH    0.99  0.001   10  -10 y 30      60
DateMinutes            20  0   dm    0.99  0.001   10  -10 y 300 600
DateSeconds            20  0   ds    0.99  0.001   10  -10 y 75      60
Integer                15  0   nI    0.99  0.001   10  -10 n
Real                   15  0   nR    0.99  0.001   10  -10 n
Char                   1   0   c     0.99  0.001   5   -5
pro                    15  0   p     0.99  0.001   10  -10 20 5 5
```

Probability Type Section

The first line of the match configuration file defines the probability type to use for matching. Specify "0" (zero) to use m-probabilities and u-probabilities to determine a field's match weight; specify "1" (one) to use agreement and disagreement weight ranges. If the probability type is set to use agreement and disagreement weight ranges, the **m-prob** and **u-prob** columns in the matching rules section are ignored. Likewise, if the probability type is set to use m-probabilities and u-probabilities, the **agreement-weight** and **disagreement-weight** columns in the matching rules section are ignored. The default is to use agreement and disagreement weight ranges because they are more intuitive.

For more information about probabilities and weights, see Probabilities and Direct Weights.

Matching Rules Section

The section after the first line of the match configuration file contains match field rows, with each row defining how a certain data type or field will be matched. These are the

rules you specify in the match string you define for a master person index application. The syntax for this section is:

```
match-field size null-field function m-prob u-prob agreement disagreement
params data-sources
```

Table 2–1 describes each element in a match field row.

Table 2–1 Match Configuration File Columns

Column Number	Column Name	Description
1	match-field	A value that indicates to the Master Person Index Match Engine how each field should be weighted. Each field included in the match string (the MatchingConfig section of `mefa.xml`) must have a match type corresponding to a value in this column.
2	size	The number of characters in the field on which matching is performed, beginning with the first character. For example, to match on only the first four characters in a 10-digit field, the value of this column should be "4."
3	null-field	An index that specifies how to calculate the total weight for null fields or fields that only contain spaces. You can specify any of the following values: ■ **0** - (zero) If one or both fields are empty, the weight used for the field is 0 (zero). ■ **1** - (one) If both fields are empty, the agreement weight is used; if only one field is empty, the disagreement weight is used. ■ **2** - (two) This option bypasses the predefined logic for all cases when comparing fields: both fields empty, one field empty, or fields matching, or not matching. The user must make sure that any comparator configured with this option handles null or empty field to avoid having exceptions thrown. Note: This is applicable only for OHMPI patch set 2.0.11 and later. ■ **a#** - An "a" followed by a number specifies to use the agreement weight if both fields are empty. The agreement weight is divided by the number following the "a" to obtain the match weight for that field. If no number is specified, the default is "2." You can specify any number from 1 through 10. ■ **d#** - A "d" followed by a number specifies to use the disagreement weight if only one field is empty. The disagreement weight is divided by the number following the "d" to obtain the match weight for the field. If no number is specified, the default is "2." You can specify any number from 1 through 10. Note: In the above descriptions, the agreement and disagreement weights are either specified in the file or calculated using a logarithmic formula based on the m and u-probabilities (depending on the probability type).
4	function	The type of comparison to perform when weighting the field. For information about the available comparison functions, see Chapter 5, "OHMPI Match Engine Comparison Functions". An overview of the comparison functions is provided in Table 2–2.
5	m-prob	The initial probability that the specified field in two records will match if the records match. The probability is a double value between 0 and 1, and can have up to 16 decimal points.
6	u-prob	The initial probability that the specified field in two records will match if the records do not match. The probability is a double value between 0 and 1, and can have up to 16 decimal points.
7	agreement	The matching weight to be assigned to a field given that the fields match between two records. This number can be between 0 and 100 and can have up to 16 decimal points. It represents the maximum match weight for a field.
8	disagreement	The matching weight to be assigned to a field given that the fields do not match between two records. This number can be between 0 and -100 and can have up to 16 decimal points. It represents the minimum match weight for a field.
9	params	The parameters that correspond to the comparison function specified in column 4. Some comparison functions do not take any parameters and some take multiple parameters. For additional information about parameters, see Chapter 5, "OHMPI Match Engine Comparison Functions".
10	dataSources	The complete path to any data sources used by the comparison function specific in column 4. You can define as many data sources as there are data sources listed for the comparator in the comparators list file. The default comparators do not use data sources, but you can create a custom comparator that does.

OHMPI Match Engine Matching Comparison Functions at a Glance

Match field comparison functions, or **comparators**, compare the values of a field in two records to determine whether the fields match. The fields are then assigned a matching weight based on the results of the comparison function. You can use several different types of comparison functions in the match configuration file to define how the OHMPI Match Engine should match the fields in the match string. The OHMPI Match Engine provides several options to use with each function. You can also define custom comparison functions. For more information, see Chapter 6, "Creating Custom Comparators for the OHMPI Match Engine".

Table 2–2 summarizes each comparison function. A complete reference of the comparison functions and their parameters is included in Chapter 5, "OHMPI Match Engine Comparison Functions".

Note: The names of these comparison functions are configurable. The following table lists their default names.

Table 2–2 Comparison Function Summary

Comparison Function	Name	Description
b1	Bigram Comparator	Compares two strings using an algorithm based on the **Bigram** algorithm. This function compares two strings using all combinations of two consecutive characters and returns the total number of combinations that are the same.
b2	Advanced Bigram Comparator	Compares two strings allowing for character transpositions. This function is similar to the standard Bigram Comparator (b1).
u	Advanced Jaro String Comparator	Compares two strings taking into account uncertainty factors, such as string length, transpositions, and characters in common. This function is based on the **Jaro** algorithm.
ua	Winkler-Jaro String Comparator	Compares two strings similar to the Advanced Jaro String Comparator (u), but increases the agreement weight if the initial characters of each string are exact matches. This function takes into account key punch and visual memory errors. It is based on the **Jaro** algorithm with variants of Winkler / Lynch and McLaughlin.
uf	Advanced Jaro Adjusted for First Names	Based on the generic string comparator (u), this function is designed to specifically weight first name values. The string is analyzed and the weight adjusted based on statistical data.
ul	Advanced Jaro Adjusted for Last Names	Based on the generic string comparator (u), this function is designed to specifically weight last name values. The string is analyzed and the weight adjusted based on statistical data.
un	Advanced Jaro Adjusted for House Numbers	Based on the generic string comparator (u), this function is designed to specifically weight house number values. The string is analyzed and the weight adjusted based on statistical data.
us	Condensed String Comparator	Compares two strings similar to the Advanced Jaro String Comparator (u), but this function is a custom string comparator that compares two strings taking into account such uncertainty factors as string length, transpositions, key punch errors, and visual memory errors. Unlike the Advanced Jaro String Comparator, this function handles diacritical marks. This function also improves processing speed.
usu	Unicode String Comparator	Compares two strings similar to the Condensed String Comparator (us), but this function is based in Unicode to support multiple languages and alphabets. This comparator takes one parameter indicating the language to use.

Table 2–2 (Cont.) Comparison Function Summary

Comparison Function	Name	Description
usus	Unicode AlphaNumeric Comparator	Compares two strings similar to the Unicode String Comparator, but this function is designed to match on unique identifiers such as national IDs. This comparator takes one parameter indicating the language to use plus any of the following parameters: ■ Field length ■ Character types ■ Invalid values
ujs	Advanced Jaro AlphaNumeric Comparator	Compares two strings similar to the Advanced Jaro String Comparator, but this function is designed to match on unique identifiers such as national IDs. This comparator takes any of the following parameters: ■ Field length ■ Character types ■ Invalid values
c	Exact Character-to-Character Comparator	Compares string fields character by character. Each character must match in order for an agreement weight to be assigned.
nI	Integer Comparator	Compares integer fields using a relative distance value to determine the match weight. As the difference between the two fields increases, the match weight decreases. Once the difference is beyond the relative distance, a disagreement weight is assigned. This comparator takes two parameters; the first indicates whether to use a relative distance or direct string comparison and the second indicates the relative distance to use.
nR	Real Number Comparator	Compares fields containing real numbers using a relative distance value to determine the match weight. As the difference between the two fields increases, the match weight decreases. Once the difference is beyond the relative distance, a disagreement weight is assigned. This comparator takes two parameters; the first indicates whether to use a relative distance or direct string comparison, and the second indicates the relative distance to use.
nS	Condensed AlphaNumeric SSN Comparator	Compares social security numbers or other unique identifiers, taking into account any of these parameters: ■ Field length ■ Character types ■ Invalid values
dY	Date Comparator With Years as Units	Compares year values using relative distance values prior to and following the given year to determine the match weight. As the difference between the two fields increases, the match weight decreases. Once the difference is beyond the relative distance, a disagreement weight is assigned. The date comparison functions handle Gregorian years. This comparator takes up to three parameters; the first indicates whether to use a relative distance or direct string comparison, and the second and third indicate the relative distance before and after.
dM	Date Comparator With Months as Units	Compares the month and year using a relative distance as described above for the year comparison function (dY).
dD	Date Comparator With Days as Units	Compares the day, month, and year using a relative distance as described above for the year comparison function (dY).
dH	Date Comparator With Hours as Units	Compares the hour, day, month, and year using a relative distance as described above for the year comparison function (dY).
dm	Date Comparator With Minutes as Units	Compares the minute, hour, day, month, and year using a relative distance as described above for the year comparison function (dY).
ds	Date Comparator With Seconds as Units	Compares the second, minute, hour, day, month, and year using a relative distance as described above for the year comparison function (dY).
p	Prorated Comparator	Prorates the disagreement weight for a date or numeric field based on values you specify. Differences greater than the amount you specify receive the full disagreement weight. This comparator takes three parameters indicating the relative distance and the agreement and disagreement ranges.

Learning About the OHMPI Match Engine Comparator Definition List

The comparator definition list defines each comparator that is included in a master person index application. If a comparator is not included in this list, it cannot be used in the application. If you define a comparator in this list that is not provided with the OHMPI Match Engine, you need to define the logic of the new comparator in Java classes (for more information, see Chapter 6, "Creating Custom Comparators for the OHMPI Match Engine").

Below is an excerpt from the default comparators list file that defines two numeric comparators, Real Number Comparator and Integer Comparator. Both comparators take two parameters, and are dependent on a second comparator class named `CondensedStringComparator`.

```
<comparator description="Numerics comparator">
  <className>NumericsComparator</className>
  <codes>
    <code description="Real Number Comparator" name="n[R, ]"/>
    <code description="Integer Comparator" name="nI" />
  </codes>
  <params>
    <param description="distance/string comparison option"
           name="switch" type="java.lang.String"/>
    <param description="Spectrum of comparison"
           name="range" type="java.lang.Integer|java.lang.Double"/>
  </params>
  <data-sources/>
  <dependency-classes>
    <dependency-class matchfield="CSC"
     name="com.sun.mdm.matcher.comparators.base.CondensedStringComparator"/>
  </dependency-classes>
  <curve-adjust status="false"/>
</comparator>
```

The comparators are defined in XML format. Table 2–3 lists and describes each element in the XML file.

Table 2–3 Comparator Definition List Elements

Element	Attribute	Description
group	-	An element that contains a list of comparators that all share the same Java package.
group	description	A brief description of the comparator group.
group	path	The Java package that contains the code that defines the comparators in the group.
comparator	-	A definition for one subgroup of comparators that are all based on the same Java class, have the same Java class dependencies, accept the same parameters and data sources, and have the same curve adjustment setting.
comparator	description	A brief description of the comparator subgroup.
className	-	The name of the class that defines the logic for the comparators. The class must be contained in the package specified for the group element, as described above.
codes	-	A container element for a list of the comparators in the subgroup, with descriptions and processing codes of each comparator.
code	-	A description and processing code for one comparator.
code	description	A description of the comparator. The value you specify here appears in the comparator drop-down list on the Master Person Index Configuration Editor.
code	name	A unique identifying name for the comparator. These are the comparator names used in the rules definitions in the match configuration file (`matchConfigFile.cfg`).
params	-	A container element for a list of static parameters for the subgroup of comparators. Parameters are optional.

Table 2–3 (Cont.) Comparator Definition List Elements

Element	Attribute	Description
param	-	One parameter definition for the comparators.
param	description	A brief description of the parameter.
param	name	A short name for the parameter.
param	type	The Java data type of the values that can be specified for the parameter.
data-sources	-	A container element for a list of data files that contain additional information for the subgroup of comparators. For example, a comparator that generates weights based on the distance between postal codes might use lookup files containing information about the zip codes. Data sources are optional.
data-source	-	A definition for one data source. Currently, only file data sources are supported.
data-source	description	A brief description of the data source.
data-source	name	The complete path and filename of the data source.
data-source	type	The type of data source being used. Currently, the only value you can specify is "java.io.File".
dependency-classes	-	A container element that defines a list of Java classes on which the comparator class is dependent. The current comparator class inherits from the comparator classes you specify here as well as all the match fields (defined in `matchConfigFile.cfg`) that use that comparator.
dependency-class	-	A definition for one comparator class, called a **dependency comparator**, on which the current comparator class is dependent.
dependency-class	matchField	The name of the dependency comparator's match field.
dependency-class	name	The name of the dependency comparator class.
curve-adjust	-	An indicator of whether to apply special adjustments to the weighting curve. The curve adjustment is defined for each comparator individually in a Java class named *comparator_name*`CurveAdjustor`.
curve-adjust	status	The status of the curve adjustor. Specify **true** to use the curve adjustor; specify **false** to disable the curve adjustor.

3

Match Engine Configuration for Common Data

This chapter provides conceptual information on how the OHMPI Match Engine can match on any type of data. Common data types for matching include person names, addresses, and business names. It also provides information on configuring the match engine for matching on these data types in a master person index application, fine-tuning weights and measures, and customizing match configuration and thresholds.

This chapter includes the following sections:

Learning About the OHMPI Match String and Match Types

This section provides information about the OHMPI match string, match string fields, and match types.

The OHMPI Match String

The data string that is passed to the OHMPI Match Engine for match processing is called the **match string**. For a master person index application, the match string is defined in the MatchingConfig section of `mefa.xml`. The match and standardization engine configuration files, the blocking query, and the matching configuration are closely linked in the search and matching processes. The blocking query defines the select statements for creating the candidate selection pool during the matching process. The matching configuration defines the match string that is passed to the match engine from the records in the candidate selection pool. Finally, the OHMPI Match Engine configuration files define how the match string is processed.

The OHMPI Match Engine configuration files are dependent upon the match string, and it is very important when you modify the match string to ensure that the match type you specify corresponds to the correct row in the match configuration file (`matchConfigFile.cfg`). For example, if you are using person matching and add "MaritalStatus" as a match field, you need to specify a match type for the MaritalStatus field that is listed in the first column of the match configuration file. You must also make sure that the matching logic defined in the corresponding row of the match configuration file is defined appropriately for matching on the MaritalStatus field. For more information about match types, see OHMPI Match Engine Match Types.

OHMPI Match Engine Match String Fields

In a master person index application, the match string processed by the OHMPI Match Engine is defined by the match fields specified in mefa.xml, and the logic for how the fields are matched is defined in the match configuration file (`matchConfigFile.cfg`). The match engine can process any combination of fields you specify for matching using the predefined comparators or any new comparators you define. Not all fields in a record need to be processed by the OHMPI Match Engine. Before you define the match string, analyze your data to determine the fields that are most likely to indicate a match or non-match between two records.

The following sections provide additional information about the match string for different data types:

- Person Data Match String Fields on page 2
- Address Data Match String Fields on page 2
- Business Name Match String Fields on page 2

Person Data Match String Fields

By default, the match configuration file (`matchConfigFile.cfg`) includes rows specifically for matching on first name, last name, social security numbers, and dates (such as a date of birth). It also includes a row for matching a single character with logic specialized for a gender field. You can use any of the existing rows for matching or you can add rows for the fields you want to match. When matching on person names, determine whether you want to use the original field values, the normalized field values, or the phonetic values. The match engine can handle any of these types of fields, but the best comparator for each type might be different. Also determine how much weight you want to give each field type and configure the match configuration file accordingly.

Address Data Match String Fields

By default, the match configuration file (`matchConfigFile.cfg`) includes rows specifically for matching on the fields that are parsed from the street address fields, such as the street number, street direction, and so on. The file also defines several generic match types you can configure for address fields. You can use any of the existing rows for matching or you can add rows for the fields you want to match. If you specify an "Address" match type for any field in the Master Person Index Wizard, the default fields that store the parsed data are automatically added to the match string in `mefa.xml`. These fields include the house number, street direction, street type, and street name. You can remove any of these fields from the match string.

When matching on address fields, determine whether you want to use the original field values, the standardized field values, or the phonetic values. The match engine can handle any of these types of fields, but the best comparator for each type might be different. Also determine how much weight you want to give each field type and configure the match configuration file accordingly.

Business Name Match String Fields

By default, the match configuration file (`matchConfigFile.cfg`) includes rows specifically for matching on the fields that are parsed from the business name fields. The file also defines several generic match types you can customize to use with business name fields. You can use any of the existing rows for matching or you can add rows for the fields you want to match. If you specify a "BusinessName" match type for any field in the wizard, most of the parsed business name fields are automatically added to the match string in mefa.xml, including the name, organization

type, association type, sector, industry, and URL. You can remove any of these fields from the match string.

When matching on business name fields, determine whether you want to use the original field values, the standardized field values, or the phonetic values. The match engine can handle any of these types of fields, but the best comparator for each type might be different. Also determine how much weight you want to give each field type and configure the match configuration file accordingly.

OHMPI Match Engine Match Types

The default match configuration file, `matchConfigFile.cfg`, defines several rules that you can customize for the type of data being processed. Each rule is identified by a **match type** in the first column of each row. This value identifies the type of matching to perform to the match engine. In a master person index application, the match type is entered for each field in the match string section of `mefa.xml`.

The match configuration OHMPI Match Engine's `matchConfigFile.cfg` appears under the Match Engine node of the master person index project. For more information about the comparison functions used for each match type and how the weights are tuned, see Customizing the Match Configuration and Chapter 5, "OHMPI Match Engine Comparison Functions".

The following tables list the match types that are typically used in processing different data types, including:

- Table 3–1, " Person Data Match Types"
- Table 3–2, " Address Match Types"
- Table 3–3, " Business Name Match Types"
- Table 3–4, " Miscellaneous Match Types"

Table 3–1 lists the match types that are designed for matching on person data.

Table 3–1 Person Data Match Types

This indicator ...	processes this data type ...
FirstName	A first name field, including middle name, alias first name, and alias middle name fields.
LastName	A last name field, including alias last name fields.
SSN	A field containing a social security number.
Gender	A field containing a gender code.

Table 3–2 lists the match types that are designed for matching on address data.

Table 3–2 Address Match Types

This indicator ...	processes this data type ...
StreetName	The parsed street name field of a street address.
HouseNumber	The parsed house number field of a street address.
StreetDir	The parsed street direction field of a street address.
StreetType	The parsed street type field of a street address.

Table 3–3 lists the match types that are designed for matching on business names.

Table 3–3 Business Name Match Types

This match type ...	processes this data type ...
PrimaryName	The parsed name field of a business name.
OrgTypeKeyword	The parsed organization type field of a business name.
AssocTypeKeyword	The parsed association type field of a business name.
LocationTypeKeyword	The parsed location type field of a business name.
AliasList	The parsed alias type field of a business name.
IndustrySectorList	The parsed industry sector field of a business name.
IndustryTypeKeyword	The parsed industry type field of a business name.
Url	The parsed URL field of a business name.

Miscellaneous match types provide additional logic for matching on a variety of data types, such as date, numeric, string, and character fields.

Table 3–4 Miscellaneous Match Types

This indicator ...	processes this data type ...
Date	The year of a date field.
DateDays	The day, month, and year of a date field.
DateMonths	The month and year of a date field.
DateHours	The hour, day, month, and year of a date field.
DateMinutes	The minute, hour, day, month, and year of a date field.
DateSeconds	The seconds, minute, hour, day, month, and year of a date field.
String	A generic string field.
Unistring	A generic Unicode string field.
Integer	A field containing integers.
Real	A field containing real numbers.
Char	A field containing a single character.
pro	Any field on which you want the OHMPI Match Engine to use prorated weights.
Exac	Any field you want the OHMPI Match Engine to match character for character.
CSC	A generic string.
DOB	A date of birth in string rather than date format.

Configuring the Match String for a Master Person Index Application

The **MatchingConfig** section of `mefa.xml` determines which fields are passed to the OHMPI Match Engine for matching (the match string). The match types specified in this section help the match engine determine the algorithm and custom logic to use for matching on each field.

If you are matching on fields parsed from a free-form text field, define each individual parsed field you want to use for matching in the Master Person Index Wizard or Configuration Editor. The match types you can use for each field in this section are defined in the first column of the match configuration file (`matchConfigFile.cfg`). Make sure the match type you specify has the correct matching logic defined in the match configuration file. See OHMPI Match Engine Match Types for more information.

The following topics provide more information about matching on different types of data:

Configuring the Match String for Person Data

When matching on person data, you can include any field stored in the database for matching. To configure the match string, see the *Oracle Healthcare Master Person Index Configuration Guide*. For the OHMPI Match Engine, each data type has a different match type (specified by the match-type element in the matching configuration file). The FirstName, LastName, SSN, Gender, and DOB match types are specific to person matching. You can specify any of the other match types defined in the match configuration file as well. For more information, see OHMPI Match Engine Match Types.

A sample match string for person matching is shown below. This sample matches on first and last names, date of birth, social security number, gender, and the street name of the address.

```
<match-system-object>
   <object-name>Person</object-name>
   <match-columns>
      <match-column>
         <column-name>
            Enterprise.SystemSBR.Person.FirstName_Std
         </column-name>
         <match-type>FirstName</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Person.LastName_Std
         </column-name>
         <match-type>LastName</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Person.SSN
         </column-name>
         <match-type>SSN</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Person.DOB
         </column-name>
         <match-type>DateDays</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Person.Gender
         </column-name>
         <match-type>Char</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Person.Address.StreetName
         </column-name>
         <match-type>StreetName</match-type>
      </match-column>
   </match-columns>
</match-system-object>
```

Configuring the Match String for Address Data

For matching on street address fields, make sure the match string you specify in the MatchingConfig section of mefa.xml contains all or a subset of the fields that contain the standardized data (the original text in street address fields is generally too inconsistent to use for matching). You can include additional fields for matching, such as the city name or postal code.

To configure the match string, see the *Oracle Healthcare Master Person Index Configuration Guide*. For the OHMPI Match Engine, each component of a street address has a different match type (specified by the **match-type** element in the matching configuration file). The default match types for addresses are StreetName, HouseNumber, StreetDir, and StreetType. You can specify any of the other match types defined in the match configuration file, as well. For more information, see OHMPI Match Engine Match Types.

A sample match string for address matching is shown below.

```
<match-system-object>
   <object-name>Person</object-name>
   <match-columns>
      <match-column>
         <column-name>Enterprise.SystemSBR.Person.Address.StreetName
         </column-name>
         <match-type>StreetName</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Person.Address.HouseNumber
         </column-name>
         <match-type>HouseNumber</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Person.Address.StreetDir
         </column-name>
         <match-type>StreetDir</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Person.Address.StreetType
         </column-name>
         <match-type>StreetType</match-type>
   </match-column>
   </match-columns>
</match-system-object>
```

Configuring the Match String for Business Names

For matching on business name fields, make sure the match string you specify in the MatchingConfig section of `mefa.xml` contains all or a subset of the fields that contain the standardized data (the unparsed business names are typically too inconsistent for matching). You can include additional fields for matching if required.

To configure the match string, see the *Oracle Healthcare Master Person Index Configuration Guide*. For the OHMPI Match Engine, each data type has a different match type (specified by the match-type element of the matching configuration file). The PrimaryName, OrgTypeKeyword, AssocTypeKeyword, IndustrySectorList, IndustryTypeKeyword, and Url match types are specific to business name matching. You can specify any of the other match types defined in the match configuration file, as well. For more information, see OHMPI Match Engine Match Types.

A sample match string for business name matching is shown below. This sample matches on the company name, the organization type, and the sector.

```
<match-system-object>
   <object-name>Company/object-name>
   <match-columns>
      <match-column>
         <column-name>Enterprise.SystemSBR.Company.Name_PrimaryName
         </column-name>
         <match-type>PrimaryName</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Company.Name_OrgType
         </column-name>
         <match-type>OrgTypeKeyword</match-type>
      </match-column>
      <match-column>
         <column-name>Enterprise.SystemSBR.Company.Name_Sector
         </column-name>
         <match-type>IndustryTypeKeyword</match-type>
      </match-column>
   </match-columns>
</match-system-object>
```

Fine-Tuning Weights and Thresholds for Oracle Healthcare Master Person Index

Each Oracle Healthcare Master Person Index implementation is unique, typically requiring extensive data analysis to determine how to best configure the structure and matching logic of the master person index application. The following topics provide an overview of the process of fine-tuning the matching logic in the match configuration file and fine-tuning the match and duplicate thresholds.

- Data Analysis Overview on page 7
- Customizing the Match Configuration and Thresholds on page 8

Data Analysis Overview

A thorough analysis of the data to be shared with the master person index application is a must before beginning any implementation. This analysis not only defines the types of data to include in the object structure, but indicates the relative reliability of each system's data, helps determine which fields to use for matching, and indicates the relative reliability of each match field.

To begin the analysis, the legacy data that will be converted into the master person index database is extracted and analyzed. Once the initial analysis is complete, you can perform an iterative process to fine-tune the matching and duplicate thresholds and to determine the level of potential duplication in the existing data. If you plan to use the Data Profiler and Bulk Matcher tools generated by Oracle Healthcare Master Person Index to analyze data, see *Oracle Healthcare Master Person Index Analyzing and Cleansing Data User's Guide* and *Oracle Healthcare Master Person Index Loading the Initial Data Set User's Guide* before you extract the legacy data.

Customizing the Match Configuration and Thresholds

There are three primary steps to customizing how records are matched in a master person index application.

Determining the Match Fields

Before extracting data for analysis, review the types of data stored in the messages generated by each system. Use these messages to determine which fields and objects to include in the object structure of the master person index application. From this object structure, select the fields to use for matching. When selecting these fields, keep in mind how representative each field is of a specific object. For example, in a master person index, the social security number field, first and last name fields, and birth date are good representations whereas marital status, suffix, and title are not. Certain address information or a home telephone number might also be considered. In a master company index, the match fields might include any of the fields parsed from the complete company name field, as well as a tax ID number or address and telephone information.

Customizing the Match Configuration

Once you determine the fields to use for matching, determine how the weights will be generated for each field. The primary tasks include determining whether to use probabilities or agreement weight ranges and then choosing the best comparison functions to use for each match field.

Probabilities or Agreement Weights

The first step in configuring the match configuration is to decide whether to use m-probabilities and u-probabilities or agreement and disagreement weight ranges. Both methods will give you similar results, but agreement and disagreement weight ranges allow you to specify the precise maximum and minimum weights that can be applied to each match field, giving you control over the value of the highest and lowest matching weights that can be assigned to each record.

Defining Relative Value

For each field used for matching, define either the m-probabilities and u-probabilities or the agreement and disagreement weight ranges in the match configuration file. Review the information provided under OHMPI Match Engine Matching Weight Formulation to help determine how to configure these values. Remember that a higher m-probability or agreement weight gives the field a higher weight when field values agree.

Determining the Weight Range

In order to find the initial values to set for the match and duplicate thresholds, you must determine the total range of matching weights that can be assigned to a record. This weight is the sum of all weights assigned to each match field. Using the data analysis tool provided can help you determine the match and duplicate thresholds.

Weight Ranges Using Agreement Weights

For agreement and disagreement weight ranges, determining the match weight ranges is very straightforward. Simply total the maximum agreement weights for each field to determine the maximum match weight. Then total the minimum disagreement weights for each match field to determine the minimum match weight. The following table provides a sample agreement / disagreement configuration for matching on person data. As you can see, the range of match weights generated for a master person index application with this configuration is from -36 to +38.

Table 3–5 Sample Agreement and Disagreement Weight Ranges

Field Name	Maximum Agreement Weight	Minimum Disagreement Weight
First Name	8	-8
Last Name	8	-8
Date of Birth	7	-5
Gender	5	-5
SSN	10	-10
Maximum Match Weight	38	-
Minimum Match Weight	-	-36

Weight Ranges Using Probabilities

Determining the match weight ranges when using m-probabilities and u-probabilities is a little more complicated than using agreement and disagreement weights. To determine the maximum weight that will be generated for each field, use the following formula:

```
LOG2(m_prob/u_prob)
```

To determine the minimum match weight that will be generated for each field, use the following formula:

```
LOG2((1-m_prob)/(1-u_prob))
```

The following table illustrates m-probabilities and u-probabilities, including the corresponding agreement and disagreement weights that are generated with each combination of probabilities. As you can see, the range of match weights generated for a master person index application with this configuration is from -35.93 to +38

Table 3–6 Sample m-probabilities and u-probabilities

Field Name	m-probability	u-probability	Max Agreement Weight	Min Disagreement Weight
First Name	.996	.004	7.96	-7.96
Last Name	.996	.004	7.96	-7.96
Date of Birth	.97	.007	7.11	-5.04
Gender	.97	.03	5.01	-5.01
SSN	.999	.001	9.96	-9.96
Maximum Match Weight	-	-	38	-
Minimum Match Weight	-	-	-	-35.93

Comparison Functions

The match configuration file defines several match types for different types of fields. You can either modify existing rows in this file or create new rows that define custom matching logic. To determine which comparison functions to use, review the information provided in Chapter 5, "OHMPI Match Engine Comparison Functions". Choose the comparison functions that best suit how you want the match fields to be processed.

Determining the Weight Thresholds

Weight thresholds tell the master person index application how to process incoming records based on the matching probability weights generated by the OHMPI Match Engine. Two parameters in master.xml provide the master person index application with the information needed to determine if records should be flagged as potential duplicates, if records should be automatically matched, or if a record is not a potential match to any existing records.

- **Match Threshold** - Specifies the weight at which two profiles are assumed to represent the same person and are automatically matched (this depends on the setting for the OneExactMatch parameter).
- **Duplicate Threshold** - Specifies the minimum weight at which two profiles are considered potential duplicates of one another. The matching threshold indicates the maximum weight for potential duplicates.

Specifying the Weight Thresholds

There are many techniques for determining the initial settings for the match and duplicate thresholds. This section discusses two methods. You can also use the Data Profiler and Bulk Matcher to determine these thresholds. For more information, see *Oracle Healthcare Master Person Index Analyzing and Cleansing Data User's Guide* and *Oracle Healthcare Master Person Index Loading the Initial Data Set User's Guide*.

The first method, the weight distribution method, is based on the calculation of the error rates of false matches and false non-matches from analyzing the distribution spectrum of all the weighted pairs. This is the standard method. The second method, the percentage method relies on measuring the total maximum and minimum weights of all the matched fields and then specifying a certain percentage of these values as the initial thresholds.

The weight distribution method is more thorough and powerful but requires analyzing a large amount of data (match weights) to be statistically reliable. It does not apply well in cases where one candidate record is matched against very few reference records. The percentage method, though simple, is very reliable and precise when dealing with such situations. For both methods, defining the match threshold and the duplicate threshold is an iterative process.

Weight Distribution Method

Each record pair in the master person index application can be classified into three categories: matches, non-matches, and potential matches. Your goal is to make sure that very few records fall into the False Matches region (if any), and that as few as possible fall into the False Non-matches region. You can see how modifying the thresholds changes this distribution. Balance this against the number of records falling within the Manual Review section, as these will each need to be reviewed, researched, and resolved individually.

Percentage Method

Using this method, you set the initial thresholds as a percentage of the maximum and minimum weights. Using the information provided under Weight Ranges Using Agreement Weights or Weight Ranges Using Probabilities, determine the maximum and minimum values that can be generated for composite match weights. For the initial run, the match threshold is set intentionally high to catch only the most probable matches. The duplicate threshold is set intentionally low to catch a large set of possible matches.

Set the match threshold at 70% of the maximum composite weight starting from zero as the neutral value. Using the weight range samples in Table 17, this would be 70% of 38, or 26.6. Set the duplicate threshold near the neutral value (that is, the value in the center of the maximum and minimum weight range). The value could be set between 10% of the maximum weight and 10% of the minimum weight. Using the samples above, this would be between 3.8 (10% of 38) and -3.6 (10% of -36).

Fine-tuning the Thresholds

Achieving the correct thresholds for your implementation is an iterative process. First, using the initial thresholds described earlier, process the data extracts into the master person index database. Then analyze the resulting assumed match and potential duplicates, paying close attention to the assumed match records with matching weights close to the match threshold, to potential duplicate records close to either threshold, and to non-matches near the duplicate threshold.

If you find that most or all of the assumed matches at the low end of the match range are not actually duplicate records, raise the match threshold accordingly. If, on the other hand, you find several potential duplicates at the high end of the duplicate range that are actual matches, decrease the match threshold accordingly. If you find that most or all of the potential duplicate records in the low end of the duplicate range should not be considered duplicate matches, consider raising the duplicate threshold. Conversely, if you find several non-matches with weight near the duplicate threshold that should be considered potential duplicates, lower the duplicate threshold.

Repeat the process of loading and analyzing data and adjusting the thresholds until you are satisfied with the results.

4

Setting Match Field Variations and Agreement/Disagreement

The OHMPI Match Engine compares records using probabilistic matching to determine the proximity of the records. This comparison of records is configured at design time specific to a custom application, and is based on a set of fields and their agreement and disagreement weights.

OHMPI 2.0 introduces a host of innovative matching options, collectively referred to as power match options, to the OHMPI Match Engine, including the capability to create variations in sets of match fields and agreement / disagreement weights. The bottom-line benefit is availability of sophisticated match options that results in higher levels of granular matching and higher levels of accuracy in match results. Value substitution can be used to reduce data entry errors that cause last name and first name swapping in records. Ability to configure multiple MatchSets based on various conditions enables customers to configure powerful OHMPI solutions catering to customers' requirements for varying levels of data fields and sources of data.

This chapter includes the following sections:

Introducing the New Types of Matching Available in OHMPI

OHMPI provides a pluggable interface MatcherAPI, which is a matching layer that is built on top of the Match Engine.

> **Note:** These features will work only if you have not modified default MatcherAPI implementation - `SbmeMatcherAdapter`, that is specified in the <matcher-api> tag of the `mefa.xml` file.

Any previous project will work seamlessly with this release.

System-dependent Matching

It is possible to associate a different set of match columns with different systems. For example, if System A has reliable SSN data but System B does not have reliable SSN data, execute matching with a different set of match columns that do not have SSN when the data comes from System B.

System-dependent matching matches different configurations for distinct system sources; that is, two different system sources with the same match field type (for example, FirstName) will have access to different matching configuration information. This means that the FirstName/LastName comparators for System A will have different agreement /disagreement weights when compared with those for System B.

Conditional Matching

Conditional criteria with a set of match columns is capable of determining which set of columns will be used for matching. This is especially useful when the nature of a person-identity parameter defines the match columns that are to be used. For example, if an incoming *Person ID* is a drivers license, "driverLicense" would be used as the match field; however, if an incoming *Person ID* is a medical record, "medicalID" would be used as the match field, and so on.

Frequency-based Matching

Frequency-based algorithms at the field-level provide frequency-based matching that results in frequency-based weights. This means that matches on less-frequent data must obtain a higher match weight than more frequent data (that is, names with higher frequencies must evaluate to lower match scores). This feature is important when we deal with an odd distribution of data. For example, if the field is a person's first name, then we may have to deal with data with very high frequency (for example, "John") and very low frequency (for example, "Alrik") in a place like California. This feature helps readjust the associated weights for the first name so that occurrences of "John" have less impact than "Alrik" in the overall weight. The customer and/or implementer must be able to set a step-down of the associated weight (as a percentage) and also manage the lists of field values to be included in a frequency based deployment. As this feature is pluggable, you can replace the default weight adjustments with plug-in classes.

Note: Frequency-based matching involves calculating frequencies with representative large data that might use one of the profiler functions.

Alias Matching and Field Swapping

Swapping or substituting names during matching, such as first and second names (for example, *Karl Robert* Els could also be known as *Robert* Els) so that it is possible to be match on *Karl* Els and *Robert* Els)

Swapping of first and last names (for example, *James Dean*) sometimes needs to be done as the first and last names have been wrongly entered into computer systems (for example, FirstName=Dean, LastName= James).

Matching of a first name must be done with occurrences in aliases as well. This means that OHMPI matches FirstName=First Name (for example, *James*) and FirstName=Alias (for example, *Jim*).

Cap for Agreement Matching

There is a maximum cap on agreement weights for a certain group of match fields. For example, if the match fields are DOB, SSN, FirstName, [address, Phone#], the group [address, Phone#] fields are given individual maximum agreement scores of 10 and 10. However, since the maximum cap for this group [address, Phone#] is also set to 10, if any one of these fields matches, the group is considered a good match candidate. In a cap group, the fields are associated similar to an 'OR' operator, and they are typically used to reduce false positives. For example, we can get false positives because a street name [address] and a home phone number [Phone#] would both match for someone else living at the same address. Since it is probable that this person would also have the same last name, we would be generating a weight that is too high for this particular situation. Only one of the weights [street name or home phone number] should be used, or the weight should be capped as the higher of the two values. This is a common problem when there are multiples of related fields used for matching. The matching process assumes that each match field is independent of the other and can break down when they are not, such as using a home phone number and street name for matching purposes.

Waterfall Matching

Waterfall Matching includes MatchSet, Conditional Matching, and System-based Matching.

Understanding MatchSet, Conditional Matching, System-based Matching, and Waterfall Matching

MatchSet defines a set of match fields along with optional agreement and disagreement weights. It also contains information about conditional matching based on systems and/or individual fields.

There are two kinds or pools of MatchSets

- **Conditional MatchSets**: These MatchSets are associated with conditions that are based on a system associated with input data or conditions that are based on record field values. A given conditional MatchSet qualifies given input data only if its conditions satisfy input data.
- **Unconditional MatchSets**: These MatchSets do not have any condition associated with them.

The matching system processes all conditional MatchSets first and selects the MatchSets whose conditions satisfy the input data. Afterwards the system proceeds to do actual matching of input data with the selected set of MatchSets. Each "MatchSet" matching evaluation with input data, will yield a score that is based on probabilistic matching of input data with match fields and agreement and disagreement fields specified in that MatchSet.

The `mefa-ext.xml` file supports a tag called **ProcessUnconditionalMatchSets** which has two values: **true** and **false**.

- **ProcessUnconditionalMatchSets - true**: After processing conditional match sets, the system proceeds to all unconditional match sets. The overall match score is the maximum of the scores evaluated from all these match sets.

- **ProcessUnconditionalMatchSets - false**: After processing conditional MatchSets, the system processes an unconditional MatchSets pool only if none of MatchSets in the conditional MatchSets pool satisfy the input data condition. In other words, if one or more Conditional MatchSets satisfy input data condition, then the unconditional MatchSets are not processed.

Table 4–1 MatchSet, Conditional Matching, and System-based Matching Examples

MatchSet name	MatchSet ID	Description	Input 1 SO with System A	Input 2 SO with System B	Input 3 SO with System C	Input 4 SO with System E
MS – System A or B + Gender = M	1	System A or B and Gender condition	X	X	-	-
MS – System A + ColorOfHair = W	2	System A and ColorOfHair condition	X	-	-	-
MS – System C	3	System C no field condition	-	-	X	-
MS – System C or D	4	System C or D no field level condition	-	-	X	-
MS Default (added by system based on `mefa.xml`)	5	Default contains no system or field level conditions	-	-	-	X
MatchSet Gender =F	6	No system with condition	-	-	-	X
MatchSet CoH = black	7	No system with condition	-	-	-	X
MatchSet condition = some other	8	No system with condition	-	-	-	X

In Table 4–1, above, X represents the matching will be performed based on weights defined in corresponding MatchSet in column 1. For example, if the system object comes in from System A, the matching engine will go through MS ID 1 and 2 and do the waterfall logic to get the maximum weight based on two sets of comparison. Similarly, if input comes from system B the only MatchSet that satisfies the system match is "MS – System A or B" ID 1 which is MS – System A or B so the matching is done based on configuration in matchset 1 and all other matchsets will be ignored including default and 6, 7, and 8. Thus there is no waterfall matching invoked.

However if input with system E comes in, notice there is no matchset defined for system E, the extension matching will go through all matchsets that have no system attached. In this case the matching will be performed based on 5,6,7,8 matchset and waterfall logic will kick in to retrieve the maximum weight out of 4 results.

Using the Design-time Configuration

The Design-time configurations are stored in an XML file called `match-ext.xml`. The `match-ext.xml` file has three main sections:

- `<matchSet>` allows for multiples of MatchSet and properties of matchSet to support conditional and System-dependent Matching
- `<frequencyBasedFields>` defines fields whose weights are adjusted based upon the frequency of certain strings
- `<fieldsSubstitution>` contains a multiple set of fields whose data is swapped for matching & blocking

The first five sections below provide information you need to be aware of, as well as an example of the XML file you will edit to set up your MatchSets. The sixth section provides a simplistic procedure on where and how to set up the `match-ext.xml` file for the matching you require.

Understanding the XML Elements

The Design-time configurations are stored in an XML file called `match-ext.xml`. The `match-ext.xml` file has three main elements:

- `<matchSet>` allows for multiples of MatchSet and properties of matchSet
- `<frequencyBasedFields>` defines fields whose weights are adjusted based upon the frequency of certain strings
- `<fieldsSubstitution>` contains a multiple set of fields whose data is swapped for matching & blocking

matchSet

The MatchSet is set of matching fields and also consists of their agreement and disagreement weights. The core enhancement in the features set is using some of Match fields with their agreement / disagreement weights to match pairs of data, which results in different match score. The highest match score from all sets of match sets is taken to determine if there is an assumed match, duplicate match, or non-match.

> **Note:** The match threshold and the duplicate thresholds are fixed irrespective of the type of MatchSet that is used.

Each MatchSet can be associated with:

- A set of systems
- A set of conditions based on [field = value]; each such condition has an AND operator between themselves
- **scoreMultiplier**

  When there are many MatchSets to match against, some MatchSets can be more reliable than others to find assumed or potential thresholds. This factor is multiplied with the score computed by MatchEngine to normalize it with respect to the application Match threshold and Duplicate threshold. So if scoreMultiplier is .8, and total score computed for a MatchSet is 50, then it is normalized to 40.
- **child MatchSet**

  The MatchSet can contain a child MatchSet. The child MatchSet can be used for capping agreement / disagreement weights for group of match fields. This represents the 'OR' functionality within a match set.

frequencyBasedFields

FrequencyBasedFields consists of:

- **set of fields** whose max agreement weights will be normalized to a lower weight depending upon known frequency of field data in the system.
- **maxPercentWeightVariation**

  maxPercentWeightVariation is the percentage of maximum variation from the agreement weight. If it is 50%, and the agreement weight for a field is 20, then regardless of how the agreement weight was reduced via frequency based rules, the max agreementWeight reduction would be 20*.5 = 10.

  Note: This is not same score computed by the Match Engine, which depends upon a match comparison of two field values.

- **Computation of weight reduction**

  A plugin interface is provided that can override default rules on how much agreement weight should be reduced with the high frequency of a word.

  Note: Based upon the frequency of a field value, it only affects the maximum agreement weight. The disagreement weight is not affected. For example, If a name has an agreement weight of +10, the disagreement weight = -10. However, when high frequency words such as "John" and "John" are matched, the total agreement weight is lowered to perhaps +5. If one name is "George" and other name is "John" they do not match, but their disagreement score would still be -10.

Every name in FrequencyBasedFields has its absolute frequency recorded in a frequency table (see Table 4–2). You populate this table off-line using a custom process that is outside the scope of OHMPI.

Table 4–2 SBYN Frequency Table

Value	Frequency
John	10000
Evans	3000
Chu	200

FrequencyTable map in memory: During startup time of an OHMPI server (for example GlassFish or WebLogic), the frequency table is loaded in memory and their frequency percentages for each name are computed. If a name is not in memory it has low frequency and is given maximum agreement weight.

fieldsSubstitution

fieldsSubstitution consists of:

- **Value Substitution**

  The feature provides functionality to support swapping of the fields during the matching process. The swapping functionality helps to resolve cases in which certain field values such as Last Name is erroneously entered instead of First Name. Field substitution will allow the match to be performed based on swapped field values in addition to the normal matching process based on MatchSet.

Examples would be:

```
<fieldSubstitution>
         <matching>
            <substituteField>
               <targetField>Enterprise.SystemSBR.Person.LastName</targetField>
               <sourceField>Enterprise.SystemSBR.Person.FirstName</sourceField>
              </substituteField>
         </matching>
         <blocking>
            <substituteField>
               <targetField>Enterprise.SystemSBR.Person.LastName</targetField>
              <sourceField>Person.FirstName</sourceField>

            </substituteField>

         </blocking>
      </fieldSubstitution>
```

The absence of the <*fieldSubstitution*> element from the `match-ext.xml` file is treated as if the Value Substitution feature has been turned off.

Description of Various Elements:

- **matching** defines the various elements needed to configure the matching part of value substitution
- **targetField** defines the target field whose value will be substituted by one or more sourceField
- **sourceField** defines the source field that will contain input data and also be queried to get the substitute value

- **Alias Matching**

  The feature enhances existing matching by enabling matching done based on previously used names or aliases.

  It is a function to match a First Name against FirstName and Aliases. The value substitution construct described above could be used to achieve this functionality. The target field would describe the actual field that needs to be matched against source fields such as Aliases and so on.

Sample XML File

Below is an example of the `match-ext.xml` file. See XML File Explanation for details.

```
<MatchExtConfiguration>
      <processUnconditionalMatchSets>false</processUnconditionalMatchSets>
      <matchSet ID="1">
          <scoreMultiplier>.9</scoreMultiplier>
    <matchColumns>
     <matchColumn>
      <columnName>Enterprise.SystemSBR.Person.FirstName_Std</columnName>
      <matchType>FirstName</matchType>
      <agreementWeight>10</agreementWeight>
      <disagreementWeight>-10</disagreementWeight>
     </matchColumn>
    <matchColumn>
      <columnName>Enterprise.SystemSBR.Person.LastName_Std</columnName>
      <matchType>LastName</matchType>
    </matchColumn>
```

```
    </matchColumns>
      <conditions>
       <fieldCondition>
        <field>Enterprise.SystemSBR.Person.ID</field>
        <value>DL</value>
       </fieldCondition>
       <systems>
        <system>HospitalA</System>
        <system>HospitalB</System>
       </systems>
      </conditions>
      <childMatchSet capAgreement="10" capDisagreement="-10">
      <matchColumns>
      <matchColumn>
       <columnName>Enterprise.SystemSBR.Person.Phone.PhoneNum</columnName>
       <columnName>Enterprise.SystemSBR.Person.Phone.PhoneNum</columnName>
       <matchType>Phone</matchType>
        <agreementWeight>10</agreementWeight>
        <disagreementWeight>-10</disagreementWeight>
      </matchColumn>
      <matchColumn>
        <columnName>Enterprise.SystemSBR.Person.Address.Staddress</columnName>
        <matchType>StAddress</matchType>
      </matchColumn>
      </matchColumns>

      </childMatchSet>
      </matchSet>
   <fieldSubstitution>
    <matching>
     <substituteField>
      <targetField>
         Enterprise.SystemSBR.Person.FirstName
      </targetField>
      <sourceField>
         Enterprise.SystemSBR.Person.LastName
      </sourceField>
      <sourceField>
         Enterprise.SystemSBR.Person.Alias.FirstName
      </sourceField>
     </substituteField>
    </matching>
    <blocking>
     <substituteField>
      <targetField>Enterprise.SystemSBR.Person.FirstName_phon</targetField>
      <sourceField>Person.LastName</sourceField>
      <phoneticEncodertype>Soundex</phoneticEncodertype>
     </substituteField>
    </blocking>
   </fieldSubstitution>

 <frequencyBasedFields>
   <field>
    <fieldName>Enterprise.SystemSBR.Person.FirstName_Std</fieldName>
   </field>
   <alternateMatchColumn>
      <columnName>Enterprise.SystemSBR.Person.FirstName_Std</columnName>
      <matchType>LastName</matchType>
   </alternateMatchColumn>
   </field>
```

```
   <maxPercentWeightVariation>50</maxPercentWeightVariation>
   <frequencyWeightReducerPlugin>
      customPackage.CustomFreuqencyReducer
   </frequencyWeightReducerPlugin>
 </frequencyBasedFields>
</MatchExtConfiguration>
```

XML File Explanation

This section explains the match extensions in the `match-ext.xml` file. When setting up matching, simply comment out the match extensions you do not want to use for matching in the `match-ext.xml` file.

> **Note:** The sample `match-ext.xml` file that is delivered with OHMPI is commented out. To use this file you must remove these comments at the beginning (`<!--`) and end (`-->`) of the sample file. Also, you need to modify the object names and field names to your project object model and configure it based upon your business needs.

```
<MatchExtConfiguration>
```

/** There can be multiple match sets. A matchSet has a unique String ID. Each match set is evaluated independently. The match score evaluated from all the match sets for a given tuple is the one having the highest match score. A matchSet can contain child-matchSets, but a child matchSet should be unconditional.

```
*/
<matchSet ID="1">
  <scoreMultiplier>.9</scoreMultiplier>
/** A scoreMultiplier, is multiplied with the match score computed for that set of
match fields, to determine total score. */
   <matchColumns>
    <matchColumn>
       <columnName>Enterprise.SystemSBR.Person.FirstName_Std</columnName>
       <matchType>FirstName</matchType>
/** agreementWeight & disagreementWeight are optional fields. The default value is
chosen from matchconfig.cfg */
       <agreementWeight>10</agreementWeight>
       <disagreementWeight>-10</disagreementWeight>
     </matchColumn>
     <matchColumn>
       <columnName>Enterprise.SystemSBR.Person.LastName_Std</columnName>
       <matchType>LastName</matchType>
     </matchColumn>
   </matchColumns>
/*
```

The given match set has a condition associated with it. A given pair of tuples are compared using a particular matchSet only if Conditions for this matchSet is evaluated to **true**. Conditions can be composed of many individual fieldCondition instances. Each fieldCondition has an implicit "AND" operator with other fieldCondition instances. A fieldCondition is **true** only if the field at run time has a value that is specified in a value element in the fieldCondition. The set of fieldConditions also has an implicit "AND" with systems. However, the set of systems within the *systems* element has an implicit "OR" operator. So this particular matchSet is run through the match engine only if the given Person object at run time has ID="DL" and belongs to either system HospitalA or HospitalB.

The *conditions* element can be empty, which implies that it is an unconditional matchSet. Similarly, the *systems* element can be empty, which means this "conditions" is **true** for all systems.

```
*/
<conditions>
  <fieldCondition>
    <field>Enterprise.SystemSBR.Person.ID</field>
    <value>DL</value>
  </fieldCondition>
  <systems>
    <system>HospitalA</System>
    <system>HospitalB</System>
  </systems>
</conditions>
/** childMatchSet contains set of match columns. Total cap for aggregate of
agreement and disagreement weights can be specified in childMatchSet attributes*/
<childMatchSet capAgreement="10" capDisagreement="-10">
  <matchColumns>
     <matchColumn>
       <columnName>Enterprise.SystemSBR.Person.Phone.PhoneNum</columnName>
       <matchType>Phone</matchType>
       <agreementWeight>10</agreementWeight>
       <disagreementWeight>-10</disagreementWeight>
     </matchColumn>
     <matchColumn>
       <columnName>Enterprise.SystemSBR.Person.Address.Staddress</columnName>
       <matchType>StAddress</matchType>
     </matchColumn>
  </matchColumns>
</childMatchSet>
</matchSet>
/**
```

A directive to the Match Enhancer to use values from sourceField as a value for targetField during the matching of targetField. The values for fields specified in sourceField will be used in addition to the value for targeField at runtime matching. For example, If FirstName = John, lastName = Smith, and Alias.FirstName = Jack, then the first name comparisons would include the values ['John','Smith','Jack']. These can be used both at matching and for blocking.

Additionally, if PhoneticEncoder-type is specified, then that encoding is executed on the Blocking.sourceField value, before using it as additional value for targetField blocking.

```
*/
<fieldSubstitution>
  <matching>
    <substituteField>
      <targetField> Enterprise.SystemSBR.Person.FirstName</targetField>
       <sourceField weightMultiplier=.9>
         Enterprise.SystemSBR.Person.LastName
       </sourceField>
       <sourceField>
         Enterprise.SystemSBR.Person.Alias.FirstName
       </sourceField>
     </substituteField>
   </matching>
   <blocking>
     <substituteField>
       <targetField>
         Enterprise.SystemSBR.Person.FirstName_ph
```

```
      </targetField>
      <sourceField>
        Enterprise.SystemSBR.Person.lastName
      </sourceField>
      <PhoneticEncoder-type>S oundex</PhoneticEncoder-type>
    </substituteField>
  </blocking>
 </fieldSubstitution>
 /**
```

Frequency based scoring is applied to set of <field>. A field includes elements:

- **Fieldname** is a fieldname whose score is reduced based upon the frequency of the given string that requires matching.
- **maxPercentWeightVariation** specifies the maximum percentage in the score that can be reduced, regardless of any frequency-based agreement weight reduction rules.
- **FrequencyWeightReducerPlugin** specifies a frequency reducer that can evaluate how much *agreementWeight* of a given field should be reduced based upon the frequency percentage of a given field value.

```
*/
    <frequencyBasedFields>
      <field>
        <fieldName>
          Enterprise.SystemSBR.Person.FirstName_Std
        </fieldName>
        <alternateMatchColumn>
          <columnName>
            Enterprise.SystemSBR.Person.FirstName_Std
          </columnName>
          <matchType>LastName</matchType>
        </alternateMatchColumn>
      <field>

      <maxPercentWeightVariation>50</maxPercentWeightVariation>
      <frequencyWeightReducerPlugin>
        customPackage.CustomFreuqencyReducer
      </frequencyWeightReducerPlugin>
    </frequencyBasedFields>
  </MatchExtConfiguration>
```

Use the XML Editor to define the different MatchSets and FrequencyBasedFields.

Frequency Weight Reducer Plugin Interface

```
package com.sun.mdm.index.matching.matchingext;

import java.util.Map;
import java.util.HashMap;
import java.util.List;
import java.util.Set;
import com.sun.mdm.index.matching.matchingext.generated.FrequencyBasedFields;
import com.sun.mdm.index.matching.matchingext.generated.Field;
import com.sun.mdm.index.util.ConnectionUtil;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
/**
 * Computes reduction factor for a corresponding field and value based on its
input frequency. This reduction factor
 * is used to reduce the agreement weight for such field value. String with higher
frequencies
 * should return a lower reduction factor. OHMPI frequency based normalizer has a
reduction factor algorithm based on frequency
 * of a name value. So this plugin should be written only if the default agreement
reduction provided by OHMPI needs to be changed.
 * A (reduction factor returned by this method)*(agreement Weight of a field),
gives the new agreement weight for the field.
 *
 *
 * @author  sdua
 * @version $Revision: 1.1 $
 */

public interface FrequencyWeightReducer {

    /**
      returns agreement weight reduction factor for a String that participates in
matching process.
    @param field field name for which reduction factor should be computed
    @param value value for the field name. value typically would mean person's
name.
    @param percentFrequency percentage frequency of this field value relative to
all other values.
    The implementation class for this interface needs to be specified in
match-ext.xml.
   */
double computeReductionFactor(String field, String value, double
percentageFrequency);
}
```

Default Behavior of Frequency-based Reduction in Agreement Weights

The reduction in agreement weight due to high frequency of a word follows a linear function but which has not one adjustment rate (slope) but gives different weight adjustment rates at different frequency ranges. Below are the ranges of frequencies, for which there are corresponding ranges of agreement adjustment weights. So the lower and higher word percentFrequency corresponds to lower and higher range of agreement multiplication factor. Any weight reduction within a range of percentFrequency is proportionally reduced.

if (percentFrequency > 10%) new agreementWt = 20% of original agreementWt

else if (percentFrequency is 5% - 10%) new agreementWt = 20% - 30% of original agreeementWt

else if (percentFrequency is 1% - 5%) new agreementWt = 30% - 50% of original agreeementWt

else if (percentFrequency is .1% - 1%) new agreementWt = 50% - 70% of original agreeementWt

else if (percentFrequency is .01% - .1%) new agreementWt = 70% - 90% of original agreeementWt

Setting Up the match-ext.xml to Perform Matching

Basically all you need to do to set up matching is edit the `match-ext.xml` file. To do this:

1. Start NetBeans and open a project.
2. Select the Open Project icon in the NetBeans toolbar.
3. In the Open Project window select the project you want to open and click **Open Project**.
4. In the Projects pane (on the left side of NetBeans) and under Configuration (which is open), right-click **match-ext.xml** and select **Edit**.

   The XML Editor opens.
5. Edit the `match-ext.xml` file, changing the values based upon your matching needs and comment out matching that you do not want to use.
6. Save the file.
7. Deploy your project
8. Open your project in the MIDM or in a web services client.

Current Matching Configuration

This section provides sample files of `mefa.xml` and `matchConfigFile.cfg`.

Note: There are no changes in the `mefa.xml` and `matchConfigFile.cfg` files.

Sample of Existing mefa.xml

```
<match-system-object>
        <object-name>MPI1</object-name>
        <match-columns>
          <match-column>
            <column-name>Enterprise.SystemSBR.MPI1.FirstName_Std</column-name>
            <match-type>FirstName</match-type>
          </match-column>
          <match-column>
            <column-name>Enterprise.SystemSBR.MPI1.LastName_Std</column-name>
            <match-type>LastName</match-type>
          </match-column>
         </match-columns>
<match-system-object>
```

`match-columns`, which is defined in the current `mefa.xml` file, forms a default MatchSet.

If the agreements and disagreements for any match-columns are not defined in `mefa_ext.xml`, they would be used from `matchConfigFile.cfg`.

Sample of Existing matchConfigFile.cfg

```
PrimaryName             30  0   us    0.9   0.001   13  -2
StreetName              25  0   us    0.9   0.001   11   0
HouseNumber             8   0   un    0.9   0.001   11   0
StreetDir               15  0   u     0.9   0.001   7   -2
StreetType              10  0   u     0.9   0.001   7   -2
```

```
FirstName              15  0   uf    0.99  0.001   10  -4
LastName               15  0   ul    0.99  0.001   10  -4
String                 25  0   us    0.99  0.001   10  -10
```

The mefa.xml and matchconfig.cfg files contain the default set of agreement and disagreement weights. Each individual MatchSet has its own set of agreement weights which override the default weights in the mefa.xml file.

The comparators for match types are stored in MatchEngine and are not configurable from mefa_ext.xml.

Note: There is no direct mapping between mefa_ext.xml source field (match column) such as Enterprise.SystemSBR.Person.FirstName_Std and the comparator defined in matchConfigFile.cfg. This mapping is provided via the matchType column in mefa_ext.xml and match type in matchConfigFile.cfg.

Using Previous Projects with this Release

Any project created in a previous release of OHMPI will work seamlessly with this release.

- To use a project from a previous release without any of the new matching features for a component in this release:
  - Import the previous project and re-deploy it.
- To use a project from a previous release with matching in this release:
  - Import the previous project, configure it for the new matching, regenerate, clean and build, and then re-deploy it.

    Note: When you import (that is copy) a project from a previous release of OHMPI into this release of OHMPI, the project from the previous release has matching and standardization as created in the previous release. To use the matching and standardization features that come with this release you need to create a new project, which in turn creates matching and standardization. You then need to copy the new matching and standardization files that you just created into the appropriate directory of the project that you have imported into this release of OHMPI.

  - If you import a previous project, you also need to copy the sbyn_frequency table that is associated with the previous project, as the sbyn_frenquency table is created when a project is created.

5

OHMPI Match Engine Comparison Functions

This chapter introduces you to and provides conceptual information about the OHMPI Match engine comparison functions.

This chapter includes the following section:

Learning About the OHMPI Match Engine Comparison Functions

Match field comparison functions, or **comparators**, compare the values of a field in two records to determine whether the fields match or how closely they match. The fields are then assigned a matching weight based on the results of the comparison function. You can use several different types of comparison functions in the match configuration file in order to customize how the OHMPI Match Engine matches records. The comparators themselves are highly configurable and can be configured to assign differing weights or handle null values. Several comparators accept parameters that further fine-tune the matching process.

The OHMPI Match Engine provides a comprehensive group of match comparison functions to enable matching on a wide variety of data. While you should be able to configure any of the default comparison functions to accurately match your data, you can create new comparison functions and integrate them into a master person index application. For more information, see Chapter 6, "Creating Custom Comparators for the OHMPI Match Engine".

Certain comparison function types are very specific to the type of data being matched, such as the numeric functions and the date functions. Others, such as the Bigram and uncertainty functions, are more general and can be applied to various data fields.

Certain comparison function types are very specific to the type of data being matched, such as the numeric functions and the date functions. Others, such as the Bigram and uncertainty functions, are more general and can be applied to various data fields.

Be sure to review Table 2–1 for information about how the parameters in the match configuration file affect the outcome of the comparator functions. For example, parameters define how null fields are handled and what the actual agreement and disagreement weights are.

Note: The names of the comparators are configurable. The default names are used here.

Bigram Comparators

The OHMPI Match Engine provides two different comparison functions based on the Bigram algorithm, the standard bigram (b1) and the transposition bigram (b2). A Bigram algorithm compares two strings using all combinations of two consecutive characters within each string. For example, the word "bigram" contains the following bigrams: "bi", "ig", "gr", "ra", and "am". The Bigram comparison function returns a value between 0 and 1, which accounts for the total number of bigrams that are in common between the strings divided by the average number of bigrams in the strings. Bigrams handle minor typographical errors well.

Bigram Comparator (b1)

The Bigram Comparator is a standard Bigram comparison function, processing match fields as described above. This comparison function takes no parameters.

Advanced Bigram Comparator (b2)

The Advanced Bigram Comparator is based on the standard Bigram comparison function, but handles transpositions of characters within a string. This comparison function takes no parameters.

Uncertainty String Comparators

The OHMPI Match Engine provides several uncertainty comparison functions for comparing string fields. Most uncertainty comparison functions are generic, but some comparison functions are designed for specific types of information (first name, last name, house number, and national identifiers).

The uncertainty functions include the following:

- Advanced Jaro String Comparator (u) on page 3
- Winkler-Jaro String Comparator (ua) on page 3
- Condensed String Comparator (us) on page 3
- Advanced Jaro Adjusted for First Names (uf) on page 4
- Advanced Jaro Adjusted for Last Names (ul) on page 4
- Advanced Jaro Adjusted for House Numbers (un) on page 4
- Advanced Jaro AlphaNumeric Comparator (ujs) on page 4
- Unicode String Comparator (usu) on page 4
- Unicode AlphaNumeric Comparator (usus) on page 5
- Chinese String Comparator (usc) on page 6
- Chinese String Prefix Comparator (cc) on page 6

Advanced Jaro String Comparator (u)

The Advanced Jaro String Comparator is the standard uncertainty comparison function for processing string fields. This comparison function is based on the Jaro algorithm with McLaughlin adjustments for similarities. The Jaro algorithm is a string comparison function that accounts for insertions, deletions, and transpositions by performing the following steps.

1. Compute the lengths of both strings to be matched.
2. Determine the number of common characters between the two strings. In order for characters to be considered common, they must be within one-half the length of the shorter string.
3. Determine the number of transpositions. A transposition means a character from the first string is out of order with the corresponding common character from the second string.

As more differences are found between two fields, the agreement weight decreases nonlinearly. Thus, the agreement weight can remain high for several differences, but will drop sharply at a certain point. This comparison function takes no parameters.

Winkler-Jaro String Comparator (ua)

The Winkler-Jaro String Comparator is based on the standard uncertainty comparison function, **u**, with variants of Winkler / Lynch and McLaughlin. It has additional features to handle specific differences between fields, such as key punch and visual memory errors. Each feature makes use of the information made available from previous features. This comparison function takes no parameters.

The following features are included in the advanced uncertainty function.

- The function determines each character in exact agreement and then assigns a value of 1.0 to each agreeing character. It then determines each disagreeing but similar character and assigns a value of 0.3 to each. Similar characters might occur because of scanning errors (for example, inserting "1" the number instead of "l" the letter) or keypunch errors (for example, typing "S" instead of "D").
- The function gives increased value to agreement on the beginning characters of a string. The algorithm adjusts the weighting value up by a fixed amount if the first four characters in each string agree; it adjusts the weighting value up by smaller value if only the first three, two, or one characters agree.
- The function adjusts the string comparison value if the strings are longer than six characters and more than half of the characters after the fourth character agree.

Condensed String Comparator (us)

The Condensed String Comparator is a custom version of a generic string comparison function. It is similar to the Advanced Jaro String Comparator, **u**, but processes data in a more simple and efficient manner, improving processing speed. The agreement weights generated by this comparison function decrease in a more uniform manner for each difference found between two fields.

Like the Advanced Jaro String Comparator, the Condensed String Comparator takes into account such uncertainty factors as string length, transpositions, key punch errors, and visual memory errors. Unlike the uncertainty comparison function ("u"), this function handles diacritical marks. This comparison function takes no parameters.

Advanced Jaro Adjusted for First Names (uf)

The Advanced Jaro Adjusted for First Names comparator is designed specifically for matching on first name fields, and is based on the Condensed String Comparator, **us**. This comparison function analyzes the string and then adjusts the weight based on statistical data. This comparison function takes no parameters.

Advanced Jaro Adjusted for Last Names (ul)

The Advanced Jaro Adjusted for Last Names comparator is designed specifically for matching on last name fields, and is based on the Condensed String Comparator, **us**. This comparison function analyzes the string and then adjusts the weight based on statistical data. This comparison function takes no parameters.

Advanced Jaro Adjusted for House Numbers (un)

The Advanced Jaro Adjusted for House Numbers comparator is designed specifically for matching on house numbers, and is based on the Condensed String Comparator, **us**. This comparison function analyzes the string and then adjusts the weight based on statistical data. This comparison function takes no parameters.

Advanced Jaro AlphaNumeric Comparator (ujs)

The Advanced Jaro AlphaNumeric Comparator is a custom version of a generic string comparison function. It is based on the Advanced Jaro String Comparator, **u**, but is designed specifically for matching on national identifier, such as social security numbers. This function takes into account such uncertainty factors as string length, transpositions, key punch errors, and visual memory errors. It can also take into consideration field length, allowed character types, and invalid values. This comparison function takes the parameters described in Table 5–1.

Table 5–1 ujs Comparison Function Parameters

Parameter	Description
ssnLength	An optional parameter that takes the length of the field value into account. If a fixed length is specified, the match engine considers any field of a different length to be a non-match. Specify any integer smaller than the value specified for the field size in the matching configuration file (for more information, see Matching Rules Section).
recType	An indicator of whether the field must be all numeric. Specify "nu" for numeric only, or specify "an" to allow alphanumeric characters. The match engine considers any fields containing characters that are not allowed to be a non-match.
ssnList	A list of invalid characters for the field. If you specify a character, the match engine considers fields that consist of only that character to be a non-match. For example, if you specify "0", then an SSN field cannot contain all zeros. Specify as many alphanumeric characters as needed, separated by a space.

Unicode String Comparator (usu)

The Unicode String Comparator is a custom version of a generic string comparison function. It is similar to the Condensed String Comparator, **us**, but is based in Unicode to enable multilingual support. This locale-oriented comparator recognizes the nuances of each language and supports the complexities and subtleties of each. For example, when configured to use the German language set, the function recognizes "ß" and "ss" as equivalent. Like the simplex uncertainty function, the Unicode function takes into account such uncertainty factors as string length, transpositions, key punch errors, and visual memory errors. This comparison function takes the parameter described in Table 5–2.

Table 5–2 usu Comparison Function Parameter

Parameter	Description
language	An indicator of the language being used for the information stored in the database. Enter one of the following codes to indicate the language in use. **da** - Danish **sv** - Swedish **nb** - Norwegian Bokmål **nn** - Norwegian Nynorsk **nl** - Dutch **es** - Spanish **fr** - French **en** - English **it** - Italian **de** - German

Unicode AlphaNumeric Comparator (usus)

This comparison function is a custom version of a generic string comparison function. It is similar to the Unicode String Comparator, but it is also similar to the Advanced Jaro AlphaNumeric Comparator in that it is designed to work on national identifiers like social security numbers. This locale-oriented comparator recognizes the nuances of each language and supports the complexities and subtleties of each. This function takes into account such uncertainty factors as string length, transpositions, key punch errors, and visual memory errors. It can also take into consideration field length, allowed character types, and invalid values. This comparison function takes the parameters described in Table 5–3.

Table 5–3 usus Comparison Function Parameters

Parameter	Description
language	An indicator of the language being used for the information stored in the database. Enter one of the following codes to indicate the language in use. **da** - Danish **sv** - Swedish **nb** - Norwegian Bokmål **nn** - Norwegian Nynorsk **nl** - Dutch **es** - Spanish **fr** - French **en** - English **it** - Italian **de** - German
fixed-length	An optional parameter that takes the length of the field value into account. If a fixed length is specified, the match engine considers any field of a different length to be a non-match. Specify any integer smaller than the value specified for the size specified for the field (for more information, see Matching Rules Section).
character-type	An indicator of whether the field must be all numeric. Specify "nu" for numeric only, or specify "an" to allow alphanumeric characters. The match engine considers any fields containing characters that are not allowed to be a non-match.
invalid-characters	A list of invalid characters for the field. If you specify a character, the match engine considers fields that consist of only that character to be a non-match. For example, if you specify "0," then an SSN field cannot contain all zeros. Specify as many alphanumeric characters as needed, separated by a space.

Chinese String Comparator (usc)

The Chinese String Comparator is a custom version of a generic string comparison function. It is a locale-specific comparator that takes Chinese characters' structure into consideration during comparison. Character attributes such as radicals, stroke numbers and several structure-based encoding methods are taken into consideration when determining the match score between two Chinese characters. This comparison function takes no parameters.

Chinese String Prefix Comparator (cc)

The Chinese String Prefix Comparator is similar to Chinese String Comparator. The only difference is it only compares the first pre-determined length of characters without considering gaps. This comparison function takes no parameters.

Exact Character-to-Character Comparator (c)

The OHMPI Match Engine provides one exact-match comparison function, "c." With this comparison function, two fields must match exactly on each character in order to be considered a match. This comparison function takes no parameters.

Numeric Comparators

The OHMPI Match Engine provides two comparison functions for matching on numeric fields:

- Integer Comparator (nl) on page 6
- Real Number Comparator (nR) on page 7
- Chinese Integer Comparator (nIc) on page 7

The Integer Comparator and Real Number Comparator can perform either numeric string comparisons or relative distance calculations. When set for a string comparison, the functions compare numeric strings based on the advanced uncertainty comparator. When set for relative distance calculations, the matching weight between two numbers decreases as the numbers become further apart, until the relative distance plus one is reached. At this point, the numbers are considered non-matches. For example, if the relative distance is "10" and the base number for comparison is "2," a field value of 8 receives a lower matching weight than a field value of 4; but a field value of 13 is considered a complete non-match (since the distance between 2 and 13 is 11).

Integer Comparator (nl)

The Integer Comparator matches specifically on integers using the logic describe above. It accepts the parameters listed in Table 5–4.

Table 5–4 nI Comparison Function Parameters

Parameter	Description
switch	Specifies whether a relative distance calculation or a direct string comparison is used. Specify "y" to use a relative distance calculation; specify "n" to use a string comparison.
range	The greatest difference between two integers at which the values could still be considered a possible match. When the difference between two numbers is greater than the relative distance, the numbers are considered a non-match (the weight becomes zero when the actual difference is the relative distance plus one).

Real Number Comparator (nR)

The Real Number Comparator function matches specifically on real numbers based on the logic described above. It accepts the parameters listed in Table 5–5.

Table 5–5 nR Comparison Function Parameters

Parameter	Description
switch	Specifies whether a relative distance calculation or a direct string comparison is used. Specify "y" to use a relative distance calculation; specify "n" to use a string comparison.
range	The greatest difference between two integers at which the values could still be considered a possible match. When the difference between two numbers is greater than the relative distance, the numbers are considered a non-match (the weight becomes zero when the actual difference is the relative distance plus one).

Chinese Integer Comparator (nIc)

The Chinese Integer Comparator is an extension of the Integer Comparator (nI) to support numbers written in Chinese characters. Such numbers are translated into Arabic numbers before the comparison is done by the Integer Comparator.

Condensed AlphaNumeric SSN Comparator (nS)

The Condensed AlphaNumeric SSN Comparator is designed specifically for matching on numeric strings and is very useful for matching social security numbers or other unique identifiers. This comparison function can compare either alphanumeric values or numeric values, and takes into account such uncertainty factors as string length, transpositions, key punch errors, and visual memory errors. It can also take into consideration field length, allowed character types, and invalid values. It accepts the parameters listed in Table 5–6.

Table 5–6 nS Comparison Function Parameters

Parameter	Description
fixed-length	An optional parameter that takes the length of the field value into account. If a fixed length is specified, the match engine considers any field of a different length to be a non-match. Specify any integer smaller than the value specified for the size specified for the field (for more information, see Matching Rules Section).
character-type	An indicator of whether the field must be all numeric. Specify "nu" for numeric only, or specify "an" to allow alphanumeric characters. The match engine considers any fields containing characters that are not allowed to be a non-match.
invalid-characters	A list of invalid characters for the field. If you specify a character, the match engine considers fields that consist of only that character to be a non-match. For example, if you specify "0," then an SSN field cannot contain all zeros. Specify as many alphanumeric characters as needed, separated by a space.

Date Comparators

The OHMPI Match Engine provides various date comparison functions. When comparing dates, the match engine compares each date component (for example, it compares the year in the first date against the year in the second date, the month against the month, and the day against the day). This allows for multiple transpositions in each date field. The date comparators use the Java date format (java.sql.Date), allowing the comparator to use the Gregorian calendar and to take into account the time zone where the date field originated.

The following comparison functions are available for matching on date fields.

- Date Comparator With Years as Units (dY) on page 8
- Date Comparator With Months as Units (dM) on page 8

As with the numeric comparison functions, the date comparison functions can use either a direct string comparison or a relative distance calculation (see Numeric Comparators). When using a relative distance calculation, the matching weight between two dates decreases as the dates become further apart, until the relative distance is reached. When the difference becomes the relative distance plus one, the dates are considered non-matches. You can specify different relative distances for before and after the given date. Any dates falling outside of the specified time period receive a complete disagreement weight. The relative distances are specified in the smallest unit of time being matched.

Continuing, as the weight is decreased, when the difference between the two compared fields reaches either the before or after relative distance. For example, if the before relative distance is **11** and the after relative distance is **5**, if this example had been charted a light blue line would represent the agreement weight. When the base date is later than the compared date and the difference between the dates reaches **11** (distance before plus one), the fields are considered a non-match and are given the full disagreement weight. When the base date is earlier than the compared date and the difference between the dates reaches **6** (distance after plus 1), the fields are considered a non-match.

The date comparison functions take the parameters listed in Table 5–7.

Table 5–7 Date Comparison Function Parameters

Parameter	Description
switch	Specifies whether a relative distance calculation or a direct string comparison is used. Specify "y" to use a relative distance calculation; specify "n" to use a string comparison.
llimit	The number of units prior to the reference date/time for which two date fields can still be considered a match.
ulimit	The number of units following the reference date/time for which two date fields can still be considered a match.

Date Comparator With Years as Units (dY)

This date comparison function takes only the 4-character year into account for matching. If relative distance calculation is specified, the relative distance is specified in years.

Date Comparator With Months as Units (dM)

This date comparison function takes the month and year into account for matching. If relative distance calculation is specified, the relative distance is specified in months.

Date Comparator With Days as Units (dD)

This date comparison function takes the day, month, and year into account for matching. If relative distance calculation is specified, the relative distance is specified in days.

Date Comparator With Hours as Units (dH)

This date comparison function takes the hour, day, month, and year into account for matching. If relative distance calculation is specified, the relative distance is specified in hours.

Date Comparator With Minutes as Units (dm)

This date comparison function takes the minute, hour, day, month, and year into account for matching. If relative distance calculation is specified, the relative distance is specified in minutes.

Date Comparator With Seconds as Units (ds)

This date comparison function takes the second, minute, hour, day, month, and year into account for matching. If relative distance calculation is specified, the relative distance is specified in seconds.

Prorated Comparator (p)

The Prorated Comparator uses a relative distance calculation and allows you to specify how quickly the agreement weight between two fields decreases. Matching weights are assigned with a linear adjustment according to the parameters you specify. You specify an initial agreement range. If the difference between two fields falls within that range, the fields are considered a complete match. You also specify a disagreement range ending with the relative distance. If the difference between two fields falls within that range, the fields are considered a non-match. When the difference between the fields falls between those two ranges, they are considered to be partial matches and the agreement weight is adjusted linearly. Any difference greater than the relative distance is always considered a non-match.

Note: Increasing the disagreement weight causes the prorated agreement weight to decrease more sharply.

The prorated comparison functions takes the parameters listed in Table 5–8.

Table 5–8 Prorated Comparison Function Parameters

Parameter	Description
range	The greatest difference between two numbers at which they can still be considered a match or partial match.
tolerance1	The greatest difference between two numbers at which they are considered a full match. This number must be less than the relative distance.
tolerance2	This number indicates the minimum difference at which two numbers are considered a non-match and shortens or lengthens the weighting scale. To find this difference, the match engine subtracts this value from the relative distance. If the fields differ by that amount or greater, they are considered to be a non-match. The weighting scale decreases in size as the value of the full-disagreement parameter increases (see diagram).

6

Creating Custom Comparators for the OHMPI Match Engine

This chapter introduces you to conceptual information about configurable matching comparators, lists components that complete a comparator package, and provides a procedure that defines a custom comparator.

This chapter includes the following sections:

Learning About Custom Comparator for the OHMPI Match Engine

The OHMPI Match Engine provides a variety of configurable matching comparators for you to process and match your data. However, if none of the existing comparators meet your requirements, the flexible framework of the OHMPI Match Engine allows you to create custom comparators to plug in to master person index applications. The comparators are flexible components that can be modified and tailored without requiring any changes to the framework.

The following sections provide an overview of custom comparators and information about the comparator package:

Custom Comparator Overview

Creating a custom matching comparator for the OHMPI Match Engine requires coding the processing and validation logic for the comparator in Java. The OHMPI Match Engine provides the interfaces and supporting Java classes you need to implement in order to incorporate the comparators into a master person index application.

The OHMPI Match Engine framework consists of two modules. The real-time module stores the basic logic for the matching comparators. The design-time module stores all of the configuration logic for the comparators, including parameter validations, data source definitions, and curve adjustment logic. The two pieces are pulled together by the configuration in the comparators list file (`comparatorsList.xml`). For each custom comparator package you create, you need to create a comparators list file.

You can define the following information in the comparators list for each comparator you create.

- A code that is used to reference the comparator in the match configuration file (`matchCOnfigFile.cfg`).
- The class that defines the comparator logic.
- Parameters for the comparator. Parameter values are entered in the match configuration file for any entries that reference the comparator.
- Any classes from which the comparator class inherits.
- Data sources that provide additional information to the comparator during the match process.
- Whether to use curve adjustment logic for the comparator.

After you create the package, you can import the custom comparators into NetBeans using the easy import function of Oracle Healthcare Master Person Index. When you import the files, OHMPI automatically validates the files and merges the comparators list information into the comparators list for the application. You can then add and configure entries in `matchConfigFile.cfg` that reference the comparator, which makes the comparator available to be used in the match string.

About the Comparator Package

After you register your custom comparators and you create and compile the comparators and any configuration classes, you need to package the files in a ZIP file so they are available for import into NetBeans. For optimal usage, it is best to package all similar comparators in a unique ZIP file. You can create single packages for each comparator, or combine them into one package.

The ZIP file includes the following:

- The comparator Java classes
- The comparators list file (`comparatorsList.xml`)
- Any parameter validation classes (only if the comparators take parameters)
- Any data source loading or validation classes (only if the comparators use external data files)
- Any curve adjustment classes (only if the comparators use curve adjustment for weight calculation)

For the ZIP file to have the correct structure, the `comparatorsList.xml` file should be at the same level as the com folder that contains the Java classes. The following figure shows a sample ZIP file for custom comparators.

Defining Custom Comparators

The following topics provide instructions for each step of creating custom comparators. You might need to create multiple Java files and Java packages for the comparator, depending on the validations, data sources, dependency classes, and curve adjustments you use. Create them in the same directory structure because you will need to package them up into a ZIP file when you are through.

Before you create your custom comparators, take into account the following requirements for the comparators.

- Determine how many comparators you need to create and whether each will require a different Java class or some can use the same Java class.
- Determine what parameters, if any, you need to define for each comparator.

- Determine what validations, if any, need to be created.
- Determine whether you need to use a data source.
- Decide if the comparators you create will have a dependency on any other comparator classes.
- Decide whether you will use curve adjustment, linear fitting, or neither.

The following steps lead you through creating a custom comparator:

Step 1: Create the Custom Comparator Java Class

The first step to creating custom comparators is defining the matching logic in custom comparator Java classes that are stored in the real-time module of the OHMPI Match Engine. Follow these guidelines when creating the class:

- Create a working directory that will contain all the Java packages and the comparators list file for the new comparators.
- The Java classes need to implement `com.sun.mdm.matcher.comparators.MatchComparator.java` interface, located in `Matcher.jar`. This class includes the methods described below.

Once you create the Java classes, continue to Step 2: Register the Comparator in the Comparators List.

initialize

- Description: The `initialize` method initializes the values for the parameters, data sources, and dependency class used for each custom comparator. It provides the necessary information to access the comparator's configuration in the match configuration file and the comparators list file.
- Syntax: `void initialize(Map<String, Map> params, Map<String, Map> dataSources, Map<String, Map> dependClassList)`
- Parameters:

Table 6–1 initialize Parameters

Parameter	Type	Description
params	Map	A mapping of all the parameters associated with a match field in matchConfigFile.cfg.
dataSources	Map	A mapping of all the data sources associated with a match field in `matchConfigFile.cfg`.
dependClassList	Map	A mapping of all the dependency classes associated with a match field in `matchConfigFile.cfg`.

- Return Value: None.
- Throws: None.

compareFields

- Description: The `compareFields` method contains all the comparison logic needed to compare two field values and calculate a matching weight that shows how similar the values are.
- Syntax: `double compareFields(String recordA, String recordB, Map context)`
- Parameters:

Table 6–2 compareFields Parameters

Parameter	Type	Description
recordA	String	A field value from the record against which the reference record is being compared.
recordB	String	A field value from the reference record.
context	Map	A set of arguments passed to the comparator.

- Return Value: A number between zero and one that indicates how closely two field values match.
- Throws: `MatchComparatorException`

setRTParameters

- Description: The `setRTParameters` method sets the runtime parameters for the comparator, providing the ability to customize every call to the parameter.
- Syntax: void `setRTParameters(String key, String value)`
- Parameters:

Table 6–3 setRTParameters Parameters

Parameter	Type	Description
key	string	The key to map the parameter value.
value	string	The value of the parameter.

- Return Value: None.
- Throws: None.

stop

- Description: The stop method closes any related connections to the data sources used by the comparator.
- Syntax: `void stop()`
- Parameters: None.
- Return Value: None.
- Throws: None.

Step 2: Register the Comparator in the Comparators List

In order to include new comparators in a master person index application, you need to create a comparators list file defining the configuration of the comparators. When you import the comparator package into the master person index application, this file is read and the entries are added to the comparators list for the project.

Below is a sample comparators list file. Note that the first comparator includes all possible configurations (parameters, dependency classes, data sources, and curve adjust). Most comparators will not be that complex. The second comparator class defines two comparators, Approx and Adjust.

```
<?xml version="1.0" encoding="UTF-8"?>
<comparators-list xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="comparatorsList.xsd">
    <group description="New group of comparators"
     path="com.mycomparators.matchcomparators">
        <comparator description="New Exact Comparator">
            <className>NewExactComparator</className>
            <codes>
                <code description="New Exact Comparator" name="Exact" />
            </codes>
            <params>
              <param description="Fixed length" name="length"
               type="java.lang.Integer" />
              <param description="Data type" name="dataType"
               type="java.lang.String" />
            </params>
            <data-sources>
              <datasource description="Serial numbers" type="java.io.File" />
            </data-sources>
            <dependency-classes>
              <dependency-class matchfield="Serial"
               name="com.genericcomparaotrs.StringComparator" />
            </dependency-classes>
            <curve-adjust status="true" />
        </comparator>
       <comparator description="New Approximate Comparator">
            <className>NewApproxComparator</className>
            <codes>
                <code description="New approximate comparator" name="Approx" />
                <code description="New adjustable comparator" name="Adjust" />
            </codes>
        </comparator>
     </group>
</comparators-list>
```

To Register the Comparators

1. Complete Step 1: Create the Custom Comparator Java Class.
2. In the same folder where you created the custom Java class package, create a new file named `comparatorsList.xml`.

   Tip: The comparators list file needs to be in the same working directory you created for the custom comparator Java classes.

3. Add the following header information to the file. You can copy this from the `comparatorList.xml` file in a master person index application.

```
<?xml version="1.0" encoding="UTF-8"?>
<comparators-list xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="comparatorsList.xsd">
 ...
</comparators-list>
```

4. Define the following properties, using the XML structure described in Learning About the OHMPI Match Engine Comparator Definition List. Use the sample above as an example.
   - The group description and Java package for the group.
   - A description for each comparator.
   - The Java class name for each comparator or comparator subgroup.
   - The unique identifying name for each comparator.
   - A list of static parameters for each comparator or comparator subgroup (optional). If you define parameters, you must also perform the steps under Step 3: Define Parameter Validations (Optional).
   - A list of data sources for each comparator or comparator subgroup (optional). If you define data sources, you must also perforStep 4: Define Data Source Handling (Optional).
   - A list of dependency classes for each comparator or comparator subgroup (optional).
   - Whether to use curve adjustment for each comparator or comparator subgroup (optional). If you set curve adjustment to true, you must perform the steps under Step 5: Define Curve Adjustment or Linear Fitting (Optional).
5. Continue to Step 3: Define Parameter Validations (Optional)

Step 3: Define Parameter Validations (Optional)

If your custom comparators take parameters, you should create a Java class that validates the parameter properties. You need to perform this step if you defined parameters for the comparator in `comparatorsList.xml`. You do not need to create this file in the same package as the Java comparator class, but for packaging purposes, create it in the same working folder.

To Define Parameter Validations

1. Complete Step 2: Register the Comparator in the Comparators List.
2. Create a Java class named the same name as the Java class that defines the comparator with "ParamsValidator" appended.

   For example, if the comparator is defined by a class named ExactComparator, the parameter validation class would be `ExactComparatorParamsValidator`.
3. In this class, implement:
   `com.sun.mdm.matcher.comparators.validator.ParametersValidator`

   The method contained in this class is described below.
4. Continue to Step 4: Define Data Source Handling (Optional).

validateComparatorsParameters

- Description: The `ParametersValidator` class contains one method, validateComparatorsParameters, that allows you to validate parameter types,

ranges, and other properties. For logging purposes, you can use net.java.hulp.i18n, which is used within matcher.jar, or you can use your own logger.

- Syntax: `void validateComparatorsParameters(Map<String, Object> params)`
- Parameters:

Table 6–4 validateComparatorsParameters Parameters

Parameter	Type	Description
params	Map	A list of parameters to validate.

- Return Value: None.
- Throws: `MatcherException`

Step 4: Define Data Source Handling (Optional)

If your custom comparators use external data sources to provide additional information for matching weight calculations, you need to create a Java class that lets you load the file to memory or have real-time access to the data file content. You can also define validations to perform. You do not need to create this file in the same package as the Java comparator class, but for packaging purposes, create it in the same working folder.

You need to perform this step if you defined lines similar to the following in `comparatorsList.xml`:

```
<data-sources>
   <datasource description="Serial numbers" type="java.io.File" />
</data-sources>
```

To Define Data Source Handling

1. Complete Step 3: Define Parameter Validations (Optional).
2. Create a Java class named the same name as the Java class that defines the comparator with "SourcesHandler" appended.

   For example, if the comparator is defined by a class named ExactComparator, the parameter validation class would be `ExactComparatorSourcesHandler`.
3. In this class, implement `com.sun.mdm.matcher.comparators.validator.DataSourcesHandler`.

   The method in this class is described below.
4. Continue to Step 5: Define Curve Adjustment or Linear Fitting (Optional).

handleComparatorsDataSources

- Description: The `DataSourcesHandler` class contains one method, `handleComparatorsDataSources`, that allows you to define properties for the data source. This method takes one parameter that is a DataSourcesProperties object. This class and its methods are described in DataSourcesProperties Class.
- Syntax: `Object handleComparatorsDataSources(DataSourcesProperties dataSources)`
- Parameters:

Table 6–5 handleComparatorsDataSources Parameters

Parameter	Type	Description
dataSources	DataSourceProperties	A list of properties for the data handler (see DataSourcesProperties Class).

- Return Value: Object
- Throws:
  - `MatcherException`
  - `IOException`

DataSourcesProperties Class

The `DataSourcesProcerties` interface is used as a parameter to the `handleComparatorsDataSources` described in Step 4: Define Data Source Handling (Optional). The methods in the class are listed and described below.

getDataSourcesList

- Description: The `getDataSourcesList` returns the comparator's list of associated data source paths.
- Syntax: `List getDataSourcesList(String codeName)`
- Parameters:

Table 6–6 getDataSourcesList Parameters

Parameter	Type	Description
codeName	string	The name of the comparator. The name is defined in `comparatorsList.xml` in the name attribute of the code element. In the example below, the comparator's code name is "Exact". `<code description="New exact comparator" name="Exact" />`

- Return Value: A list of paths and filenames as specified in comparatorsList.xml.
- Throws: None.

isDataSourceLoaded

- Description: The `isDataSourceLoaded` method checks whether a specific file has already been loaded or opened.
- Syntax: `boolean isDataSourceLoaded(String sourcePath)`
- Parameters:

Table 6–7 isDataSourceLoaded Parameters

Parameter	Type	Description
sourcePath	string	The path and filename of the file to check.

- Return Value: A boolean indicator of whether the specified file has already been loaded or opened.
- Throws: None.

setDataSourceLoaded

- Description: The `setDataSourceLoaded` method sets the loading status of a data source.
- Syntax: `void setDataSourceLoaded(String sourcePath, boolean status)`
- Parameters:

Table 6–8 setDataSourceLoaded Parameters

Parameter	Type	Description
sourcePath	string	The path and filename of the file.
status	boolean	The load status of the file. Specify **true** if the file is loaded; otherwise specify **false**.

- Return Value: None.
- Throws: None.

getDataSourceObject

- Description: The `getDataSourceObject` method returns the file located at the specified source path.
- Syntax: `Object getDataSourceObject(String sourcePath)`
- Parameters:

Table 6–9 getDataSourceObject Parameters

Parameter	Type	Description
sourcePath	string	The path and filename of the file you want to load.

- Return Value: An object containing the data source information.
- Throws: None.

Step 5: Define Curve Adjustment or Linear Fitting (Optional)

If your custom comparators use curve adjustment or linear fitting to adjust matching weight calculations, you need to create a Java class that defines the curve. You do not need to create this file in the same package as the Java comparator class, but for packaging purposes, create it in the same working folder.

You need to perform this step if you defined the following line in `comparatorsList.xml` for the comparator:

```
<curve-adjust status="true" />
```

To Define Curve Adjustment or Linear Fitting

1. Complete Step 4: Define Data Source Handling (Optional).
2. Create a Java class named the same name as the Java class that defines the comparator with "CurveAdjustor" appended.

   For example, if the comparator is defined by a class named ExactComparator, the parameter validation class would be `ExactComparatorCurveAdjustor`.
3. In this class, implement `com.sun.mdm.matcher.configurator.CurveAdjustor`.

   The method in this class is described below.

4. Continue to Step 6: Compile and Package the Comparator.

processCurveAdjustment

- Description: The `processCurveAdjustment` method provides handling for curve adjustment within a specific match comparator.
- Syntax: `double[] processCurveAdjustment(String compar, double[] cap)`
- Parameters:

Table 6–10 processCurveAdjustment Parameters

Parameter	Type	Description
compar	string	The name of the comparator, as defined in the name attribute of the code element for the comparator.
cap	double[]	An array of values that define the curve adjustment.

- Return Value: An array of curve adjustment values.
- Throws: `MatcherException`

Step 6: Compile and Package the Comparator

Before you perform these steps, make sure you have completed Step 1: Create the Custom Comparator Java Class through Step 5: Define Curve Adjustment or Linear Fitting (Optional).

When you are finished defining all the Java classes for the comparators and have registered each comparator in your comparators list file, you can compile the Java code and package the files into a ZIP file that you can then import into a master person index application. Compile the classes using the compiler of your choice.

To package the files, create a temporary directory and copy the comparators list file to the directory. Copy all the class folders and files to the same directory. The top level of the temporary directory should include `comparatorsList.xml` and a com folder (which contains all the Java classes). Create a ZIP file of the directory. For more information about the ZIP package, see About the Comparator Package.

After you compile and package the comparator, continue to Step 7: Import the Comparator Package Into Oracle Healthcare Master Person Index.

Step 7: Import the Comparator Package Into Oracle Healthcare Master Person Index

You need to import the new comparators into NetBeans to make them available to all master person index applications or only the current application.

To Import a Comparison Function

1. Launch **NetBeans IDE**, and open the master person index project that will use the new comparators.
2. In the Projects window, expand the main master person index project.
3. Right-click **Match Engine**, and select **Import Comparator Plug-in**.
4. In the dialog box that appears, navigate to the location of the plug-in ZIP file.
5. Select the file containing the plug-in, and then click **Open**.
6. Do one of the following:

- To import the plug-in and make it available to all future master person index applications, click **Yes**.
- To import the plug-in and make it only available to the current master person index application, click **No.**

The contents of the ZIP file are imported into the Match Engine node and the new comparators are added to the list of comparator definitions in `comparatorsList.xml`.

7. In the Match Engine node, navigate to the `/lib` folder that was added and verify that all of the required files are there.
8. Open `comparatorsList.xml` and verify the new comparator definitions are included.

Step 8: Configure the Comparator in the Match Configuration File

After you import custom comparators, you need to add them to the match configuration file (`matchConfigFile.cfg`) and define the matching configuration. This makes the comparator available for use in the master person index match string. For information about this file, see Understanding the OHMPI Match Engine Match Configuration File. For instructions on modifying the file, see the *Oracle Healthcare Master Person Index Configuration Guide*.