SUN SEEBEYOND

IMPLEMENTING THE SUN SEEBEYOND MATCH ENGINE WITH eVIEW™ STUDIO

Release 5.1.3

Part Number: 820-0936-10

Version 20070425124355

Contents

Appendix B

List of Tables

Chapter 1

Introduction

This guide explains how to implement the Sun SeeBeyond Match Engine with applications created by the Sun SeeBeyond eView™ Studio, referred to as eView Studio throughout this guide. This chapter provides an overview of this guide and the conventions used throughout, as well as a list of supporting documents and information about using this guide.

What's in This Chapter

1.1 About the Sun SeeBeyond Match Engine

The Sun SeeBeyond Match Engine (SBME) provides data parsing, data standardization, phonetic encoding, and record matching capabilities for master index applications created by the Sun SeeBeyond eView™ Studio (eView Studio). 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 or parsed. Once the data is conditioned, the match engine determines a match weight for each field defined for matching. The match weight is based on your configuration of the match engine and the fields on which matching is performed. The composite weight (the sum of weights generated for all match fields in the records) indicates how closely two records match.

1.2 What's New in This Release

For this release, no changes were made to the match engine. The names of the comparison functions (or *comparators*) in the match configuration file were changed to be more descriptive of the comparator types (this refers to the names used in this document and not the name in the match configuration file itself).

1.3 About This Document

This guide provides comprehensive information about working with the components of the Sun SBME and implementing the match engine with eView Studio. As a component of the Java Composite Application Platform Suite, eView Studio helps you integrate information from disparate systems throughout your organization, using a matching algorithm to identify data.

This guide includes complete descriptions of the components of the Sun SBME along with information about customizing each component. It also describes how to customize the eView Studio configuration files to define standardization and matching fields for the Sun SBME. This guide is designed to be used in conjunction with the *Sun SeeBeyond eView Studio User's Guide* and the *Sun SeeBeyond eView Studio Configuration Guide*.

1.3.1 What's in This Document

This guide is divided into the following chapters and appendixes that cover the topics shown below.

- **Chapter 1 "Introduction"** gives a general preview of this document—its purpose, scope, and organization—and provides sources of additional information.
- **Chapter 2 "The Sun SeeBeyond Match Engine"** gives an overview of the Sun SBME, the underlying matching algorithm, and the components of the match engine.
- **Chapter 3 "Standardization Configuration Files"** gives an overview of the standardization configuration files and their purpose.
- **Chapter 4 "Matching Configuration Files"** gives an overview of the matching configuration files and a reference of the Sun SBME comparison functions.
- **Chapter 5 "eView Studio and the Sun SeeBeyond Match Engine"** describes how the Sun SBME works with an eView Studio master index.
- **Chapter 6 "Person Data Type Configuration"**gives information and instructions for customizing the standardization and matching configuration files for person data, and also gives instructions for customizing the Match Field file.
- **Chapter 7 "Address Data Type Configuration"**gives information and instructions for customizing the standardization and matching configuration files for address fields, and also gives instructions for customizing the Match Field file.
- **Chapter 8 "Business Names Data Type Configuration"**gives information and instructions for customizing the standardization and matching configuration files for business name fields, and also gives instructions for customizing the Match Field file.
- **Appendix A "Fine-tuning Weights and Thresholds"** describes the process of fine-tuning the matching logic and weight thresholds.
- **Appendix B "Match Configuration Comparison Functions"** lists and describes the comparison functions you can use with the Sun SBME.

1.3.2 Scope

This guide provides background information and instructions for implementing the Sun SBME with an eView Studio master index. It includes descriptions of all components and the default configuration, and also provides instructions for customizing the eView Studio configuration files for the match engine.

This guide does not include information or instructions for installing or configuring eView Studio. These topics are covered in the appropriate user guide (for more information, see **"Related Documents" on page 13**).

1.3.3 Intended Audience

Any user who configures the eView Studio Matching Service (that is, the eView Studio Project's Match Field file) or creates or customizes the match engine configuration files should read this book. A thorough knowledge of eView Studio is not needed to understand this guide, but familiarity with the eView Studio configuration files (especially the Match Field file) is recommended. A general understanding of basic standardization and matching logic is helpful. It is presumed that the reader of this guide is familiar with the type of data being stored in the master index and the processing requirements for that data.

1.3.4 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

Text Convention	Used For	Examples
Bold	Names of buttons, files, icons, parameters, variables, methods, menus, and objects	▪ Click **OK**. ▪ On the **File** menu, click **Exit**. ▪ Select the **eGate.sar** file.
`Monospaced`	Command line arguments, code samples; variables are shown in ***bold italic***	`java -jar` ***filename*** `.jar`
Blue bold	Hypertext links within document	See **Text Conventions** on page 12
Blue underlined	Hypertext links for Web addresses (URLs) or email addresses	http://www.sun.com

1.3.5 Screenshots

Depending on what products you have installed, and how they are configured, the screenshots in this document might differ from what you see on your system.

1.4 Related Documents

Sun has developed a suite of user's guides and related publications that are distributed in an electronic library. The following documents might provide information useful in implementing the Sun SBME.

- *Sun SeeBeyond eView Studio User's Guide*
- *Sun SeeBeyond eView Studio Configuration Guide*
- *Sun SeeBeyond eView Studio Reference Guide*

1.5 Sun Microsystems, Inc. Web Site

The Sun Microsystems web site is your best source for up-to-the-minute product news and technical support information. The site's URL is:

http://www.sun.com

1.6 Documentation Feedback

We appreciate your feedback. Please send any comments or suggestions regarding this document to:

CAPS_docsfeedback@sun.com

Chapter 2

The Sun SeeBeyond Match Engine

The Sun SeeBeyond Match Engine (SBME) is the standard match engine designed to work with the master indexes created by eView Studio. It is highly configurable in the eView Studio environment and can be used to match on various types of data.

This chapter provides information about the configurable components of the match engine and how the Sun SBME standardizes and matches data.

What's in This Chapter

2.1 About the Matching Algorithm

The Sun SBME compares records containing similar data types by calculating how closely 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. The formula used to determine the matching weight is based on either matching and unmatching probabilities or on agreement and disagreement weight ranges.

The Sun SBME is also designed to standardize freeform text fields, such as street address fields or business names. This allows the match engine to generate a more accurate weight for freeform data.

2.2 Standardization and Matching

The Sun SBME matching algorithm uses a proven methodology to process and weight records in the master index database. By providing both standardization and matching capabilities, the match engine allows you to 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.

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 the match engine has performed these steps, survivorship is determined by the master index, based on how the overall matching weight compares to the duplicate and match thresholds of the master index. These thresholds are configured for the eView Studio Manager Service in the Threshold file.

2.3 Data Types

You can standardize and match on different types of data with the Sun SBME. In its default implementation with eView Studio, the match engine supports data standardization and matching on the three primary types of data listed below.

- Person Information (described in **Chapter 6 "Person Data Type Configuration"**)
- Street Addresses (described in **Chapter 7 "Address Data Type Configuration"**)
- Business Names (described in **Chapter 8 "Business Names Data Type Configuration"**)

In addition, the Sun SBME provides comparison functions for matching on various types of fields contained within the primary data types, such as numbers, dates, Social Security Numbers, single characters, and so on.

When processing person information, the match engine assumes that each match field is stored in a separate field. For street address and business name processing, eView Studio is configured to parse freeform text fields for searching and matching. Each data type requires specific customizations to the Match Field file in the eView Studio Project.

2.4 How it Works

The Sun SBME compares two records and returns a match weight indicating the likelihood of a match between the two records. The three primary components of the Sun SBME are the configuration files, the standardization engine, and the match engine.

- **Configuration Files**
  The Sun SBME includes several sets of files that define standardization and matching logic for all supported data types. One set is common to all national domains, and one additional set is provided for the following national domains: Australia, France, Great Britain, and the United States. You can customize these files to adapt the standardization and matching logic to your specific needs.
- **Standardization Engine**
  Standardization involves converting non-standard data into a standardized form for more accurate and efficient processing. Standardization consists of any one or more of the following actions:
  - **Parsing** - separating a free-text field into its individual components, such as street address information or a business name.
  - **Normalization** - changing the value of a field to a standard version, such as changing a nickname to a common name.
  - **Phonetic Encoding** - changing the value of a field to its phonetic version. The field to be converted can be the original field, a parsed field, a normalized field, or a parsed and normalized field.

  Using the person data type, for example, first names such as "Bill" and "Will" are normalized to "William", which is then phonetically converted. Using the street address data type, street addresses are parsed into their component parts, such as house numbers, street names, and so on. The street name is then phonetically converted. Standardization logic is defined in the standardization engine configuration files and in the **StandardizationConfig** section of the eView Studio Match Field file, and is performed prior to assigning match weights.
- **Match Engine**
  Matching involves comparing two standardized records and returning a weight that indicates the likelihood of a match between the two records. A higher weight indicates a greater likelihood of a match. Matching criteria and logic are defined in the match engine configuration files. The data fields that are sent to the Sun SBME for matching, known as the *match string*, are defined in the **MatchingConfig** section of the eView Studio Match Field file. The match engine configuration files define how the match string is standardized and which matching rules to use to process each match field.

2.5 Matching Weight Formulation

The Sun SBME 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 Sun SBME 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.

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, a disagreement weight is subtracted from the composite match weight. 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 Weight Ranges

Defining agreement and disagreement weight ranges is a more direct way to implement m-probabilities and u-probabilities. Like probabilities, the maximum agreement and minimum disagreement weights you define for each field allow you to define 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 Sun SBME assigns a matching weight to each field that falls between the agreement and disagreement weights specified for the field. This provides a more convenient and intuitive representation of conditional probabilities.

Using the SSN and gender field example above, the SSN field would be 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”. Agreement and disagreement weights are expressed as double values and can have up to 16 decimal points.

Chapter 3

Standardization Configuration Files

The standardization configuration files for the Sun SeeBeyond Match Engine must follow certain rules for formatting and interdependencies. This chapter provides an overview of the types of configuration files provided for standardization, the architecture of those files, and formatting descriptions.

What's in This Chapter

3.1 About Standardization Configuration Files

The standardization configuration files define additional logic used by the Sun SBME to standardize specific data types. This logic helps define how fields in incoming records are parsed, standardized, and classified for processing. Standardization files include data patterns files, category files, clues files, key type tables, constants files, and reference files.

The standardization configuration files are stored in the eView Studio Project and appear as nodes in the **Standardization Engine** node of the Project. Several standardization files are common to all implementations of the Sun SBME, but each national domain uses a subset of unique files. The common files are listed directly under the **Standardization Engine** node of the eView Studio Project; the files unique to each national domain are listed in individual sub-folders under the **Standardization Engine** node.

3.1.1 Standardization Configuration File Types

Several different types of configuration files are included with the Sun SBME, each providing specific information to help the engine standardize and match data according to requirements. Several of these files are common to all supported nationalities, but a small subset is specific to each.

- **Category Files**
  The Sun SBME uses category files when processing person or business names. These files list common values for certain types of data, such as titles, suffixes, and nicknames for person names or industries and organizations for business names. Category files also define standardized versions of each term or classify the terms

into different categories, and some files perform both functions. When processing address files, category files named "clues files" are used.

- **Clues Files**
  The Sun SBME uses clues files when processing address data types. These files list general terms used in street address fields, define standardized versions of each term, and classify the terms into various component types using predefined address tokens. These files are used by the standardization engine to determine how to parse a street address into its various components. Clues files provide clues in the form of tokens to help the engine recognize the component type of certain values in the input fields.
- **Constants Files**
  The Sun SBME refers to constants files for information about the standardization files, such as the maximum length of the files. For the address data type, the constants file also describes input and output field lengths.
- **Patterns Files**
  The patterns files specify how incoming data should be interpreted for standardization based on the format, or pattern, of the data. These files are used only for processing data contained in freeform text fields that must be parsed prior to matching (such as street address fields or business names). Patterns files list possible input data patterns, which are encoded in the form of tokens. Each token signifies a specific component of the freeform text field. For example, in a street address field, the house number is identified by one token, the street name by another, and so on. Patterns files also define the format of the output fields for each input pattern.
- **Key Type Files**
  For business name processing, the Sun SBME refers to a number of key type files for processing information. These files generally define standard versions of terms commonly found in business names and some classify these terms into various components or industries. These files are used by the standardization engine to determine how to parse a business name into its different components and to recognize the component type of certain values in the input fields.
- **Reference Files**
  Reference files define general terms that appear in input fields for each data type. Some reference files define terms to ignore, and some define terms that indicate the business name is continuing. For example, in business name processing "and" is defined as a joining term. This helps the standardization engine to recognize that the primary business name in "Martin and Sons, Inc." is "Martin and Sons" instead of just "Martin". Reference files can also define characters to be ignored by the standardization engine.

3.2 Internationalization

The Sun SBME supports addresses and names originating from Australia, France, Great Britain, and the United States. Each national domain uses a set of common standardization files and a smaller set of unique, domain-specific files to account for

international differences in address formats, names, and so on. These files are described in detail in later chapters. You can process with your data with using the standardization files for a single national domain or you can use multiple domains depending on how the Match Field file is configured.

Chapter 4

Matching Configuration Files

The matching configuration files for the Sun SeeBeyond Match Engine must follow certain rules for formatting and interdependencies. This chapter provides an overview of the two matching configuration files provided, the architecture of those files, and formatting descriptions. It also provides a reference of comparison functions used in the match configuration file.

What's in This Chapter

4.1 About Matching Configuration Files

The matching configuration files define how the Sun SBME processes records to assign matching probability weights, allowing the master index to identify matches, potential duplicates, and non-matches. These files consist of two configurable files, the match configuration file and the match constants file. Together these files define additional logic for the Sun SBME to use when determining the matching probability between two records. A third file, the internal match constants file, is read-only and used internally by the match engine. It defines each comparison function and the comparison options.

The matching configuration files are very flexible, allowing you to customize the matching logic according to the type of data stored in the master index and for the record matching requirements of your business. The matching configuration files are stored in the eView Studio Project and appear as nodes in the **Match Engine** node of the Project. The Sun SBME typically standardizes the data prior to matching, so the match process is performed against the standardized data.

4.2 The Match Configuration File

The match configuration file, **matchConfigFile.cfg**, contains the matching logic for each field on which matching is performed. This file handles 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 Sun SBME 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.

4.2.1 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.

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   15  -5
LastName                15  0   ul    0.99  0.001   15  -5
String                  25  0   ua    0.99  0.001   10  -5
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
Numeric                 15  0   n     0.99  0.001   10  -10 y 8
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

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.

Matching Rules

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. The syntax for this section is:

```
match-type size null-field function m-prob u-prob agreement-weight
disagreement-weight parameters
```

Table 2 describes each element in a match field row.

Table 2 Match Configuration File Columns

Column Number	Column Name	Description
1	match-type	A value that indicates to the Sun SBME how each field should be weighted. Each field included in the match string (the **MatchingConfig** section of the Match Field file) 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. ▪ **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 this file or calculated using a logarithmic formula based on the m and u-probabilities (depending on the probability type).*

Table 2 Match Configuration File Columns

Column Number	Column Name	Description
4	function	The type of comparison to perform when weighting the field. For information about the available comparison functions, see **Match Configuration Comparison Functions** on page 100.
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-weight	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-weight	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	parameters	The parameters 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 **Match Configuration Comparison Functions** on page 100.

4.2.2 Matching Comparison Functions

Match field comparison functions 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 Sun SBME should match the fields in the match string. The Sun SBME also provides several options to use with each function. Table 3 summarizes each comparison function. A

complete reference of the comparison functions and their parameters is included in **Appendix B "Match Configuration Comparison Functions"**.

Table 3 Comparison Functions

Comparison Function	Name	Description
b1	Bigram String Comparator	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 String Comparator	Similar to the standard Bigram comparison function (b1), but allows for character transpositions.
u	Generic String Comparator	Based on the *Jaro* algorithm, this function compares two strings taking into account uncertainty factors, such as string length, transpositions, and characters in common.
ua	Advanced Generic String Comparator	Based on the *Jaro* algorithm with variants of Winkler/ Lynch and McLaughlin, this function is similar to the generic string comparator (u), but increases the agreement weight if the initial characters of each string are exact matches. This comparison function takes into account key punch and visual memory errors.
uf	Simplified String Comparator - FirstName	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	Simplified String Comparator - LastName	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	Simplified String Comparator - HouseNumber	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	Simplified String Comparator	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 generic string comparator ("u"), this function handles diacritical marks. This function also improves processing speed.
usu	Language-specific String Comparator	A custom string comparator similar to the "us" comparator with the exception that it is based in Unicode to support multiple languages and alphabets. This comparator takes one parameter indicating the language to use.
c	Exact char-by-char Comparator	Compares string fields character by character. Each character must match in order for an agreement weight to be assigned.

Table 3 Comparison Functions

Comparison Function	Name	Description
n	Generic Number Comparator	Compares numeric 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 it 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.
nI	Integer Comparator	Compares integer fields using a relative distance comparison. This comparison function is based on the generic number comparator (n), and accepts the same parameters.
nR	Real Number Comparator	Compares fields containing real numbers using a relative distance comparison. This comparison function is based on the generic number comparator (n), and accepts the same parameters.
nS	Alpha-Numeric 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 - Year only	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 - Month-Year	Compares the month and year using a relative distance as described above for the year comparison function (dY).
dD	Date Comparator - Day-Month-Year	Compares the day, month, and year using a relative distance as described above for the year comparison function (dY).
dH	Date Comparator - Hour-Day-Month-Year	Compares the hour, day, month, and year using a relative distance as described above for the year comparison function (dY).
dm	Date Comparator - Min-Hour-Day-Month-Year	Compares the minute, hour, day, month, and year using a relative distance as described above for the year comparison function (dY).

Table 3 Comparison Functions

Comparison Function	Name	Description
ds	Date Comparator - Sec-Min-Hour-Day-Month-Year	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.

4.3 The Match Constants File

The match constants file, **matchConstants.cfg**, defines certain configurable constants used by the match engine. This file includes four parameters, but currently only the first parameter, **nFields**, is used. This parameter defines the maximum number of fields being used for matching. This must be equal to or greater than the number of fields defined in the **match-columns** element of the eView Studio Match Field file. The match constants file defines the following constants for the match engine.

nFields

This constant defines the maximum number of different matching fields. You can enter any integer, but this number must be equal to or greater than the number of fields defined in the match-columns elements in the eView Studio Match Field file.

maxFreqTableSize

This constant is only used when frequency tables are used. This is not currently available and this constant is ignored.

maxNumberTables

This constant is only used when frequency tables are used. This is not currently available and this constant is ignored.

mcls

This constant is only used when the generic-type frequency tables are used. This is not currently available and this constant is ignored.

Chapter 5

eView Studio and the Sun SeeBeyond Match Engine

Implementing the Sun SeeBeyond Match Engine with an eView Studio master index requires some customization to the Match Field file in the eView Studio Project. You can also customize the Sun SBME configuration files to better suit your data standardization and matching requirements.

This chapter provides information about the required customizations, and how the Match Field file corresponds to the configuration files.

What's in This Chapter

5.1 The Sun SeeBeyond Match Engine and eView Studio

eView Studio master index applications use the Sun SBME specifically for standardization and probabilistic weighting, while the master index determines survivorship. This process relies on the logic specified in the configuration files of the eView Studio Project and of the Sun SBME.

5.1.1 Searching and Matching in eView Studio

When a new record is passed to the master index database, the index selects a subset of possible matches from the database. The master index then uses the Sun SBME's matching algorithm to assign a matching probability weight for each record in this subset (known as the *candidate selection pool*). To create the candidate selection pool, the master index makes a series of query passes of the existing data, searching for matches on specific combinations of data. These combinations are defined by the blocking query, which is defined in the Candidate Select file and specified in the Threshold file.

Matching is performed on the fields included in the match string in the Match Field file. Each field is assigned a matching weight. The weights for each field are summed to determine the matching probability weight for the entire record (known as the *composite weight*). Before matching on some fields, such as the first name, the index

might standardize the field based on information in the standardization files. You can customize how each field is weighted in the match configuration file.

5.1.2 The Standardization and Matching Process

The standardization and matching processes use logic that is defined by a combination of Sun SBME configuration files and eView Studio configuration files. During the standardization and match processes, the following occurs.

1. The Sun SBME receives an incoming record.
2. The Sun SBME standardizes the fields specified for parsing, normalization, and phonetic encoding. These fields are defined in the **StandardizationConfig** section of the Match Field file and the rules for standardization are defined in the Sun SBME standardization configuration files.
3. eView Studio queries the database for a candidate selection pool (records that are possible matches) using the specified blocking query. If the blocking query uses standardized or phonetic fields, the criteria values are obtained from the database.
4. For each possible match, eView Studio creates a match string (based on the match columns in **MatchingConfig**) and sends the string to the Sun SBME.
5. The Sun SBME checks the incoming record against each possible match, producing a matching weight. Matching is performed using the weighting rules defined in the match configuration file.

5.1.3 The Match String

The data string that is passed to the Sun SBME for match processing is called the *match string* and is defined in the **MatchingConfig** section of the Match Field file. The Sun SBME 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 Sun SBME from the records in the candidate selection pool. Finally, the Sun SBME configuration files define how the match string is processed.

The Sun SBME 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.

5.1.4 Field Identifiers

The Sun SBME breaks down fields into various components. For example, it breaks addresses into floor number, street number, street name, street direction, and so on. Some of these components are similar and are typically stored in the same field in the

database. In the default configuration, for example, when the standardization engine finds a house number, rural route number, or PO box number, the value is stored in the HouseNumber database field. You can customize this as needed, as long as any field you specify to store a component is also included in the defined object structure.

The Sun SBME uses field identifiers to determine how to process fields that are defined for normalization or parsing. The IDs are defined internally in the match engine and are referenced in the Match Field file. The field IDs you specify for each field in the Match Field file determine how that field is processed by the standardization engine. The field IDs for person names determine how each name is normalized. The field IDs for business names specify which business type key file to use for standardization. The field IDs for addresses determine which database fields store each field component and how each component is standardized.

Table 4 lists each field component generated by the Sun SBME along with their corresponding field IDs. You can only specify the predefined field IDs that are listed in this table.

Table 4 Standardization Field Identifiers

Field ID	Description
Person Name Standardization Field Identifiers	
FirstName	Specifies a first name field for normalization.
LastName	Specifies a last name field for normalization.
Address Standardization Field Identifiers	
HouseNumber	Specifies the parsed house number from a standardized address field. By default, this is stored in the **<field_name>_HouseNo** field (or the **HouseNumber** field for eIndex SPV).
RuralRouteIdentif	Specifies the parsed rural route identifier from a standardized address field. By default, this is stored in the **<field_name>_HouseNo** field (or the **HouseNumber** field for eIndex SPV).
BoxIdentif	Specifies the parsed PO box number from a standardized address field. By default, this is stored in the **<field_name>_HouseNo** field (or the **HouseNumber** field for eIndex SPV).
MatchStreetName	Specifies the parsed and standardized street name from a standardized address field and is used internally by the match engine. If you want to store the standardized street name in the database (recommended), map this field to the street name field in the database. By default, this is stored in the **<field_name>_StName** field (or the **StreetName** field for eIndex SPV).

Table 4 Standardization Field Identifiers

Field ID	Description
OrigStreetName	Specifies the parsed street name from an address field. If you want to store the original street name in the database, map this field to the street name field in the database. This address component is not included in the default standardization structure, but you can add it if needed.
RuralRouteDescript	Specifies the parsed rural route description from a standardized address field. By default, this is stored in the **<field_name>_StName** field (or the **StreetName** field for eIndex SPV).
BoxDescript	Specifies the PO box type from a standardized address field. By default, this is stored in the **<field_name>_StName** field (or the **StreetName** field for eIndex SPV).
PropDesPrefDirection	Specifies the parsed property direction from a standardized address field. This field ID handles cases where the direction is a prefix to the property description. By default, this is stored in the **<field_name>_StDir** field (or the **StreetDir** field for eIndex SPV).
PropDesSufDirection	Specifies the parsed property direction from a standardized address field. This field ID handles cases where the direction is a suffix to the property description. By default, this is stored in the **<field_name>_StDir** field (or the **StreetDir** field for eIndex SPV).
StreetNamePrefDirection	Specifies the parsed street direction from a standardized address field. This field ID handles cases where the direction is a prefix to the street name. By default, this is stored in the **<field_name>_StDir** field (or the **StreetDir** field for eIndex SPV).
StreetNameSufDirection	Specifies the parsed street direction from a standardized address field. This field ID handles cases where the direction is a suffix to the street name. By default, this is stored in the **<field_name>_StDir** field (or the **StreetDir** field for eIndex SPV).
StreetNameSufType	Specifies the parsed street type from a standardized address field. This field ID handles cases where the street type is a suffix to the street name. By default, this is stored in the **<field_name>_StType** field (or the **StreetType** field for eIndex SPV).

Table 4 Standardization Field Identifiers

Field ID	Description
StreetNamePrefType	Specifies the parsed street type from a standardized address field. This field ID handles cases where the street type is a prefix to the street name. By default, this is stored in the **<field_name>_StType** field (or the **StreetType** field for eIndex SPV).
PropDesSufType	Specifies the parsed property type from a standardized address field. This field ID handles cases where the street type is a suffix to the property description. By default, this is stored in the **<field_name>_StType** field (or the **StreetType** field for eIndex SPV).
PropDesPrefType	Specifies the parsed property type from a standardized address field. This field ID handles cases where the street type is a prefix to the property description. By default, this is stored in the **<field_name>_StType** field (or the **StreetType** field for eIndex SPV).
HouseNumPrefix	Specifies the parsed house number prefix from a standardized address field (such as the "A" in "A 1587 4th Street"). This address component is not included in the default standardization structure, but you can add it if needed.
SecondHouseNumberPrefix	Specifies the parsed *second* house number prefix from a standardized address field (such as "25" in "25 319 10th Ave."). This address component is not included in the default standardization structure, but you can add it if needed.
SecondHouseNumber	Specifies the parsed *second* house number prefix from a standardized address field. This address component is not included in the default standardization structure, but you can add it if needed.
HouseNumSuffix	Specifies the parsed house number suffix from a standardized address field. This address component is not included in the default standardization structure, but you can add it if needed.
OrigSecondStreetName	Specifies the parsed *second* street name from a standardized address field (for example, an address might include a cross-street or a thoroughfare and dependent thoroughfare). This address component is not included in the default standardization structure, but you can add it if needed.

Table 4 Standardization Field Identifiers

Field ID	Description
SecondStreetNameSufDirection	Specifies the parsed *second* street direction from a standardized address field. This address component is not included in the default standardization structure, but you can add it if needed.
SecondStreetNameSufType	Specifies the parsed *second* street type from a standardized address field.This address component is not included in the default standardization structure, but you can add it if needed.
StreetNameExtensionIndex	Specifies the parsed street name extension from a standardized address field. This address component is not included in the default standardization structure, but you can add it if needed.
WithinStructDescript	Specifies the parsed internal descriptor (such as “Floor”) from a standardized address field. This address component is not included in the default standardization structure, but you can add it if needed.
WithinStructIdentif	Specifies the parsed internal identifier (such as a floor number) from a standardized address field. This address component is not included in the default standardization structure, but you can add it if needed.
OrigPropertyName	Specifies the parsed original property name (such as the name of a complex or business park) from a standardized address field. This address component is not included in the default standardization structure, but you can add it if needed.
MatchPropertyName	Specifies the parsed match property name from a standardized address field and is used internally by the match engine for blocking and phonetic encoding. This address component is not included in the default standardization structure, but you can add it if needed.
CenterDescript	Specifies the parsed structure description from a standardized address field. This address component is not included in the default standardization structure, but you can add it if needed.

Table 4 Standardization Field Identifiers

Field ID	Description
CenterIdentif	Specifies the parsed structure identifier from a standardized address field. This address component is not included in the default standardization structure, but you can add it if needed.
ExtraInfo	Specifies any extra information that was not included in any of the other parsed components. This address component is not included in the default standardization structure, but you can add it if needed.
Business Name Standardization Field Identifiers	
PrimaryName	Specifies the field containing the parsed name in a freeform text business name field.
OrgTypeKeyword	Specifies the field containing the parsed organization type in a freeform text business name field.
AssocTypeKeyword	Specifies the field containing the parsed association type in a freeform text business name field.
IndustrySectorList	Specifies the field containing the parsed industry sector in a freeform text business name field.
IndustryTypeKeyword	Specifies the field containing the parsed industry type in a freeform text business name field (industry type is a subset of the sector).
AliasList	Specifies the field containing the parsed alias in a freeform text business name field.
Url	Specifies the field containing the parsed URL in a freeform text business name field.

5.1.5 Match and Standardization Types

The Match Field file in the eView Studio Project uses indicators to reference the type of matching and standardization to perform on each field. You must specify one of these indicators, called *match types* and *standardization types,* for the fields you define for standardization or matching. The match types correspond to the match types listed in the first column of the match configuration file (**matchConfigFile.cfg**). The standardization types are defined internally in the match engine. The Sun SBME uses these types to determine how to process each field.

Table 5 lists the default standardization types; Table 6 lists the default match types. You can modify the match type names, but not the standardization type names. For more information about match and standardization types, see "Match and Standardization Types" in the *Sun SeeBeyond eView Studio User's Guide*. Note that the match types you

can specify in the Match Field file (listed in Table 6) are not the same values you specify for the **Match Type** field drop-down list in the eView Studio Wizard.

Table 5 Standardization Types

This indicator …	processes this data type …
Address	Freeform street address fields.
PersonName	Pre-parsed name fields (including any first, middle, last, or alias names).
BusinessName	Freeform business names.

The standardization types listed above correspond to the three categories of match types listed below. You can also specify miscellaneous match types, which do not correspond to any standardization types.

Table 6 Match Types

This indicator …	processes this data type …
Business Name Match Types	
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.
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.
Address Match Types	
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.
Person Name Match Types	
FirstName	A first name field, including middle name, alias first name, and alias middle name fields.

Table 6 Match Types

This indicator …	processes this data type …
LastName	A last name field, including alias last name fields.
Date Match Types	
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.
Miscellaneous Match Types	
String	A generic string field.
Numeric	A numeric field.
Integer	A field containing integers.
Real	A field containing real numbers.
SSN	A field containing a social security number.
Char	A field containing a single character.
pro	Any field on which you want the Sun SBME to use prorated weights.
Exac	Any field you want the Sun SBME to match character for character.

5.1.6 About Configuration File Modifications

The Sun SBME configuration files are designed to perform very specific functions in the standardization and match processes. These files should only be modified by personnel with an understanding of the Sun SBME and an understanding of the data integrity requirements of your organization. Sun recommends that any modifications to both the eView Studio configuration files and the Sun SBME configuration files be made while the master index is in the pre-production stages. Modifying the files after the master index has moved into production might cause variances in matching weights and data processing.

The most common modifications to the Sun SBME configuration files are generally in the match configuration file, where you can fine-tune the weighting process. This file defines probabilities used by the algorithm to determine a matching probability weight for each match field. You can use the match comparison functions provided by the Sun SBME to fine-tune the matching logic in this file. Another common modification is inserting additional names or terms into category files, such as the first name category file (**personFirstName*.dat**).

Depending on your data requirements, you might need to modify additional standardization files. Some of the patterns files (most notably the address patterns files) are very complex and should only be modified by personnel who thoroughly understand the defined patterns and tokens. If you modify standardization files, make sure you modify them for each national domain specified in the Match Field file.

5.2 Configuring the Matching Service

To configure an eView Studio master index for specific data types and for the Sun SBME, you must customize the Matching Service by modifying the Match Field file in the eView Studio Project. Configuring the matching service consists of the following four tasks.

5.2.1 Standardization Configuration

The **StandardizationConfig** section of the Match Field file determines which fields are normalized, parsed, or phonetically encoded and defines the nationality of the data being processed. The standardization section includes the following structures.

The **StandardizationConfig** section defines fields that will be normalized, fields that will be parsed and normalized, and fields that will be phonetically encoded. The standardization types you specify in this section correspond to the match configuration file; the field IDs you can specify are listed in **Table 4 on page 30**.

Normalization Structures

The normalization structure defines fields that are already parsed, but need to be normalized. It also tells the Sun SBME where to place the normalized data in the object structure. Matching on any of these fields is determined by the match string and the logic is defined in the match configuration file.

Of the three data types processed by the Sun SBME, only the person name data type is expected to provide information in fields that are already parsed; that is, the first, last, and middle names appear in separate fields, as do the suffix, title, and so on. The person standardization files define logic for normalizing person name fields. By default, only person first, last, and middle names and the alias first, last, and middle names are defined for normalization. You can specify additional name fields for

normalization, such as maiden name, spouse's name, and so on. For each normalization structure, you must specify the national domains for the data you are processing.

Defining new fields for normalization

The fields you define for normalization in the Match Field file can include any name fields. If you define normalization for fields that are not currently defined for normalization in the Match Field file, you must make the following modifications to the remaining configuration files.

1. In the Match Field file, define the normalization structure, using the appropriate standardization type (PersonName), domain selector, and field IDs (FirstName, MiddeName, or LastName).
2. Add the new fields that will store the normalized field value to the appropriate objects in the Object Definition file.
3. If any of the normalized fields are to be used for blocking, modify the Candidate Select file by adding the new fields to the blocking query.
4. Regenerate the eView Studio application in Enterprise Designer to include the new fields in the database creation script, the outbound Object Type Definition (OTD), and the method OTD.

Defining new normalized fields for matching

If you want to match on the new fields storing the normalized data, you must perform the following steps.

1. Determine the match type or the match comparison function you want to use to match the normalized data, and modify the match configuration file (**matchConfigFile.cfg**) if needed.
2. Add the new normalized field to the **match-columns** element of the **MatchingConfig** section of the Match Field file, making sure to use the appropriate match type from the match configuration file.

Standardization Structures (Parsing and Normalization)

The fields that must be parsed, and possibly normalized, are defined in a standardization structure in **StandardizationConfig**. The standardization structure tells the Sun SBME where to place the standardized information extracted from the parsed fields. The target fields you specify for standardization facilitate searching by the parsed values. Matching on any of these fields is determined by the match string and the logic is defined in the match configuration file.

The Sun SBME expects business names and street address information in freeform text fields that must be parsed and normalized prior to matching. The logic for parsing and normalizing street address information is contained in the address standardization files; the logic for parsing and normalizing business names is contained in the business standardization files. You can customize the standardization of these data types by modifying the appropriate patterns file. For each standardization structure, you must specify the national domains for the data being processed.

Defining new Fields for Standardization

The fields you define for standardization in the Match Field file can include any street address or business name field. Perform the following tasks if you need to define one of these field types for standardization.

1. If necessary, modify the patterns file (you can define new input and output patterns or modify existing ones).
2. Define the standardization structure, using the appropriate standardization type (BusinessName or Address), domain selector, and field IDs (described in **Table 4 on page 30**).
3. Add the new fields that will store the parsed or normalized data to the appropriate objects in the Object Definition file.
4. If any of the parsed or normalized fields are to be used for blocking, modify the Candidate Select file by adding the new fields to the blocking query.
5. Regenerate the eView Studio application in Enterprise Designer to include the new fields in the database creation script, the outbound Object Type Definition (OTD), and the method OTD.

Defining new standardized fields for matching

If you want to match on the new fields that store standardized data, you must perform the following steps.

1. Determine the match type or the match comparison function you want to use to match the parsed data, and modify the match configuration file (**matchConfigFile.cfg**) if needed.
2. Add the new standardized field to the **match-columns** element of the **MatchingConfig** section of the Match Field file, making sure to use the appropriate match type from the match configuration file.

Phonetic Encoding Structures

The fields that must be phonetically encoded are defined in a phonetic encoding structure in **StandardizationConfig**. The phonetic encoding structure tells the Sun SBME where to place the phonetic data created from the fields that are encoded. You can define any field in the object structure for phonetic encoding.

Defining new fields for phonetic encoding

The fields you define for phonetic encoding in the Match Field file can include any field.

1. Determine the type of phonetic encoder to use to convert the field. You can use any of the encoders described in **Table 8 on page 41**.
2. Define the phonetic encoding structure, as described in chapters 6 through 8.
3. Add the new fields that will store the phonetic values to the appropriate objects in the Object Definition file.
4. If any the phonetic fields are to be used for blocking, modify the Candidate Select file by adding the new fields to the blocking query.

5 Regenerate the eView Studio application in Enterprise Designer to include the new fields in the database creation script, the outbound OTD, and the method OTD.

5.2.2 Matching Configuration

The **MatchingConfig** section determines which fields are passed to the Sun SBME for matching (the match string). If you are matching on fields parsed from a freeform text field, define each individual parsed field you want to use for matching. The default fields listed in **MatchingConfig** depend on the fields you specified for matching in the eView Wizard (for eIndex SPV, the default fields are FirstName, LastName, DOB, Gender, and SSN).

The match types you can use for each field in this section are defined in the first column of the match configuration file. Make sure the match type you specify has the correct matching logic defined in the match configuration file.

5.2.3 Match and Standardization Engine Configuration

The **MEFAConfig** section of the Match Field file defines which standardization and match engines will be used by the eView Studio application. By default, the eView Studio application is already configured to use the Sun SBME for matching and standardization. For more information, see "MEFA Configuration" in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*.

Table 7 lists the elements in the Match Field file that define the match and standardization engine, along with the appropriate values for the Sun SBME.

Table 7 Sun SeeBeyond Match Engine Standardization and Match Classes

Match Field File Element	Sun SeeBeyond Match Engine Value
standardizer-api	com.stc.eindex.matching.adapter.SbmeStandardizerAdapter
standardizer-config	com.stc.eindex.matching.adapter.SbmeStandardizerAdapter Config
matcher-api	com.stc.eindex.matching.adapter.SbmeMatcherAdapter
matcher-config	com.stc.eindex.matching.adapter.SbmeMatcherAdapter Config

5.2.4 Phonetic Encoder Configuration

The Sun SBME supports several phonetic encoders, which are defined in the PhoneticEncodersConfig section of the Match Field file. Any encoders specified in the phonetic encoding structures (see **"Phonetic Encoding Structures" on page 39**) must also be defined in the PhoneticEncodersConfig section. The classes for the encoders are listed in Table 8.

- **Soundex** - This algorithm is an industry standard for phonetically encoding first names.
- **French Soundex** - This algorithm is based on the Soundex algorithm, but is customized for French characters and names.

- **Refined Soundex** - This algorithm is similar to the Soundex algorithm, but is optimized for spell checking.
- **NYSIIS** - This algorithm is an industry standard for phonetically encoding last names.
- **Metaphone** - This algorithm is similar to the Soundex algorithm, but is better at identifying words that sound similar. This encoder is limited to encoding a single word in ASCII format containing only characters in the A - Z range. No punctuation or numbers can be in the input string.
- **Double Metaphone** - This algorithm is an improvement on the Metaphone algorithm, at times returning two encodings for a word that could have multiple pronunciations.

Table 8 Phonetic Encoder Classes for the Sun SeeBeyond Match Engine

Encoder	Java Class
Soundex	com.stc.eindex.phonetic.impl.Soundex
NYSIIS	com.stc.eindex.phonetic.impl.NYSIIS
Metaphone	com.stc.eindex.phonetic.impl.Metaphone
Double Metaphone	com.stc.eindex.phonetic.impl.DoubleMetaphone
Refined Soundex	com.stc.eindex.phonetic.impl.RefinedSoundex
French Soundex	com.stc.eindex.phonetic.impl.SoundexFR

5.3 Implementing Domain-specific Standardization Files

Implementing domain-specific standardization files involves two primary steps:

- **Specifying a Domain Selector** on page 41
- **Specifying Multiple Domains** on page 42
- **Loading Standardization Files** on page 44

You only need to perform the first step if you are implementing national domains other than the United States. Perform the second step only if you want to process data from multiple national domains.

5.3.1 Specifying a Domain Selector

An attribute in the Match Field file, **domain-selector**, lets the standardization engine know which standardization files to use. Each standardization and normalization structure must be modified to point to the correct domain. If a domain selector is not specified, the default is the United States domain.

To specify a domain selector

1. In the Project Explorer in Enterprise Designer, expand the eView Studio Project, expand the eView Studio application, and then expand the **Configuration** folder.

2 Check out and open the Match Field file.

3 For each normalization or standardization structure, change the value of the **domain-selector** attribute to the appropriate domain selector (for possible values, see Table 9). For example:

```
<group standardization-type="PersonName" domain-selector=
"com.stc.eindex.matching.impl.SingleDomainSelectorUK">
```

4 Save and check in the Match Field file.

Note: *For more information about modifying the Match Field file, see the* **Sun SeeBeyond eView Studio Configuration Guide**.

Table 9 Domain Selectors

For this national domain…	enter this domain-selector
Australia	com.stc.eindex.matching.impl.SingleDomainSelectorAU
France	com.stc.eindex.matching.impl.SingleDomainSelectorFR
Great Britain	com.stc.eindex.matching.impl.SingleDomainSelectorUK
United States	com.stc.eindex.matching.impl.SingleDomainSelectorUS
Multiple Domains	com.stc.eindex.matching.impl.MultiDomainSelector

5.3.2 Specifying Multiple Domains

The multiple domain selector requires that one field in the object structure identify which national domain to use for each field that will be standardized. For example, the value of the **Country** field in a system record could be used to tell the standardization engine which domain to use for a particular set of data. If you specified the multiple domain selector in the **domain-selector** element, you must also define the identifying field and then map the values that can be populated into that field to their corresponding national domain (or *locale*).

The following rules apply to the multiple domain selector.

- You can specify a value of "Default" for the identifying field. The corresponding national domain will be used if the identifying field is blank, contains the value "Default", or contains a value not defined by any of the **value** elements.
- If a "Default" value is not defined, the system default domain, United States, is used as the default.

To specify multiple domains

1 In the Project Explorer in Enterprise Designer, expand the eView Studio Project, expand the eView Studio application, and then expand the **Configuration** folder.

2 Check out and open the Match Field file.

3 For each normalization or standardization structure for which you want to use multiple domains, change the **domain-selector** value to

"com.stc.eindex.matching.impl.MultiDomainSelector", and then specify the elements described in Table 10. For example:

```
<locale-field-name>Person.PobCountry</locale-field-name>
   <locale-maps>
      <locale-codes>
         <value>GB</value>
         <locale>UK</locale>
      </locale-codes>
      <locale-codes>
         <value>UNST</value>
         <locale>US</locale>
      </locale-codes>
      <locale-codes>
         <value>AU</value>
         <locale>AU</locale>
      </locale-codes>
      <locale-codes>
         <value>FR</value>
         <locale>FR</locale>
      </locale-codes>
         <value>Default</value>
         <locale>AU</locale>
      </locale-codes>
   </locale-maps>
```

4 Save and check in the Match Field file.

5 Regenerate the application and redeploy the Project.

Table 10 Domain Configuration Elements

Element	Description
locale-field-name	The ePath to a field in the object structure that will identify which of the defined **local-codes** elements to use. If this field is blank, the standardization engine defaults to the United States domain, regardless of whether any of the following elements are defined. This field must be contained in the object that contains the fields to be standardized.
locale-maps elements	
locale-codes	Each locale codes stanza defines a value (**value**) that could be contained in the identifying field (**locale-field-name**) along with the corresponding domain (**locale**).
value	The value that must be contained in the identifying field to indicate that the standardization engine will use the corresponding **locale** element to determine which national domain to use to standardize the data. You can specify a default domain by entering "Default" in the **value** element and one of the locale codes described below in the **locale** element.

Table 10 Domain Configuration Elements

Element	Description
locale	A domain code indicating which national domain to use to standardize data when the identifying field value in a transaction matches the corresponding **value** element. Use any of the following codes: ♦ For Australian data, specify **AU** ♦ For French data, specify **FR** ♦ For Great Britain data, specify **UK** ♦ For United States data, specify **US**

5.3.3 Loading Standardization Files

Loading the standardization files brings them into the Repository and the eView Studio Project. This procedure is only required for Projects that were upgraded from previous versions and that do not contain all the needed files. The files are loaded into the **Standardization Engine** node, with domain-specific files being loaded into their own subdirectory. In a fresh installation of eView Studio, all files are automatically loaded.

To load standardization files

1. In the Project Explorer in Enterprise Designer, expand the eView Studio Project, and then expand the eView Studio application.
2. Right-click the **Standardization Engine** folder, and then select **Load Configuration Files** from the context menu.
3. In the Open dialog box, open the folder containing the files you want to load.
4. Select the files to load, and then click **Open**.
5. On the Information dialog box, click **OK**.

Chapter 6

Person Data Type Configuration

Processing person data involves normalizing and phonetically encoding certain fields prior to matching. This chapter describes the configuration files that define person processing logic, and provides instructions for modifying the Match Field file for processing person data. The information presented in this chapter is especially pertinent to the eIndex SPV application.

What's in This Chapter

6.1 Person Matching Overview

Matching on the person data type includes standardizing and matching a person's demographic information. The Sun SBME can create normalized and phonetic values for person data. Several configuration files designed specifically to handle person data are included to provide additional logic for the standardization and phonetic encoding process. The Sun SBME can phonetically encode any field you specify, with some modification to the standardization files. It can also match on any field, as long as the match type for the field is defined in the match configuration file (**matchConfigFile.cfg**).

In addition, when storing person information, you might want to standardize addresses to enable searching against address information. This requires working with the address configuration files described in **Chapter 7**, **"Address Data Type Configuration"**.

6.1.1 Person Data Processing Fields

When matching on person data, not all fields in a record need to be processed by the Sun SBME. The match engine only needs to process fields that must be parsed, normalized, or phonetically converted, and the fields against which matching is performed. These fields are defined in the Match Field configuration file and

processing logic for each field is defined in the Sun SBME standardization and matching configuration files.

Match String Fields

The match string processed by the Sun SBME is defined by the match fields specified in the Match Field file. The match engine can process any combination of fields you specify for matching. By default, the match configuration file includes rows specifically for matching on first name, last name, social security number, and dates (such as a date of birth). It also includes a row for matching a single character, such as might be the case in a gender field. You can use any of the existing rows for matching or you can create new rows for the fields you want to match. Any field for which you specify a Match Type in the eView Wizard is added to the match string.

Standardized Fields

The Sun SBME expects person data to be provided in separate fields within a single record, meaning that no parsing is required of the name fields prior to normalization. The match engine is designed to normalize only first and last name fields in person data and can phonetically convert any field you choose using any of the supported phonetic encoders.

The Object Structure

The fields you specify for person name matching in the eView Studio Wizard are automatically defined for standardization and phonetic encoding. If you specify the appropriate match types in the eView Wizard, the following fields are automatically added to the object structure and database creation script.

- <*field_name*>_Std
- <*field_name*>_Phon

where <*field_name*> is the name of the field for which you specified person name matching. For example, if you specify the PersonFirstName match type for the **FirstName** field, two fields, **FirstName_Std** and **FirstName_Phon**, are automatically added to the structure. You can also add these fields manually if you do not specify match types in the eView Studio Wizard. If you store additional names in the database, such as alias names, maiden names, parent names, and so on, you can modify the phonetic structure to phonetically encode those names as well.

Note: *The object structure for eIndex SPV uses a slightly different naming convention. For the names of the fields defined for eIndex SPV, refer to the* ***Sun SeeBeyond eIndex Single Patient View User's Guide****.*

6.2 Match Configuration for Person Data

The default match configuration file, **matchConfigFile.cfg**, defines several match types for the kinds of data typically included in a person master index. You can customize the existing match types or create new match types for the data being processed. The following match types are typical for matching on person data

- FirstName
- LastName
- String
- Date
- Numeric
- Integer
- Real
- SSN
- Char
- pro
- Exac

This file appears under the **Match Engine** node of the eView Studio Project. For more information about the comparison functions used for each match type and how the weights are tuned, see **"The Match Configuration File" on page 21** and **"Match Configuration Comparison Functions" on page 100**.

6.3 Standardization Configuration for Person Data

Several configuration files are used to define standardization logic for the Sun SBME. You can customize any of the configuration files described in this section to fit your processing and standardization requirements for person data. There are two types of standardization files for person data: common and domain-specific. The common files appear under the **Standardization Engine** node of the eView Studio Project and are used for all national domains; the domain-specific files appear within sub-folders of the **Standardization Engine** node and each corresponds to a specific national domain.

6.3.1 Common Standardization Files for Person Data

The standardization files described in this section are common to all national domains. These files define special characters to remove from name fields and define hyphenated first names. A patterns file is also common, but is not currently used.

personFirstNameDash.dat

The hyphenated name category file defines first names that include hyphens (such as Anne-Marie) to help the Sun SBME recognize and process these values as first names. The file also classifies each name into a gender category. This file is used to standardize all domains except Australia, which uses the **personFirstNameDashAU.dat** file located in the **Australia** folder, and France, which uses the **personFirstNameDashFR.dat** file located in the **France** folder.

The hyphenated name category files use the following syntax:

```
name gender-class
```

You can modify or add entries in this table as needed. Table 11 describes the columns in the **personFirstNameDash.dat** file.

Table 11 Hyphenated Name Category File

Column	Description
name	A hyphenated first name.
gender-class	An indicator of the gender with which the first name corresponds. The possible values are: ▪ **N**—the name is neutral, and can be applied to male or female first names. ▪ **F**—the name is used for females. ▪ **M**—the name is used for males.

Following is an excerpt from the **personFirstNameDash.dat** file.

```
ANNE-MARIE          F
JEAN-NOEL           M
JEAN-MARIE          M
JEAN-BAPTISTE       M
JEAN-PIERRE         M
JEAN-YVES           M
```

personNamePatt.dat

The person name patterns file is not currently used, but is designed to standardize freeform text name fields.

personRemoveSpecChars.dat

The special characters reference file lists characters that might appear in person data, but that should be ignored. The Sun SBME removes these characters from a field before making any comparisons or before normalizing data. You can define additional characters to remove from person data by simply adding the character to the list.

An excerpt from the **personRemoveSpecChars.dat** file appears below.

```
[
]
{
}
<
>
/
?
*
^
#
!
```

6.3.2 Domain-specific Standardization Files

Most standardization files for person data are specific to each national domain. Each domain node within the **Standardization** node of the Project includes the files defined

in this section. The domain corresponding to each file is indicated at the end of the file name; for example, **personConstants*UK*.cfg** and **personConstants*FR*.cfg**. These domain abbreviations are indicated by an asterisk (*) in the descriptions.

Note: *You can customize these files to add entries of other nationalities or languages, including those containing diacritical marks.*

personConjon*.dat

The conjunction reference file is not currently used, but is designed to work with the person name patterns file during standardization.

personConstants*.cfg

The person constants file defines certain information about the standardization files used for processing person data, primarily the number of lines contained in each file. The number of lines specified here must be equal to or greater than the number of lines actually contained in each file. The constants file for United States data is in the **Standardization** node of the Project and is named **personConstants.cfg**; the person constants file for the other domains is located under the domain name node.

Table 12 lists and describes each parameter in the constants file. The files referenced by these parameters are described on the following pages.

Table 12 Person Constants File Parameters

Parameter	Description
words	The maximum number of words in a given freeform text field containing a person name. This parameter is not currently used.
conjmax	The maximum number of lines in the person conjunction reference file (**personConjon*.dat**).
jrsrmax	The maximum number of lines in the generational suffix category file (**personGenSuffix*.dat**).
nickmax	The maximum number of lines in the first name category file (**personFirstName*.dat**).
lastmax	The maximum number of lines in the last name category file (**personLastName*.dat**).
premax	The maximum number of lines in the last name prefix category file (**personLastNamePrefix*.dat**).
titlmax	The maximum number of lines in the title category file (**personTitle*.dat**).
sufmax	The maximum number of lines in the occupational suffix category file (**personOccupSuffix*.dat**).
skpmax	The maximum number of lines in the business name reference file (**businessOrRelated*.dat**).
ptrnmax1	The maximum number of lines in the person patterns file (**personNamePatt.dat**).

Table 12 Person Constants File Parameters

Parameter	Description
twomax	The maximum number of lines in the two-character reference file for occupational suffixes (**personTwo*.dat**).
thremax	The maximum number of lines in the three-character reference file for occupational suffixes (**personThree*.dat**).
blnkmax	The maximum number of lines in the special characters reference file (**personRemoveSpecChars.dat**).
dashSize	The maximum number of lines in the hyphenated name category file (**personFirstNameDash.dat**).

personFirstName*.dat

The first name category file defines standardized versions of first names and assigns a gender classification for each name. This file is used to standardize first names when comparing person names. The gender classification helps to further clarify the match. The Sun SBME uses this file when a first name field is defined for normalization or standardization in the Match Field file.

The syntax of this file is:

```
original-value standardized-form gender-class
```

You can modify or add entries in this table as needed. Table 13 describes the columns in the **personFirstName*.dat** file.

Table 13 First Name Category File

Column	Description
original-value	The original value of the first name.
standardized-form	The standardized version of the original value. A zero (0) in this field indicates that the original value is already in its standardized form. If this column contains a name instead of a zero, that name must also be listed in a different entry as an original value with a standardized form of “0”.
gender-class	An indicator of the gender with which the first name corresponds. The possible values are: ▪ **N**—the name is neutral, and can be applied to male or female first names. ▪ **F**—the name is used for females. ▪ **M**—the name is used for males.

Following is an excerpt from the **personFirstNameUS.dat** file. Certain rows contain a zero (0) for the standardized form, indicating that the name is already standard (for example, Stephen, Sterling, and Summer).

```
STEPHEN          0                M
STEPHENIE        STEPHANIE        F
STEPHIE          STEPHANIE        F
```

```
STEPHINE        STEPHANIE       F
STEPHNIE        STEPHANIE       F
STERLING        0               M
STEVE           STEPHEN         M
STEVEN          STEPHEN         M
STEVIE          STEPHEN         N
STEW            STUART          M
STEWART         STUART          M
STU             STUART          M
STUART          0               M
SU              SUSAN           F
SUE             SUSAN           F
SUHANTO         0               M
SULLIVAN        0               F
SULLY           SULLIVAN        F
SUMMER          0               F
```

personGenSuffix*.dat

The generational suffix category file defines standardized versions of generational suffixes, such as Jr., III, and so on. This file is used to compare standard versions of the suffix field. You can define additional suffixes and their standardized form following the syntax below.

```
field-value standard-form
```

Table 14 describes each column of the **personGenSuffix*.dat** file.

Table 14 Generational Suffix Category File

Column	Description
field-value	The original value of the generational suffix in the record being processed.
standard-form	The standard form of the generational suffix. A zero (0) in this column indicates that the value listed in column one is already in its standardized form. If this column contains a suffix instead of a zero, that suffix must also be listed in a different entry as an original value with a standard form of “0”.

An excerpt from the **personGenSuffixUS.dat** file appears below. In this excerpt, certain suffixes, such as 2ND, 3RD and JR, are already in their standardized form.

```
11              2ND
111             3RD
1V              4TH
2ND             0
3RD             0
4TH             0
FOURTH          4TH
II              2ND
III             3RD
IV              4TH
JR              0
JUNIOR          JR
SECOND          2ND
SENIOR          SR
```

personLastNamePrefix*.dat

The last name prefix category file defines standardized versions of last name prefixes, such as “Van” or “Le”. This file is used to standardize these prefixes prior to standardizing the last name when comparing person names. The Sun SBME uses this file when a last name field is defined for normalization or standardization in the Match Field file.

The syntax of this file is:

```
original-value standardized-form
```

You can modify or add entries in this table as needed. Table 15 describes the columns in the **personLastNamePrefix*.dat** file.

Table 15 Last Name Prefix Category File

Column	Description
original-value	The original value of the last name prefix.
standardized-form	The standardized version of the original value. A zero (0) in this field indicates that the original value is already in its standardized form. If this column contains a prefix instead of a zero, that prefix must also be listed in a different entry as an original value with a standardized form of “0”.

Following is an excerpt from the **personLastNamePrefixUS.dat** file. Some of these prefixes are already in their standardized form, such as “Los” and “Mac”.

```
LOS                 0
MAC                 0
MC                  MAC
SAINT               0
ST                  SAINT
VAN                 0
VAN DER             0
VANDE               VAN DER
```

personLastName*.dat

The last name category file defines standardized versions of last names. This file is used to standardize last names when comparing person names. The Sun SBME uses this file when a last name field is defined for normalization or standardization in the Match Field file.

The syntax of this file is:

```
original-value standardized-form
```

You can modify or add entries in this table as needed. Table 16 describes the columns in the **personLastName*.dat** file.

Table 16 Last Name Category File

Column	Description
original-value	The original value of the last name.

Table 16 Last Name Category File

Column	Description
standardized-form	The standardized version of the original value. A zero (0) in this field indicates that the original value is already in its standardized form. If this column contains a name instead of a zero, that name must also be listed in a different entry as an original value with a standardized form of "0".

Following is an excerpt from the **personLastNameUS.dat** file.

```
FINK                            0
PHINQUE                         FINK
```

personOccupSuffix*.dat

The occupational suffix category file is not currently used, but is designed to work with the person name patterns file during standardization.

personThree*.dat

This reference file is not currently used, but is designed to work with the person name patterns file during standardization.

personTitle*.dat

The title category file defines standard forms for titles and classifies each title into a gender category. For example, "Mister" is standardized to "MR" and is classified as male; "Doctor" is standardized to "DR" and is classified as gender neutral. You can add, modify, or delete entries in this file as needed. Use the following syntax.

```
original-value standardized-form gender-class
```

Table 17 describes each column of the **personTitle*.dat** file.

Table 17 Person Title Category File

Column	Description
original-value	The original value of the title in the person name field.
standardized-form	The standardized version of the original value. A zero (0) in this field indicates that the original value is already in its standardized form. If this column contains a title instead of a zero, that title must also be listed in a different entry as an original value with a standardized form of "0".

Table 17 Person Title Category File

Column	Description
gender-class	An indicator of the gender with which the title corresponds. The default values are: ▪ **N**—the title is neither male nor female. ▪ **F**—the title is used for females. ▪ **M**—the title is used for males.

An excerpt from the **personTitleUS.dat** file appears below. In this excerpt, certain titles, such as DR, GEN, and MISS, are already in their standardized form.

```
CTO                    0           N
DEAN                   0           N
DIR                    DIRECTOR    N
DIRECTOR               0           N
DOC                    DR          N
DOCTOR                 DR          N
DR                     0           N
DRS                    0           N
EMERITUS               0           N
FOUNDER                0           N
GEN                    0           N
GENERAL                GEN         N
MANAGER                0           N
MGR                    MANAGER     N
MISS                   0           F
MISSUS                 MRS         F
```

personTwo*.dat

This reference file is not currently used, but is designed to work with the person name patterns file during standardization.

businessOrRelated*.dat

The business-related category file is used to identify business terms in person name information. Examples of when this could occur would be when indexing both person and business names or when business information is included within a person object structure. The Sun SBME removes these terms for person matching. This file contains a list of common business terms that might be found in person data. You can modify this file by adding, changing, or deleting terms.

An excerpt from the **businessOrRelatedUS.dat** file appears below.

```
ACCOUNTANT
ACCT
ACDY
ACRE
ACREAGE
ACRES
ACS
ACT
AD
ADATU
ADM
ADMIN
ADMINISTRATIO
```

```
ADMINISTRATION
ADMINISTRATOR
```

6.4 Customizing Person Data Configuration Files

To customize the Sun SBME configuration files for processing person data, you can modify any of the files described in this chapter using the text editor provided in Enterprise Designer. Before modifying the match configuration file, review the information provided in **Chapter 4 "Matching Configuration Files"** and **Appendix B "Match Configuration Comparison Functions"**. Make sure a thorough data analysis has been performed to determine the best fields for matching and the best comparison functions to use for each field.

Updating most standardization files is a straight-forward process. Make sure to follow the syntax guidelines provided in **"Standardization Configuration for Person Data" on page 47**. If you add any lines to any of the standardization configuration files, be sure to adjust the corresponding parameter in the person constants file (**personConstants*.cfg**).

6.5 Configuring the eView Studio Matching Service for Names

To ensure correct processing of person information, you must customize the eView Studio Matching Service. This includes modifying the Match Field file to support the fields on which you want to match, to standardize the appropriate fields, and to specify the Sun SBME as the match and standardization engine (by default, the Sun SBME is already specified so this does not need to be changed). Perform the following tasks to configure the eView Studio Matching Service.

- **Configuring the Standardization Structure** on page 55
- **Configuring the Match String** on page 58

When configuring the eView Studio Matching Service, keep in mind the information presented in **"Configuring the Matching Service" on page 37**.

6.5.1 Configuring the Standardization Structure

The standardization structure is configured in the **StandardizationConfig** section of the Match Field file, which is described in detail in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*. To configure the required fields for normalization and phonetic encoding, modify the normalization and phonetic encoding structures in the Match Field file. This is described in detail in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*. The following sections provide additional guidelines and samples specific to standardizing person data.

Note: *In the current configuration, the rules defined for the person data type assume the incoming data to be parsed prior to processing. Therefore, you do not need to configure fields to parse unless you want to search on address information. In that case, you must configure address fields to parse and normalize.*

Normalization Structures

The fields defined for normalization for the person data type can include any name fields. By default this includes first, middle, and last name fields. Follow the instructions under "Defining Normalization" in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide* to define fields for normalization. For the **standardization-type** element, enter "PersonName" (for more information, see **"Match and Standardization Types" on page 34**). For a list of field IDs to use in the **standardized-object-field-id** element, see **Table 4 on page 30**.

A sample normalization structure for person data is shown below. This sample specifies that the PersonName standardization type is used to normalize the first name, alias first name, last name, and alias last name fields. For all name fields, both United States and United Kingdom domains are defined for standardization.

```
<structures-to-normalize>
   <group standardization-type="PersonName"
    domain-
selector="com.stc.eindex.matching.impl.MultiDomainSelector">
      <locale-field-name>Person.PobCountry</locale-field-name>
      <locale-maps>
         <locale-codes>
            <value>UNST</value>
            <locale>US</locale>
         </locale-codes>
         <locale-codes>
            <value>GB</value>
            <locale>UK</locale>
            </locale-codes>
      </locale-maps>
      <unnormalized-source-fields>
         <source-mapping>
            <unnormalized-source-field-name>Person.FirstName
            </unnormalized-source-field-name>
            <standardized-object-field-id>FirstName
            </standardized-object-field-id>
         </source-mapping>
         <source-mapping>
            <unnormalized-source-field-name>Person.LastName
            </unnormalized-source-field-name>
            <standardized-object-field-id>LastName
            </standardized-object-field-id>
         </source-mapping>
      </unnormalized-source-fields>
         <normalization-targets>
            <target-mapping>
               <standardized-object-field-id>FirstName
               </standardized-object-field-id>
               <standardized-target-field-name>Person.FirstName_Std
               </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
               <standardized-object-field-id>LastName
               </standardized-object-field-id>
               <standardized-target-field-name>Person.LastName_Std
```

```
                    </standardized-target-field-name>
                </target-mapping>
            </normalization-targets>
        </group>
        <group standardization-type="PersonName" domain-selector=
          "com.stc.eindex.matching.impl.MultiDomainSelector">
          <locale-field-name>Person.PobCountry</locale-field-name>
          <locale-maps>
            <locale-codes>
              <value>UNST</value>
              <locale>US</locale>
            </locale-codes>
            <locale-codes>
              <value>GB</value>
              <locale>UK</locale>
            </locale-codes>
          </locale-maps>
          <unnormalized-source-fields>
            <source-mapping>
              <unnormalized-source-field-name>Person.Alias[*].FirstName
              </unnormalized-source-field-name>
              <standardized-object-field-id>FirstName
              </standardized-object-field-id>
            </source-mapping>
            <source-mapping>
              <unnormalized-source-field-name>Person.Alias[*].LastName
              </unnormalized-source-field-name>
              <standardized-object-field-id>LastName
              </standardized-object-field-id>
            </source-mapping>
          </unnormalized-source-fields>
          <normalization-targets>
            <target-mapping>
              <standardized-object-field-id>FirstName
              </standardized-object-field-id>
              <standardized-target-field-name>
              Person.Alias[*].FirstName_Std
              </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
              <standardized-object-field-id>LastName
              </standardized-object-field-id>
              <standardized-target-field-name>
              Person.Alias[*].LastName_Std
              </standardized-target-field-name>
            </target-mapping>
          </normalization-targets>
        </group>
    </structures-to-normalize>
```

Phonetic Encoding

When you specify a name field for person name matching in the eView Wizard, these fields are automatically defined for phonetic encoding. You can define additional names, such as maiden names or alias names, for phonetic encoding as well. Follow the instructions under "Defining Phonetic Encoding" in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide* to define fields for phonetic encoding.

A sample of fields defined for phonetic encoding is shown below. This sample converts name and alias name fields, as well as the street name.

```
<phoneticize-fields>
```

```
<phoneticize-field>
   <unphoneticized-source-field-name>Person.FirstName_Std
   </unphoneticized-source-field-name>
   <phoneticized-target-field-name>Person.FirstName_Phon
   </phoneticized-target-field-name>
   <encoding-type>Soundex</encoding-type>
</phoneticize-field>
<phoneticize-field>
   <unphoneticized-source-field-name>Person.LastName_Std
   </unphoneticized-source-field-name>
   <phoneticized-target-field-name>Person.LastName_Phon
   </phoneticized-target-field-name>
   <encoding-type>NYSIIS</encoding-type>
</phoneticize-field>
<phoneticize-field>
   <unphoneticized-source-field-name>Person.Alias[*].FirstName_Std
   </unphoneticized-source-field-name>
   <phoneticized-target-field-name>Person.FirstName_Phon
   </phoneticized-target-field-name>
   <encoding-type>Soundex</encoding-type>
</phoneticize-field>
<phoneticize-field>
   <unphoneticized-source-field-name>Person.Alias[*].LastName_Std
   </unphoneticized-source-field-name>
   <phoneticized-target-field-name>Person.LastName_Phon
   </phoneticized-target-field-name>
   <encoding-type>NYSIIS</encoding-type>
</phoneticize-field>
<phoneticize-field>
   <unphoneticized-source-field-name>
     Person.Address[*].AddressLine1_StName
   </unphoneticized-source-field-name>
   <phoneticized-target-field-name>
     Person.Address[*].AddressLine1_StPhon
   </phoneticized-target-field-name>
   <encoding-type>NYSIIS</encoding-type>
</phoneticize-field></phoneticize-fields>
```

6.5.2 Configuring the Match String

You can include any fields on which you want to match in the match string. The match string is defined by the **match-column** elements in the **MatchingConfig** section of the Match Field file. If you specify a Match Type for a field in the eView Wizard, that field (or any fields parsed from that field) is automatically defined in the match string.

To configure the match string, follow the instructions under "Configuring the Match String" in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*. For the Sun SBME, each data type has a different match type (specified by the **match-type** element). The FirstName and LastName 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 **"Match and Standardization Types" on page 34**.

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>
```

Chapter 7

Address Data Type Configuration

Processing street addresses involves parsing, normalizing, and phonetically encoding certain fields prior to matching. This chapter describes the configuration files that define address processing logic and provides instructions for modifying the Match Field file for processing address fields.

What's in This Chapter

7.1 Address Matching Overview

Matching on the address data type includes both standardizing and matching on address information in the master index. You can implement street address standardization and matching on its own, or within a master index designed to process person or business information. For example, standardizing address information allows you to include address fields as search criteria, even though matching might not be performed against these fields.

The Sun SBME can create standardized and phonetic values for street address information. Several configuration files are designed specifically to handle address data to define additional logic for the standardization and phonetic encoding process. These include address clues files, a patterns file, and a constants file. The United States address standardization engine is based on the work performed at the US Census Bureau. The clues files, in particular, are based on census bureau statistics.

The Sun SBME can match on any field as long as the match type for the field is defined in the match configuration file (**matchConfigFile.cfg**).

7.1.1 Address Data Processing Fields

When matching on address data, not all fields in a record need to be processed by the Sun SBME. The match engine only needs to process fields that must be parsed, normalized, or phonetically encoded, and the fields against which matching is

performed. These fields are defined in the Match Field file and processing logic for each field is defined in the standardization and matching configuration files.

Match String Fields

The match string processed by the Sun SBME is defined by the match fields specified in the Match Field file. If you specify an "Address" match type for any field in the eView Wizard, the parsed address fields are automatically added to the match string in the Match Field file. These fields include the house number, street direction, street type, and street name. You can remove any of these fields from the match string.

The match engine can process any combination of fields you specify for matching. By default, the match configuration file 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 use any of the existing rows for matching or you can create new rows for the fields you want to match.

Standardized Fields

The Sun SBME expects that street address data will be provided in a freeform text field containing several components that must be parsed. The match engine is designed to parse these components and to normalize and phonetically encode the street name. You can specify additional fields for phonetic encoding.

If you specify an "Address" match type for any field in the eView Wizard, a standardization structure for that field is defined in the Match Field file. The fields listed below under **"The Object Structure"** are automatically defined as the target fields. Each of these fields has several entries in the standardization structure. This is because different parsed components can be stored in the same field. For example, the house number, post office box number, and rural route identifier are all stored in the house number field. If you do not specify address fields for matching in the eView Wizard but want to standardize the fields, you can create a standardization structure in the Match Field file.

The Object Structure

The address fields specified for standardization are parsed into several additional fields. If you specify the "Address" match type in the eView Wizard, the following fields are automatically added to the object structure and database creation script.

- <*field_name*>_HouseNo
- <*field_name*>_StName
- <*field_name*>_StDir
- <*field_name*>_StType
- <*field_name*>_StPhon

  where <*field_name*> is the name of the field for which you specified address matching. For example, if you specify the Address match type for the **AddressLine1** field, the following fields are automatically added to the structure:

AddressLine1_HouseNo, **AddressLine1_StName**, **AddressLine1_StDir**, **AddressLine1_StType**, and **AddressLine1_StPhon**.

You can add these fields manually if you do not specify a match type in the eView Wizard.

Note: *The object structure for eIndex SPV uses a slightly different naming convention. For the names of the fields defined for eIndex SPV, refer to the* ***Sun SeeBeyond eIndex Single Patient View User's Guide****.*

7.2 Match Configuration for Address Data

The default match configuration file, **matchConfigFile.cfg**, defines several match types for the kinds of address data typically included in the match string. You can customize the existing match types or create new match types for the data being processed. The following match types are typical for matching on address data.

- StreetNumber
- HouseNumber
- StreetDir
- StreetType

In addition, you can use any of these generic match types for matching on address data.

- String
- Date
- Numeric
- Integer
- Real
- SSN
- Char
- pro
- Exac

The match configuration file appears under the **Match Engine** node of the eView Studio Project. For more information about the comparison functions used for each match type and how the weights are tuned, see **"The Match Configuration File" on page 21** and **"Match Configuration Comparison Functions" on page 100**.

7.3 Standardization Configuration for Address Data

Several configuration files are used to define address processing logic for the Sun SBME. You can customize any of the configuration files described in this section to fit your processing and standardization requirements for address data. There are no address standardization files that are common to all domains; all address files are domain-specific. These files are located within the domain-specific folders of the **Standardization Engine** node (with two exceptions noted below).

Address standardization files are specific to each domain and include patterns and clues files, as well as files that define internal and external constants. The domain corresponding to each file is indicated at the end of the file name; for example,

addressConstants*UK*.cfg and **addressConstants*FR*.cfg**. These domain abbreviations are indicated by an asterisk (*) in the following descriptions.

addressConstants*.cfg

The address constants file defines certain information about the standardization files used for processing address data, primarily the number of lines contained in each file. The number of lines specified here must be equal to or greater than the number of lines actually contained in each file. The constants file for United States data is in the **Standardization** node of the Project and is named **addressConstants.cfg**; the constants file for the other domains is located under the domain name node.

Table 18 lists and describes each parameter in the constants file. The files referenced by these parameters are described on the following pages.

Table 18 Address Constants File Parameters

Parameter	Description
maxWords	The maximum number of words in a given address field.
clueArraySize	The maximum number of lines in the address clues file (**addressClueAbbrev*.dat**).
patternArraySize	The maximum number of lines in the patterns file (**addressPatterns*.dat**).
maxPattSize	The maximum length (in characters) of any pattern in the address patterns file.
imageSize	The maximum length of an input address field.
nameOutputFieldSize	The maximum output length of a street or property name.
numberOutputFieldSize	The maximum output length of a house number or rural route number within the structure identifier or post office box fields.
directionOutputFieldSize	The maximum output length of a directional field (prefix or suffix).
typeOutputFieldSize	The maximum output length of a street type field (prefix or suffix).
prefixOutputFieldSize	The maximum length of a number prefix field.
suffixOutputFieldSize	The maximum length of a number suffix field.
extensionOutputFieldSize	The maximum output length of any extension field.
extrainfoOutputFieldSize	The maximum output length of any miscellaneous information that is not recognized as a known type.

addressClueAbbrev*.dat

The address clues file lists common terms in street addresses, specifies a normalized value for each common term, and categorizes the terms into street address component types. A term can be categorized into multiple component types. The relevance value

specifies which of the component types the term is most likely to be. For example, the term "Junction" is standardized as "Jct", and is classified as a street type, building unit, and generic term (giving relevance in that order).

This file helps the Sun SBME recognize common terms in street addresses, and to parse and normalize the values correctly. The syntax of this file is:

```
common-term normalized-term ID-number/type-token
```

You can modify or add entries in this table as needed. Table 19 describes the columns in the **addressClueAbbrev*.dat** file.

Table 19 Address Clues File Columns

Column	Description
common-term	A term commonly found in street addresses.
normalized-term	The normalized version of the common term.
ID-number/type-token	An ID number and a token indicating the type of address component represented by the common term. The ID number corresponds to an ID number in the address master clues file, and the type token corresponds to the type specified for that ID number in the address master clues file. One term might have several ID number and token type pairs.

Following is an excerpt from the **addressClueAbbrevUS.dat** file.

```
TRLR VLG          Trpk          59BU
TRPK              Trpk          59BU
TRPRK             Trpk          59BU
VILLA             Vlla          305TY    60BU
VLLA              Vlla          305TY    60BU
VILLAS            Vlla          60BU
VILL              Vlg           317TY    61BU     364AU
VILLAG            Vlg           317TY    61BU     364AU
VLG               Vlg           317TY    61BU     364AU
VILLAGE           Vlg           317TY    61BU     364AU
VILLG             Vlg           317TY    61BU     364AU
VILLIAGE          Vlg           317TY    61BU     364AU
VLGE              Vlg           317TY    61BU     364AU
VIVI              Vivi          62BU
VIVIENDA          Vivi          62BU
COLLEGE           Coll          64BU               0AU
CLG               Coll          64BU
COTTAGE           Cott          65BU     65BP      0AU
```

addressInternalConstants*.cfg

The address internal constants file defines and configures tokens and array sizes used by the address standardizer. This file is used internally by the standardization engine and most of the parameters should not be modified.

One parameter you might need to modify is **spCh**, which defines any special characters that should not be removed from addresses during standardization. By default, the standardization process keeps hyphens (-), pound signs (#), forward slashes (/), ampersands (&), and pipes (|). Any other special characters found in the address are

removed unless they are defined for the **spCh** parameter. Delineate each special character in the list with a space, as shown below.

```
spCh = & < >
```

Characters that are not included in the standard ISO 8859-1 (Latin-1) character set must be preceded by a back slash (\) and represented in Unicode. For example, use the following to retain right and left single quotes (‘ ’) in addresses:

```
spCh = \u2018 \u2019
```

Note: *Periods (.) and commas (,) are always removed from addresses, even if they are added to the* ***spCh*** *list.*

addressMasterClues*.dat

The address master clues file lists common terms in street addresses as defined by the United States Postal Service (USPS), the United Kingdom’s Royal Mail, the Australian Postal Corporation, or France’s La Poste (depending on the domain in use). For each common term, this file specifies a normalized value, defines postal information, and categorizes the terms into street address component types. A term can be categorized into multiple component types.

The syntax of this file is:

```
ID-number common-term normalized-term short-abbrev postal-abbrev
CFCCS type-token usage-flag postal-flag
```

You can modify or add entries in this table as needed. Table 20 describes the columns in the **addressMasterClues*.dat** file.

Table 20 Address Master Clue File Columns

Column	Description
ID-number	A unique identification number for the address common term. This number corresponds to an ID number for the same term in the address clues file.
common-term	A common address term, such as Park, Village, North, Route, Centre, and so on.
normalized-term	The normalized version of the common term.
short-abbrev	A short abbreviation of the common term.
postal-abbrev	The standard postal abbreviation of the common term.

Table 20 Address Master Clue File Columns

Column	Description
CFCCS	The census feature class code of the term (as defined in the Census Tiger® database). The following values are used: ▪ **A**—Road ▪ **B**—Railroad ▪ **C**—Miscellaneous ▪ **D**—Landmark ▪ **E**—Physical feature ▪ **F**—Nonvisible feature ▪ **H**—Hydrography ▪ **X**—Unclassified
type-token	The type of address component represented by the common term. Types are specified by an address token (for more information, see **Address Type Tokens** on page 69).
usage-flag	A flag indicating how the term is used (for more information, see **Pattern Classes** on page 71)
postal-flag	The standard postal code for the term.

Following is an excerpt from the **addressMasterCluesUS.dat** file.

```
11Alley              Alley       Al       Aly A    TY R U
12Alternate Route    Alt Rte     Alt      Alt A    TY R
15Arcade             Arcade      Arc      Arc A    TY R U
16Arroyo             Arroyo      Arryo    ArryHA   TY R
17Autopista          Atpta       Apta     AptaA    TY R
18Avenida            Avenida     Ava      Ava A    TY R
19Avenue             Avenue      Ave      Ave A    TY R U
26Boulevard          Blvd        Blvd     BlvdA    TY R U
32Bulevar            Blvr        Blv      Blv A    TY R
33Business Route     Bus Rte     BusRt    BsRtA    TY R
34Bypass             Bypass      Byp      Byp A    TY R U
36Calle              Calle       Calle    ClleA    TY R
37Calleja            Calleja     Cja      Cja A    TY R
38Callejon           Callej      Cjon     CjonA    TY R
39Camino             Camino      Cam      Cam A    TY R
47Carretera          Carrt       Carr     CarrA    TY R
48Causeway           Cswy        Cswy     CswyAH   TY R U
51Center             Center      Ctr      Ctr DA   TY R U
```

addressPatterns*.dat

The address patterns file defines the expected input patterns of each individual street address field being standardized so the Sun SBME can recognize and process these values. Tokens are used to indicate the type of address component in the input and output fields. This file contains two rows for each pattern. The first row defines the input pattern for each address field and provides an example. The second row defines the output pattern for each address field, the pattern type, the relative importance of the pattern compared to other patterns, and usage flags (as shown below).

```
AU A1 TY          01 Oak B Street
```

```
NA NA ST          T* 75              TX
```

When an address is parsed, each line of the address is delineated by a pipe (|) and sent to the parser separately. The output tokens for each line are then concatenated and the output pattern is processed using the **addressOutPatterns*.dat** file to determine whether the output pattern is listed in the file. If the pattern is found, output patterns are modified as indicated in the **addressOutPatterns*.dat** file to resolve any ambiguities that might arise when two lines of address information contain common elements. The relative importance determines which pattern to use when the format of the input field matches more than one pattern. This file should only be modified by personnel with a thorough understanding of address patterns and tokens.

The syntax of this file is:

```
input-pattern   example
output-pattern  pattern-class pattern-modifier priority usage-flag
exclude-flag
```

You can modify or add entries in this table as needed. Table 21 describes the columns in the **addressPatterns*.dat** file.

Table 21 Address Patterns File

Column	Description
input-pattern	Tokens that represent a possible input pattern from an individual unparsed street address field. Each token represents one component. For more information about address tokens, see **Address Type Tokens** on page 69.
example	An example of a street address that fits the specified pattern. This file element is optional.
output-pattern	Tokens that represent the output pattern for the specified input pattern. Each token represents one component of the output of the Sun SBME. For more information about address tokens, see **Address Type Tokens** on page 69.
pattern-class	An indicator of the type of address component represented by the pattern. Possible pattern types are listed in **Pattern Classes** on page 71.
pattern-modifier	An indicator of whether the priority of the pattern is averaged against other patterns that match the input. Pattern modifiers are listed in **Pattern Modifiers** on page 72.
priority	The priority weight to use for the pattern when the pattern is a sub-pattern of a larger input pattern. For more information, see **Priority Indicators** on page 72.
usage-flag	A flag indicating how the term is used (for more information, see **Pattern Classes** on page 71). This file element is optional.
exclude-flag	This file element is optional.

Following is an excerpt from the **addressPatternsUS.dat** file.

```
NU DR TY A1 AU        01   123 South Avenida B Oak
HN PD PT NA NA        H* 70

NU DR TY NU DR        01   123 South Avenida 1 West
HN PD PT NA SD        H* 70

NU A1 TY AU TY        01   123 C circle hill drive
HN HS NA NA ST        H* 70

NU A1 AM A1 TY        01   123 M & M road
HN NA NA NA ST        H* 65

NU TY AU A1           01   123 Avenida Oak B
HN PT NA NA           H* 60

NU TY NU A1           01   123 Avenida 1 B
HN PT NA NA           H* 60
```

addressOutPatterns*.dat

The address output patterns file uses the field patterns output by the **addressPatterns*.dat** file to determine how to parse all standardized address fields. As with the **addressPatterns*.dat** file, tokens are used to indicate the type of address component in the input and output data. This file contains two rows for each pattern. The first row defines the input pattern received from **addressPatterns*.dat** and provides an example. The second row defines the output pattern (as shown below).

```
EI|BN BT|*           // HILLVIEW|FULBOURN HOSPITAL
BN|BI BY
```

The syntax of this file is:

```
input-pattern example
output-pattern
```

You can modify or add entries in this table as needed. Table 22 describes the columns in the **addressOutPatterns*.dat** file.

Table 22 Address Output Patterns File

Column	Description
input-pattern	Tokens that represent a possible input pattern from **addressPatterns*.dat**. Each token represents one component and the pattern for each address field in the address is separated by a pipe (|). For more information about address tokens, see **Address Type Tokens** on page 69. Note that this file only uses output tokens.
example	An example of a street address that fits the specified pattern. This file element is optional.

Table 22 Address Output Patterns File

Column	Description
output-pattern	Tokens that represent the output pattern for the specified input pattern. Each token represents one component of the output of the Sun SBME. For more information about address tokens, see **Address Type Tokens** on page 69.

Following is an excerpt from the **addressPatternsUS.dat** file. In the first example, **addressPatternsUS.dat** outputs three address fields containing these components: building name and type; street name and type; and street name and type. **addressOutPatternsUS.dat** changes the tokens for the second street name and type to indicate they are not the primary street name and type. Therefore, "New Bridge" is populated into the parsed street name field in the database.

```
BN BT|NA ST|NA ST|*          // PROTEA HOUSE|NEW BRIDGE|MARINE PARADE
BN BT|NA ST|N2 S2

HN NA ST|HN NA ST|*          // 21 HEIGHWAY COURT|45 BROOKLAND ROAD
HN NA ST|H2 N2 S2

HN NA ST|NA ST|*             // 21 HEIGHWAY COURT|BROOKLAND ROAD
HN NA ST|N2 S2

NA ST|HN NA ST|*             // HEIGHWAY COURT|45 BROOKLAND ROAD
NA ST|H2 N2 S2
```

Address Pattern File Components

The address patterns files use pattern type tokens, pattern classes, pattern modifies, and priority indicators to process and parse address data. Before modifying any of the patterns files, you must have a good understanding of these file components.

Address Type Tokens

The address pattern and clues files use tokens to denote different components in a street address, such as street type, house number, street names, and so on. These files use one set of tokens for input fields and another set for output fields. You can use only the predefined tokens to represent address components; the Sun SBME does not recognize custom tokens.

Table 23 lists and describes each input token; Table 24 lists and describes each output token.

Table 23 Input Address Pattern Type Tokens

Token	Description
A1	Alphabetic value, one character in length
AM	Ampersand
AU	Generic word
BP	Building property

Table 23 Input Address Pattern Type Tokens

Token	Description
BU	Building unit
BX	Post office box
DA	Dash (as a starting character)
DR	Street direction
EI	Extra information
EX	Extension
FC	Numeric fraction
HR	Highway route
MP	Mile posts
NL	Common words, such as “of”, “the”, and so on
NU	Numeric value
OT	Ordinal type
PT	Prefix type
RR	Rural route
SA	State abbreviation
TY	Street type
WD	Descriptor within the structure
WI	Identifier within the structure

Table 24 Output Address Pattern Tokens

Token	Description
1P	Building number prefix
2P	Second building number prefix
BD	Property or building directional suffix
BI	Structure (building) identifier
BN	Property or building name
BS	Building number suffix
BT	Property or building type suffix
BX	Post office box descriptor
BY	Structure (building) descriptor
DB	Property or building directional prefix
EI	Extra information
EX	Extension index
H1	First house number (the actual number)

Table 24 Output Address Pattern Tokens

Token	Description
H2	Second house number (house number suffix)
HN	House number
HS	House number suffix
N2	Second street name
NA	Street name
NB	Building number
NL	Conjunctions that connect words or phrases in one component type (usually the street name)
P1	House number prefix
P2	Second house number prefix
PD	Directional prefix to the street name
PT	Street type prefix to the street name
RR	Rural route descriptor
RN	Rural route identifier
S2	Street type suffix to the second street name
SD	Directional suffix to the street name
ST	Street type suffix to the street name
TB	Property or building type prefix
WI	Identifier within the structure
WD	Descriptor within the structure
XN	Post office box identifier

Pattern Classes

Each pattern defined in the address patterns file must have an associated pattern class. The pattern class indicates a portion of the input pattern or the type of address data that is represented by the pattern. You can specify any of the following pattern classes.

- **H** - the address pattern represents a house
- **B** - the address pattern represents a building
- **W** - the address pattern represents a unit within a structure, such as an apartment or suite number
- **T** - the address pattern represents a street type or direction
- **R** - the address pattern represents a rural route
- **P** - the address pattern represents a Post Office box
- **N** - the address pattern is mostly numeric

These classes are also specified as usage flags in the patterns file and the master clues file.

Pattern Modifiers

Each pattern type must be followed by a pattern modifier that indicates how to handle cases where one or more defined patterns is found to be a sub-pattern of a larger input pattern. In this case, the Sun SBME must know how to prioritize each defined pattern that is a part of the larger pattern. There are two pattern modifiers.

- * - An asterisk indicates that the priority weight for the matching pattern is averaged down equally with the other matching sub-patterns.
- + - A plus sign indicates that the priority weight for the matching pattern is not averaged down equally with the other matching sub-patterns.

Priority Indicators

The priority indicator is a numeric value following the pattern modifier that indicates the priority weight of the pattern. These values work best when defined as a multiple of five between and including 35 and 95. If a pattern is assigned a priority of 90 or 95 and the pattern matches, or is a sub-pattern of, the input pattern, the match engine stops searching for additional matching patterns and uses the high-priority matching pattern.

7.4 Modifying Address Data Configuration Files

To customize the Sun SBME configuration files for processing street address data, you can modify any of the files described in this chapter using the text editor provided in Enterprise Designer. Before modifying the match configuration file, review the information provided in **Chapter 4 "Matching Configuration Files"** and **Appendix B "Match Configuration Comparison Functions"**. Make sure a thorough data analysis has been performed to determine the best fields for matching and the best comparison functions to use for each field.

Updating most standardization files is a straight-forward process. Make sure to follow the syntax guidelines provided in **"Standardization Configuration for Address Data" on page 62**. If you add rows to any of the standardization files, make sure to adjust the corresponding parameter in the address constants file (**addressConstants.cfg**).

Modifying the patterns file is a more complex task. Only modify this file once you fully understand pattern tokens, types, relevance, and flags.

7.5 Configuring the eView Studio Matching Service

To ensure the master index uses the Sun SBME to process address information, you must customize the eView Studio Matching Service. This includes modifying the Match Field file to support the fields on which you want to match, to standardize the appropriate fields, and to specify the Sun SBME as the match and standardization engine (by default, the Sun SBME is already specified so this does not need to be changed). Perform the following tasks to configure the eView Studio Matching Service.

- **Configuring the Standardization Structure** on page 73

- **Configuring the Match String** on page 75

When configuring the eView Studio Matching Service, keep in mind the information presented in **"Configuring the Matching Service" on page 37**.

7.5.1 Configuring the Standardization Structure

The standardization structure is configured in the **StandardizationConfig** section of the Match Field file, which is described in detail in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*. To configure the required fields for standardization and phonetic encoding, modify the standardization and phonetic encoding structures. This is described in detail in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*. The following sections provide additional guidelines and samples specific to standardizing address data.

Note: *In the default configuration, the rules defined for the address data type assume that all input fields must be parsed as well as normalized. Thus, there is no need to configure fields only for normalization.*

Standardization Structures

For address fields, the source fields in the standardization structure must include the fields predefined for parsing and normalization. This includes any fields containing street address information, which are parsed into the street address fields listed in **"The Object Structure" on page 61** (excluding the phonetic street name field). The target fields can include any of these parsed fields. Follow the instructions under "Defining Normalization" in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide* to define fields for normalization. For the **standardization-type** element, enter "Address" (for more information, see **"Match and Standardization Types" on page 34**). For a list of field IDs to use in the **standardized-object-field-id** element, see **Table 4 on page 30**.

A sample standardization structure for address data is shown below. This structure parses the first two lines of street address into the standard street address fields. Only the United States domain is defined in this structure.

```
free-form-texts-to-standardize>
   <group standardization-type="ADDRESS"
    domain-
selector="com.stc.eindex.matching.impl.SingleDomainSelectorUS">
      <unstandardized-source-fields>
         <unstandardized-source-field-name>Person.Address[*].Address1
         </unstandardized-source-field-name>
         <unstandardized-source-field-name>Person.Address[*].Address2
         </unstandardized-source-field-name>
      </unstandardized-source-fields>
      <standardization-targets>
         <target-mapping>
            <standardized-object-field-id>HouseNumber
            </standardized-object-field-id>
            <standardized-target-field-
name>Person.Address[*].HouseNumber
            </standardized-target-field-name>
         </target-mapping>
         <target-mapping>
            <standardized-object-field-id>RuralRouteIdentif
```

```
                </standardized-object-field-id>
                <standardized-target-field-
name>Person.Address[*].HouseNumber
                </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
                <standardized-object-field-id>BoxIdentif
                </standardized-object-field-id>
                <standardized-target-field-
name>Person.Address[*].HouseNumber
                </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
                <standardized-object-field-id>MatchStreetName
                </standardized-object-field-id>
                <standardized-target-field-
name>Person.Address[*].StreetName
                </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
                <standardized-object-field-id>RuralRouteDescript
                </standardized-object-field-id>
                <standardized-target-field-
name>Person.Address[*].StreetName
                </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
                <standardized-object-field-id>BoxDescript
                </standardized-object-field-id>
                <standardized-target-field-
name>Person.Address[*].StreetName
                </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
                <standardized-object-field-id>PropDesPrefDirection
                </standardized-object-field-id>
                <standardized-target-field-
name>Person.Address[*].StreetDir
                </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
                <standardized-object-field-id>PropDesSufDirection
                </standardized-object-field-id>
                <standardized-target-field-
name>Person.Address[*].StreetDir
                </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
                <standardized-object-field-id>StreetNameSufType
                </standardized-object-field-id>
                <standardized-target-field-
name>Person.Address[*].StreetType
                </standardized-target-field-name>
            </target-mapping>
            <target-mapping>
                <standardized-object-field-id>StreetNamePrefType
                </standardized-object-field-id>
                <standardized-target-field-
name>Person.Address[*].StreetType
                </standardized-target-field-name>
            </target-mapping>
        </standardization-targets>
    </group>
</free-form-texts-to-standardize>
```

Phonetic Encoding

When you match or standardize on street address fields, the street name should be specified for phonetic conversion (this is done by default). Follow the instructions under "Defining Phonetic Encoding" in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide* to define fields for phonetic encoding.

A sample of the **phoneticize-fields** element is shown below. This sample only converts the address street name. You can define additional fields for phonetic encoding.

```
<phoneticize-fields>
   <phoneticize-field>
      <unphoneticized-source-field-name>Person.Address[*].StreetName
      </unphoneticized-source-field-name>
      <phoneticized-target-field-
name>Person.Address[*].StreetName_Phon
      </phoneticized-target-field-name>
      <encoding-type>NYSIIS</encoding-type>
   </phoneticize-field>
</phoneticize-fields>
```

7.5.2 Configuring the Match String

For matching on street address fields, make sure the match string you specify in **MatchingConfig** contains all or a subset of the fields that contain the standardized data (the original text in street address fields are 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, follow the instructions under "Configuring the Match String" in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*. For the Sun SBME, each component of a street address has a different match type (specified by the **match-type** element). 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 **"Match and Standardization Types" on page 34**.

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>
```

Chapter 8

Business Names Data Type Configuration

Processing business name fields involves parsing, normalizing, and phonetically encoding certain fields prior to matching. This chapter describes the configuration files that define business name processing logic and provides instructions for modifying the Match Field file for processing business names.

What's in This Chapter

8.1 Business Name Matching Overview

Matching on the business name data type includes standardizing and matching on freeform business name fields. You can implement business name standardization and matching on its own or within a master index designed to process person information. For example, standardizing business name fields allows you to include these fields as search criteria, even though matching might not be performed against these fields.

The Sun SBME can create standardized and phonetic values for business names. Several configuration files are designed specifically to handle business names to define additional logic for the standardization and phonetic encoding process. These include reference files, a patterns file, and key type files. The Sun SBME can match on any field as long as the match type for the field is defined in the match configuration file (**matchConfigFile.cfg**).

The business name standardization files are common to all national domains, so no domain-specific configuration is required.

8.1.1 Business Name Processing Fields

When matching on freeform business names, not all fields in a record need to be processed by the Sun SBME. The match engine only needs to process fields that must be parsed, normalized, or phonetically converted, and the fields against which matching is

performed. These fields are defined in the Match Field file, and processing logic for each field is defined in the standardization and matching configuration files.

Match String Fields

The match string processed by the Sun SBME is defined by the match fields specified in the Match Field file. If you specify a "BusinessName" match type for any field in the eView Wizard, most of the parsed business name fields are automatically added to the match string in the Match Field file, including the name, organization type, association type, sector, industry, and URL. You can remove any of these fields from the match string.

The match engine can process any combination of fields you specify for matching. By default, the match configuration file 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 use any of the existing rows for matching or you can create new rows for the fields you want to match.

Standardized Fields

The Sun SBME expects that business name data will be provided in a freeform text field containing several components that must be parsed. The match engine is designed to parse these components, and to normalize and phonetically encode the business name. You can specify additional fields for phonetic encoding.

If you specify the "BusinessName" match type for any field in the eView Wizard, a standardization structure for that field is defined in the Match Field file. The fields defined as the target fields are listed in the next section, **"The Object Structure"**.

The Object Structure

For the default configuration of the business name data type, the address fields specified for standardization are parsed into several additional fields, one of which is also normalized. If you specify the appropriate match type in the eView Wizard, the following fields are automatically added to the object structure and database creation script.

- <*field_name*>_Name
- <*field_name*>_NamePhon
- <*field_name*>_OrgType
- <*field_name*>_AssocType
- <*field_name*>_Industry
- <*field_name*>_Sector
- <*field_name*>_Alias
- <*field_name*>_Url

  where <*field_name*> is the name of the field for which you specified business name matching. For example, if you specify the BusinessName match type for the

Company field, the fields automatically added to the structure include **Company_Name**, **Company_NamePhon**, **Company_OrgType**, and so on.

You can add these fields manually if you do not specify a match type in the eView Wizard.

8.2 Match Configuration for Business Names

The default match configuration file, **matchConfigFile.cfg**, defines several match types for the kinds of business name data typically included in the match string. You can customize the existing match types or create new match types for the data being processed. The following match types are typical for matching on business names.

- PrimaryName
- OrgTypeKeyword
- AssocTypeKeyword
- IndustrySectorList
- AliasList
- IndustryTypeKeyword
- URL

In addition, you can use any of these generic match types for matching on business names.

- String
- Date
- Numeric
- Integer
- Real
- Char
- pro
- Exac

This file appears under the **Match Engine** node of the eView Studio Project. For more information about the comparison functions used for each match type and how the weights are tuned, see **"The Match Configuration File" on page 21** and **"Match Configuration Comparison Functions" on page 100**.

8.3 Standardization Configuration for Business Names

Several configuration files are used to define business name processing logic for the Sun SBME. You can customize any of the configuration files described in this section to fit your data processing and standardization requirements. These files appear under the **Standardization Engine** node of the eView Studio Project.

bizConstants.cfg

The business constants file defines certain information about the standardization files used for processing business data, primarily the number of lines contained in each file. The number of lines specified must be equal to or greater than the number of lines actually contained in each file.

Table 25 lists and describes each parameter in the constants file. The files referenced by these parameters are described on the following pages.

Table 25 Business Constants File Parameters

Parameter	Description
cityMax	The maximum number of lines in the city or state key type file (**bizCityorStateTypeKey.dat**).
primaryMax	The maximum number of lines in the primary business names reference file (**bizCompanyPrimaryNames.dat**).
countryMax	The maximum number of lines in the country key type file (**bizCountryTypeKeys.dat**).
industryMax	The maximum number of lines in the industry key type file (**bizIndustryTypeKeys.dat**).
patternMax	The maximum number of lines in the business patterns file (**bizPatterns.dat**).
mergerMax	The maximum number of lines in the merged business name category file (**bizCompanyMergerNames.dat**).
adjectiveMax	The maximum number of lines in the adjective key type file (**bizAdjectiveTypeKeys.dat**).
orgMax	The maximum number of lines in the organization key type file (**bizOrganizationTypeKeys.dat**).
assocMax	The maximum number of lines in the association key type file (**bizAssociationTypeKeys.dat**).
genTermMax	The maximum number of lines in the general terms reference file (**bizBusinessGeneralTerms.dat**).
charsMax	The maximum number of lines in the special characters reference file (**bizRemoveSpecChars.dat**).
bizMaxWords	The maximum number of tokens allowed in the input business name. If no value is defined for this parameter, the default is the value set for the **words** parameter in the **personConstants.cfg** file.

bizAdjectivesTypeKeys.dat

The adjectives key type file defines adjectives commonly found in business names so the Sun SBME can recognize and process these values as a part of the business name. This file contains one column with a list of commonly used adjectives, such as General, Financial, Central, and so on.

You can modify or add entries in this file as needed. Following is an excerpt from the **bizAdjectivesTypeKeys.dat** file.

```
DIGITAL
DIRECTED
DIVERSIFIED
EDUCATIONAL
ELECTROCHEMICAL
ENGINEERED
EVOLUTIONARY
```

```
EXTENDED
FACTUAL
FEDERAL
```

bizAliasTypeKeys.dat

The alias key type file lists business name acronyms and abbreviations along with their standardized names so the Sun SBME can recognize and process these values correctly. You can add entries to the alias key type file using the following syntax.

```
alias standardized-name
```

Table 26 describes the columns in the **bizAliasTypeKeys.dat** file.

Table 26 Alias Key Type File

Column	Description
alias	An abbreviation or acronym commonly used in place of a specific business name.
standardized-name	The normalized version of the alias name.

Following is an excerpt from the **bizAliasTypeKeys.dat** file.

```
BBH                 BARTLE BOGLE HEGARTY
BBH                 BROWN BROTHERS HARRIMAN
IBM                 INTERNATIONAL BUSINESS MACHINE
IDS                 INCOMES DATA SERVICES
IDS                 INSURANCE DATA SERVICES
IDS                 THE INTEGRATED DECISION SUPPORT GROUP
IDS                 THE INTERNET DATABASE SERVICE
CAL-TECH            CALIFORNIA INSTITUTE OF TECHNOLOGY
```

bizAssociationTypeKeys.dat

The association key type file lists business association types along with their standardized names so the Sun SBME can recognize and process these values correctly. You can add entries to the association key type file using the following syntax.

```
association-type standardized-type
```

Table 27 describes the columns in the **bizAssociationTypeKeys.dat** file.

Table 27 Association Type Key Table

Column	Description
association-type	A common association type for businesses, such as Partners, Group, and so on.
standardized-type	The standardized version of the association type. If this column contains a name instead of a zero, that name must also be listed in a different entry as an association type with a standardized form of “0”.

Following is an excerpt from the **bizAssociationTypeKeys.dat** file.

```
ASSOCIATES          0
BANCORP             0
```

```
BANCORPORATION      BANCORP
COMPANIES           0
GP                  GROUP
GROUP               0
PARTNERS            0
```

bizBusinessGeneralTerms.dat

The general terms reference file lists terms commonly used in business names. This file is used to identify terms that indicate a business, such as bank, supply, factory, and so on, so the Sun SBME can recognize and process the business name.

This file contains one column that lists common terms in the business names you process. You can add entries as needed. Below is an excerpt from the **bizBusinessGeneralTerms.dat** file.

```
BUILDING
CITY
CONSUMER
EAST
EYE
FACTORY
LATIN
NORTH
SOUTH
```

bizCityorStateTypeKeys.dat

The city or state key type file lists various cities and states that might be used in business names. It also classifies each entry as a city (CT) or state (ST) and indicates the country in which the city or state is located. This enables the Sun SBME to recognize and process these values correctly. You can add entries to the city or state key type file using the following syntax.

```
city-or-state type country
```

Table 28 describes the columns in the **bizCityorStateTypeKeys.dat** file.

Table 28 City or State Key Type File

Column	Description
city-or-state	The name of a city or state used in business names.
type	An indicator of whether the value is a city or state. “CT” indicates city and “ST” indicates state.
country	The country code of the country in which the city or state is located.

Following is an excerpt from the **bizCityorStateTypeKeys.dat** file.

```
ADELAIDE                CT   AU
ALABAMA                 ST   US
ALASKA                  ST   US
ALGIERS                 CT   DZ
AMSTERDAM               CT   NL
ARIZONA                 ST   US
```

```
ARKANSAS                  ST   US
ASUNCION                  CT   PY
ATHENS                    CT   GR
```

bizCompanyFormerNames.dat

The business former name reference file provides a list of common company names along with names by which the companies were formerly known so the Sun SBME can recognize a business when a record processing a record containing a previous business name. You can add entries to the business former name table using the following syntax.

```
former-name current-name
```

Table 29 describes each column in the **bizCompanyFormerNames.dat** file.

Table 29 Business Former Name Reference File

Column	Description
former-name	One of the company's previous names.
current-name	The company's current name.

Below is an excerpt from the **bizCompanyFormerNames.dat** file.

```
HELLENIC BOTTLING                    COCA-COLA HBC
INTERNATIONAL PRODUCTS               THE TERLATO WINE
ORGANIC FOOD PRODUCTS                SPECTRUM ORGANIC PRODUCTS
SUTTER HOME WINERY                   TRINCHERO FAMILY ESTATES
```

bizCompanyMergerNames.dat

The merged business name category file provides a list of companies whose name changed because of a merger along with the name of the company after the merge. It also classifies the business names into industry sectors and sub-sectors. This enables the Sun SBME to recognize the current company name and determine the sector of the business. You can add entries to the business merger name file using the following syntax.

```
former-name/merged-name sector-code
```

Table 30 describes each column in the **bizCompanyMergerNames.dat** file.

Table 30 Business Merger Name Category File

Column	Description
former-name	The name of the company whose name was not kept after the merger.
merged-name	The name of the company whose name was kept after the merger.
sector-code	The industry sector code of the business. Sector codes are listed in the **bizIndustryCategoriesCode.dat** file.

Below is an excerpt from the **bizCompanyMergerNames.dat** file.

```
DUKE/FLUOR DANIEL                                20005
FAULTLESS STARCH/BON AMI                         09004
FIND/SVP                                         10013
FIRST WAVE/NEWPARK SHIPBUILDING                  27005
GUNDLE/SLT                                       19020
HMG/COURTLAND                                    23004
J BROWN/LMC                                      10014
KORN/FERRY                                       10020
LINSCO/PRIVATE LEDGER                            14005
```

bizCompanyPrimaryNames.dat

The primary business name reference file provides a list of companies by their primary name. It also classifies the business names into industry sectors and sub-sectors. This enables the Sun SBME to determine the correct value of the sector field when parsing the business name. You can add entries to the primary business name file using the following syntax.

```
primary-name sector-code
```

Table 31 describes the columns in the **bizCompanyPrimaryNames.dat** file.

Table 31 Business Primary Name Reference File

Column	Description
primary-name	The primary name of the company.
sector-code	The industry sector code of the business. Sector codes are listed in the **bizIndustryCategoriesCode.dat** file.

Below is an excerpt from the **bizCompanyPrimaryNames.dat** file.

```
BROTHER INTERNATIONAL                            12006
BRYSTOL-MYERS SQUIBB                             11005
BURLINGTON COAT FACTORY                          24003
BURLINGTON NORTHERN SANTA FE                     27005
BV SOLUTIONS                                     06012
CABLEVISION                                      26001
CABOT                                            04006
CADENCE                                          06010
CAMPBELL                                         22006
CAPITAL BLUE CROSS                               17001
```

bizConnectorTokens.dat

The connector tokens reference file defines common values (typically conjunctions) that connect words in business names. For example, in the business name “Nursery of Venice”, “of” is a connector token. This helps the Sun SBME recognize and process the full name of a business by indicating that the token connects two parts of the full name.

This file contains one column that lists the connector tokens in the business names you process. You can add entries as needed. Below is an excerpt from the **bizConnectorTokens.dat** file.

```
AN
DE
DES
DOS
```

```
LA
LAS
LE
OF
THE
```

bizCountryTypeKeys.dat

The country key type file lists countries and continents, along with their abbreviations and assigned nationalities. For continents, the abbreviation is "CON" to separate them from countries. This enables the Sun SBME to recognize and process these values as countries or continents. You can add entries to the country key type file using the following syntax.

```
country abbreviation nationality
```

Table 32 describes the columns in the **bizCountryTypeKeys.dat** file.

Table 32 Country Key Type Files

Column	Description
country	The name of a country or continent.
abbreviation	The common abbreviation for the specified country. The abbreviation for a continent is always "CON".
nationality	The nationality assigned to a person or business originating in the specified country.

Following is an excerpt from the **bizCountryTypeKeys.dat** file.

```
AMERICA                    CON  AMERICAN
AFRICA                     CON  AFRICAN
EUROPE                     CON  EUROPEAN
ASIA                       CON  ASIAN
AFGHANISTAN                AF   AFGHAN
ALBANIA                    AL   ALBANIAN
ALGERIA                    DZ   ALGERIAN
```

bizIndustryCategoryCode.dat

The industry sector reference file lists and groups various industry sectors and sub-sectors, and includes an identification code for each type so the Sun SBME can identify and process the industry sectors for different businesses. You can add entries to the industry sector reference file using the following syntax.

```
sector-code industry-sector
```

Table 33 describes each column in the **bizIndustryCategoryCode.dat** file.

Table 33 Industry Sector Reference File

Column	Description
sector-code	The identification code of the specified sector. The first two numbers of each code identify the general industry sector; the last three number identify a sub-sector.

Table 33 Industry Sector Reference File

Column	Description
industry-sector	A description of the industry category. This is written in the format "<sector> - <sub-sector>", where <sector> is a general category of industry types, and <sub-sector> is a specific industry within that category.

```
02006    Automotive&TransportEquipment-RecreationalVehicles
02007    Automotive & Transport Equipment - Shipbuilding & Related
Services
02008    Automotive & Transport Equipment - Trucks, Buses & Other
Vehicles
03001    Banking - Banking
04001    Chemicals - Agricultural Chemicals
04002    Chemicals - Basic & IntermediateChemicals & Petrochemicals
04003    Chemicals - Diversified Chemicals
04004    Chemicals - Paints, Coatings & Other Finishing Products
04005    Chemicals - Plastics & Fibers
04006    Chemicals - Specialty Chemicals
05001    Computer Hardware - Computer Peripherals
05002    Computer Hardware - Data Storage Devices
05003    Computer Hardware - Diversified Computer Products
```

bizIndustryTypeKeys.dat

The industry key type file is used to standardize the value of the Industry field into common industries to which businesses belong so the Sun SBME can recognize and process the industry types for different businesses. You can add entries to the industry key type file using the following syntax.

```
industry-type standardized-form sectors
```

Table 34 describes each column in the **bizIndustryTypeKeys.dat** file.

Table 34 Industry Key Type File

Column	Description
industry-type	The original value of the industry type in the input record.
standardized-form	The normalized version of the industry type. If this column contains a name instead of a zero, that name must also be listed in a different entry as an industry type with a standardized form of "0".
sectors	The industry categories of the specified industry type. These values correspond to the sector codes listed in the industry sector file (**bizIndustryCategoryCode.dat**). You can list as many categories as apply for each type, but they must be entered with a space between each and no line breaks, and they must correspond to an entry in the industry sector file.

Below is an excerpt from the **bizIndustryTypeKeys.dat** file.

```
TECH                TECHNOLOGY          05001-05007
TECHNOLOGIES        TECHNOLOGY          05001-05007
TECHNOLOGY          0                   05001-05007
TECHSYSTEMS         0                   05001-05007
TELE PHONE          TELEPHONE           16005
TELE PHONES         TELEPHONES          16005
TELEVISION          TV                  11013  21014
TELECOM             0                   16005  26006  26009  26010
TELECOMM            TELECOMMUNICATION   16005  26006  26008
TELECOMMUNICATION   0                   16005  26006  26008
```

bizOrganizationTypeKeys.dat

The organization key type file is used to standardize the value of the Organization field into common organizations to which businesses belong. This helps the Sun SBME recognize and process the organization types for different businesses. You can add entries to the organization key type file using the following syntax.

```
original-type standardized-form
```

Table 35 describes each column in the **bizOrganizationTypeKeys.dat** file.

Table 35 Organization Key Type File

Column	Description
original-type	The original value of the organization field in an input record.
standardized-form	The normalized version of an organization type. A zero (0) in this field indicates that the value in the first column is already in its standardized form. If this column contains a name instead of a zero, that name must also be listed in a different entry as an original type with a standardized form of "0".

Below is an excerpt from the **bizOrganizationTypeKeys.dat** file.

```
INC                 INCORPORATED
INCORPORATED        0
KG                  0
KK                  0
LIMITED             0
LIMITED PARTNERSHIP 0
LLC                 0
LLP                 0
LP                  LIMITED PARTNERSHIP
LTD                 LIMITED
```

bizPatterns.dat

The business patterns file defines multiple formats expected from the business name input fields along with the standardized output of each format. The patterns and output appear in two-row pairs in this file, as shown below.

```
4 PNT AST SEP-GLC ORT
PNT AST DEL ORT
```

The first line describes the input pattern and the second describes the output pattern using tokens to denote each component. The supported tokens are described in **"Business Name Tokens" on page 88**. A number at the beginning of the first line indicates the number of components in the given business name format. You can modify this file using the following syntax.

```
length input-pattern
output-pattern
```

Table 36 lists and describes the syntax components.

Table 36 Business Patterns File Components

Component	Description
length	The number of business name components in the input field.
input-pattern	Tokens that represent a possible input pattern from the unparsed business name fields. Each token represents one component. For more information about address tokens, see **"Business Name Tokens" on page 88**.
output-pattern	Tokens that represent the output pattern for the specified input pattern. Each token represents one component. For more information about business name tokens, see **"Business Name Tokens" on page 88**.

Below is an excerpt from the **bizPatterns.dat** file.

```
4 PNT AST SEP-GLC ORT
PNT AST DEL ORT

4 NFG AJT SEP-GLC ORT
PNT PNT DEL ORT

4 NF AJT SEP-GLC ORT
PNT PNT DEL ORT

4 CST IDT NF ORT
PNT PNT PNT ORT

4 PNT AJT SEP-GLC ORT
PNT PNT DEL ORT
```

Business Name Tokens

The business patterns file uses tokens to denote different components in a business name, such as the primary name, alias type key, URL, and so on. The file uses one set of tokens for input fields and another set for output fields. The tokens indicate the type key files to use to determine the appropriate values for each output field. You can use only the predefined tokens to represent business name components; the Sun SBME does not recognize custom tokens.

Table 37 lists and describes each input token; Table 38 lists and describes each output token.

Table 37 Business Name Input Pattern Tokens

Pattern Identifier	Description
CTT	A connector token
PNT	A primary name of a business
PN-PN	A hyphenated primary name of a business
BCT	A common business term
URL	The URL of the business' web site
ALT	A business alias type key (usually an acronym)
CNT	A country name
NAT	A nationality
CST	A city or state type key
IDT	An industry type key
IDT-AJT	Both an industry and an adjective type key
AJT	An adjective type key
AST	An association type key
ORT	An organization type key
SEP	A separator key
NFG	Generic term, not recognized as a specific business name component, with an internal hyphen
NF	Generic term, not recognized as a specific business name component
NFC	A single character, not recognized as a specific business name component
SEP-GLC	A joining comma (a *glue type separator*)
SEP-GLD	A joining hyphen (a *glue type separator*)
AND	The text "and"
GLU	A glue type key, such as a forward slash, connecting two parts of a business name component
PN-NF	A business primary name followed by a hyphen and a generic term that is not recognized as a specific business name component
NF-PN	A generic term that is not recognized as a specific business name component, followed by a hyphen and a recognized business primary name

Table 37 Business Name Input Pattern Tokens

Pattern Identifier	Description
NF-NF	Two generic terms, not recognized as specific business name components and separated by a hyphen

Table 38 Business Name Output Pattern Tokens

Pattern Identifier	Description
PNT	The primary name of the business
URL	The URL of the business
ALT	The alias type key of the business (usually an acronym)
IDT	The industry type key of the business
AST	The association type key of the business
ORT	The organization type key of the business
NF	A generic term not recognized as a business name component

bizRemoveSpecChars.dat

The special characters reference file lists certain characters that should be removed from a business name prior to processing the field, which typically include punctuation marks such as exclamation points, parenthesis, and so on. This enables the Sun SBME to recognize the business name.

This file contains one column that lists the characters to be removed from the business names you process. You can add entries as needed. Below is an excerpt from the **bizRemoveSpecChars.dat** file.

```
[
]
{
}
<
>
/
?
```

8.4 Modifying Business Name Configuration Files

To customize the Sun SBME configuration files for processing business names, you can modify any of the files described in this chapter using the text editor provided in Enterprise Designer. Before modifying the match configuration file, review the information provided in Chapter 4 **"Matching Configuration Files"** and Appendix B

"Match Configuration Comparison Functions". Make sure a thorough data analysis has been performed to determine the best fields for matching, and the best comparison functions to use for each field.

Updating most standardization files is a straight-forward process. Make sure to follow the syntax guidelines provided in **"Standardization Configuration for Business Names" on page 79**. If you add rows to any standardization files, make sure to modify the corresponding parameter in the business constants file (**bizConstants.cfg**). Before making any changes to the patterns file, make sure you understand the tokens used to represent business name field components.

8.5 Configuring the eView Studio Matching Service

To ensure correct processing of business names, you must customize the eView Studio Matching Service. This includes modifying the Match Field file to support the fields on which you want to match, to standardize the appropriate fields, and to specify the Sun SBME as the match and standardization engine (by default, the Sun SBME is already specified so this does not need to be changed). Perform the following tasks to configure the eView Studio Matching Service.

- **Configuring the Standardization Structure** on page 91
- **Configuring the Match String** on page 93

When configuring the eView Studio Matching Service, keep in mind the information presented in **"Configuring the Matching Service" on page 37**.

8.5.1 Configuring the Standardization Structure

The standardization structure is configured in the **StandardizationConfig** section of the Match Field file, which is described in detail in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*. To configure the required fields for standardization and phonetic encoding, modify the standardization and phonetic encoding structures. This is described in detail in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*. The following sections provide additional guidelines and samples specific to standardizing business names.

Note: *In the default configuration, the rules defined for the business data type assume that all input fields must be parsed as well as normalized. Thus, there is no need to configure fields only for normalization.*

Standardization Structures

For business name fields, the source fields in the standardization structure must include the fields predefined for parsing and normalization. This includes any fields containing business name information, which are parsed into the business name fields listed in **"The Object Structure" on page 78** (excluding the phonetic business name field). The target fields can include any of these parsed fields. Follow the instructions under "Defining Standardization" in Chapter 6 of the *Sun SeeBeyond eView Studio*

Configuration Guide to define fields for normalization. For the **standardization-type** element, enter "BusinessName" (for more information, see **"Match and Standardization Types" on page 34**). For a list of field IDs to use in the **standardized-object-field-id** element, see **Table 4 on page 30**.

A sample standardization structure for business name data is shown below. This structure parses a business name field into the standard business name fields. Note that there is no domain selector specified, which would normally default to the United States domain; however, since business names are not domain dependent, it is irrelevant here.

```
<free-form-texts-to-standardize>
   <group standardization-type="BusinessName">
      <unstandardized-source-fields>
         <unstandardized-source-field-name>Company.Name
         </unstandardized-source-field-name>
      </unstandardized-source-fields>
      <standardization-targets>
         <target-mapping>
            <standardized-object-field-id>PrimaryName
            </standardized-object-field-id>
            <standardized-target-field-name>Company.Name_Name
            </standardized-target-field-name>
         </target-mapping>
         <target-mapping>
            <standardized-object-field-id>OrgTypekeyword
            </standardized-object-field-id>
            <standardized-target-field-name>Company.Name_OrgType
            </standardized-target-field-name>
         </target-mapping>
         <target-mapping>
            <standardized-object-field-id>AssocTypeKeyword
            </standardized-object-field-id>
            <standardized-target-field-name>Company.Name_AssocType
            </standardized-target-field-name>
         </target-mapping>
         <target-mapping>
            <standardized-object-field-id>IndustrySectorList
            </standardized-object-field-id>
            <standardized-target-field-name>Company.Name_Sector
            </standardized-target-field-name>
         </target-mapping>
         <target-mapping>
            <standardized-object-field-id>IndustryTypeKeyword
            </standardized-object-field-id>
            <standardized-target-field-name>Company.Name_Industry
            </standardized-target-field-name>
         </target-mapping>
         <target-mapping>
            <standardized-object-field-id>AliasList
            </standardized-object-field-id>
            <standardized-target-field-name>Company.Name_Alias
            </standardized-target-field-name>
         </target-mapping>
         <target-mapping>
            <standardized-object-field-id>Url
            </standardized-object-field-id>
            <standardized-target-field-name>Company.Name_URL
            </standardized-target-field-name>
         </target-mapping>
      </standardization-targets>
   </group>
</free-form-texts-to-standardize>
```

Phonetic Encoding

When you match on business name fields, the name field should be specified for phonetic conversion (by default, the eView Wizard defines this for you). Follow the instructions under "Defining Phonetic Encoding" in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide* to define fields for phonetic encoding.

A sample of the **phoneticize-fields** element is shown below. This sample only converts the business name. You can define additional fields for phonetic encoding.

```
<phoneticize-fields>
   <phoneticize-field>
      <unphoneticized-source-field-name>Company.Name_Name
      </unphoneticized-source-field-name>
      <phoneticized-target-field-name>Company.Name_NamePhon
      </phoneticized-target-field-name>
      <encoding-type>NYSIIS</encoding-type>
   </phoneticize-field>
</phoneticize-fields>
```

8.5.2 Configuring the Match String

For matching on business name fields, make sure the match string you specify in **MatchingConfig** 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, follow the instructions under "Configuring the Match String" in Chapter 6 of the *Sun SeeBeyond eView Studio Configuration Guide*. For the Sun SBME, each data type has a different match type (specified by the **match-type** element). 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 **"Match and Standardization Types" on page 34**.

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>
```

Appendix A

Fine-tuning Weights and Thresholds

Each eView Studio implementation is unique, typically requiring extensive data analysis to determine how to best configure the structure and matching logic of the master index. This chapter provides an overview of the process of fine-tuning the matching logic in the match configuration file and fine-tuning the match and duplicate thresholds.

What's in This Appendix

A.1 Data Analysis Overview

A thorough analysis of the data to be shared with the master index 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 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.

A.2 Customizing the Match Configuration and Thresholds

There are three primary steps to customizing how records are matched in the master index.

A.2.1 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 index. 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.

A.2.2 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 **"Matching Weight Formulation" on page 16** 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.

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. Table 39 provides a sample agreement/disagreement configuration for matching on person data. As you can see, the range of match weights generated for the master index with this configuration is from -36 to +38.

Table 39 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:

LOG_2(m_prob/u_prob)

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

LOG_2((1-m_prob)/(1-u_prob))

Table 40 below illustrates a sample of 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 the master index with this configuration is from -35.93 to +38

Table 40 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 **Appendix B "Match Configuration Comparison Functions"**. Choose the comparison functions that best suit how you want the match fields to be processed.

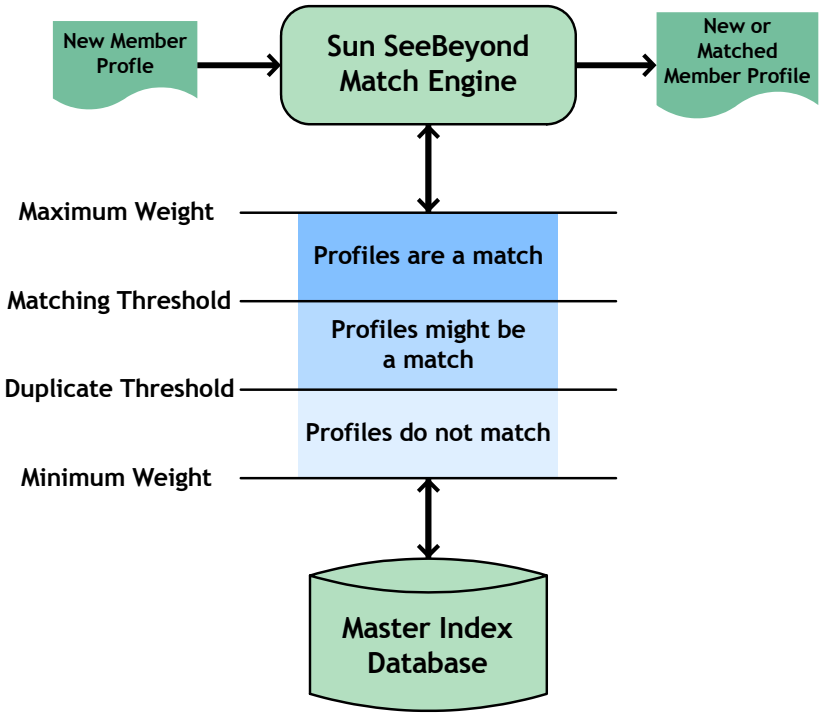
A.2.3 Determining the Weight Thresholds

Weight thresholds tell the master index how to process incoming records based on the matching probability weights generated by the Sun SBME. Two parameters in the Threshold configuration file provide the master index 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.

Figure 1 illustrates the match and duplicate thresholds in comparison to total composite match weights.

Figure 1 Weight Thresholds



Specifying the Weight Thresholds

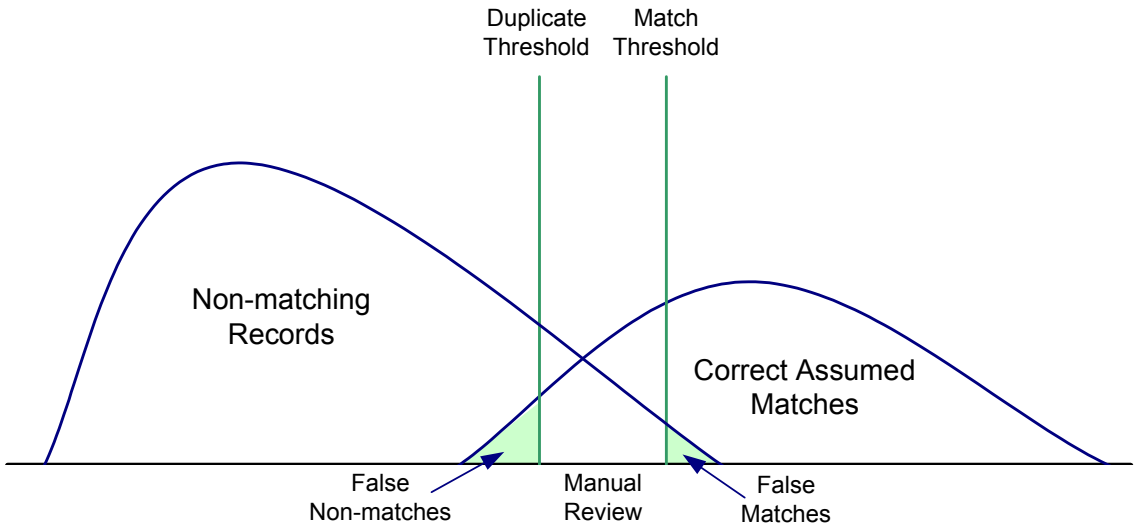
There are many techniques for determining the initial settings for the match and duplicate thresholds. This section discusses two methods. The first method, the weight distribution method, is based on the calculation of the error rates of false matches and false non-matches from analyzing the distribution spectrum of all the weighted pairs. This is the standard method, and is illustrated in Figure 2. 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 index can be classified into three categories: matches, non-matches, and potential matches. In general, the distribution of records is similar to the graph shown in Figure 2. 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.

Figure 2 Weight Distribution Chart



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 39 on page 96**, 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 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.

Appendix B

Match Configuration 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 Sun SBME matches records.

What's in This Appendix

B.1 Comparison Functions

There are six primary types of comparison functions used by the Sun SBME. The following types of comparison functions are available.

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 on page 23** for information about how the parameters in the match configuration file affect the outcome of the comparator functions. For example, these parameters define how null fields are handled and what the actual agreement and disagreement weights will be.

B.1.1 Bigram Comparators

The Sun SBME 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 String Comparator (b1)

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

Advanced Bigram String Comparator (b2)

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

B.1.2 String Comparators

The Sun SBME provides the following uncertainty comparison functions for comparing string fields. Most uncertainty comparison functions are generic, but three comparison functions are designed for specific types of information (first name, last name, and house number).

- Generic String Comparator (u)
- Advanced Generic String Comparator (ua)
- Simplified String Comparator (us)
  - Simplified String Comparator - FirstName (uf)
  - Simplified String Comparator - LastName (ul)
  - Simplified String Comparator - HouseNumber (un)
- Simplified String Comparator (usu)

Generic String Comparator (u)

This is the standard uncertainty comparison function, which processes string fields as described above. As more differences are found between two fields, the agreement weight decreases non-linearly. Thus, the agreement weight can remain high for several differences, but will drop sharply at a certain point. This comparison function takes no parameters.

The uncertainty 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.

Advanced Generic String Comparator (ua)

This comparison function 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, "1" the number versus "l" the letter) or keypunch errors (for example, "S" versus "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.

Simplified String Comparator (us)

This comparison function is a custom version of a generic string comparison function. It is similar to the basic uncertainty comparison function, **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 basic uncertainty function, the simplex function 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.

Simplified String Comparator - FirstName (uf)

This comparison function is designed specifically for matching on first name fields, and is based on the simplex uncertainty comparison function, **us**. This comparison function analyzes the string and then adjusts the weight based on statistical data. This comparison function takes no parameters.

Simplified String Comparator - LastName (ul)

This comparison function is designed specifically for matching on last name fields, and is based on the simplex uncertainty comparison function, **us**. This comparison function analyzes the string and then adjusts the weight based on statistical data. This comparison function takes no parameters.

Simplified String Comparator - HouseNumber (un)

This comparison function is designed specifically for matching on house numbers, and is based on the simplex uncertainty comparison function, **u**. This comparison function analyzes the string and then adjusts the weight based on statistical data. This comparison function takes no parameters.

Language-specific String Comparator (usu)

This comparison function is a custom version of a generic string comparison function. It is similar to the simplex uncertainty comparison function, **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 41.

Table 41 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

B.1.3 Exact char-by-char Comparator (c)

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

B.1.4 Numeric Comparators

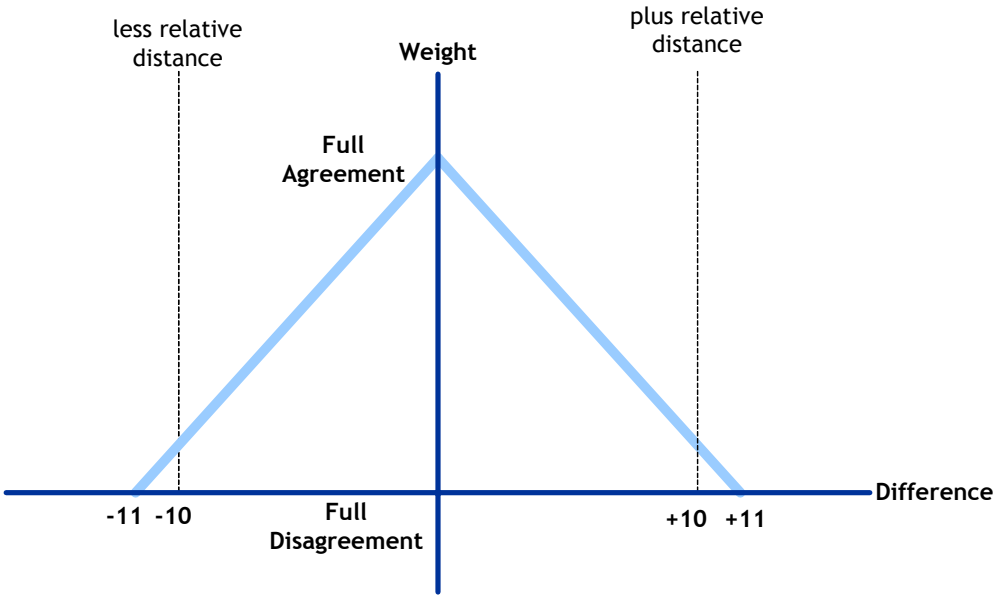
The Sun SBME provides several comparison functions for matching on numeric fields.

- Generic Number Comparator (n)
- Integer Comparator (nI)
- Real Number Comparator (nR)
- Alpha-Numeric Comparator (nS)

All but the nS comparison function can perform 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).

Figure 3 illustrates how the weight is decreased as the difference between the two compared fields reaches the relative distance. In this diagram, the relative distance is **10** and the light blue line represents the agreement weight. When the difference between two fields reaches **11** (relative distance plus one), the fields are considered a non-match and are given the full disagreement weight.

Figure 3 Numeric Relative Distance Comparison



Generic Number Comparator (n)

This is a basic numeric comparison function, processing numeric fields as described above. It accepts the parameters listed in Table 42.

Table 42 n, nI, and nR Comparison Function Parameters

Parameter	Description
distance-or-string	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.
relative-distance	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).

Integer Comparator (nI)

This numeric comparison function matches specifically on integers and accepts the parameters listed in Table 42.

Real Number Comparator (nR)

This numeric comparison function matches specifically on real numbers and accepts the parameters listed in Table 42.

Alpha-numeric Comparator (nS)

This numeric comparison function is designed specifically for matching on numeric strings and is very useful for matching social security numbers or other unique identifiers. This is the only numeric comparator that can compare alphanumeric values rather than just numeric values. It accepts the parameters listed in Table 43.

Table 43 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" on page 23**).

Table 43 nS Comparison Function Parameters

Parameter	Description
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.

B.1.5 Date Comparators

The Sun SBME 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.

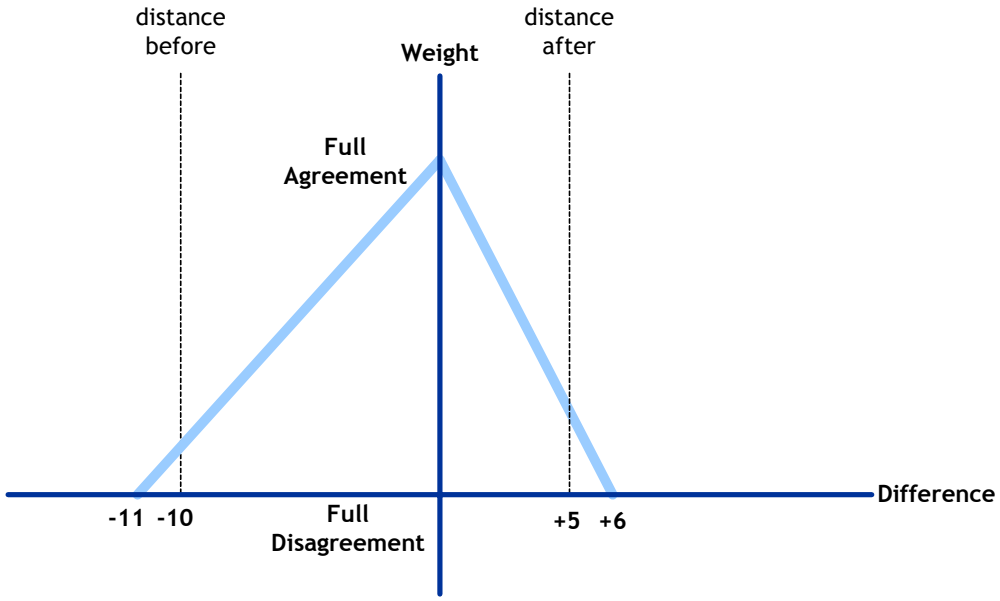
- Date Comparator - Year only (dY)
- Date Comparator - Month-Year (dM)
- Date Comparator - Day-Month-Year (dD)
- Date Comparator - Hour-Day-Month-Year (dH)
- Date Comparator - Min-Hour-Day-Month-Year (dm)
- Date Comparator - Sec-Min-Hour-Day-Month-Year (ds)

As with the numeric comparison functions, the date comparison functions can use either a direct string comparison or a relative distance calculation. 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.

Figure 4 illustrates how the weight is decreased as the difference between the two compared fields reaches either the before or after relative distance. In this diagram, the before relative distance is **11**, the after relative distance is **5**, and the light blue line represents 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.

Figure 4 Date Relative Distance Comparison



The date comparison functions take the parameters listed in Table 44.

Table 44 Date Comparison Function Parameters

Parameter	Description
distance-or-string	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.
distance-before	The number of units prior to the reference date/time for which two date fields can still be considered a match.
distance-after	The number of units following the reference date/ time for which two date fields can still be considered a match.

Date Comparator - Year only (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 - Month-Year (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 - Day-Month-Year (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 - Hour-Day-Month-Year (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 - Min-Hour-Day-Month-Year (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 - Sec-Min-Hour-Day-Month-Year (ds)

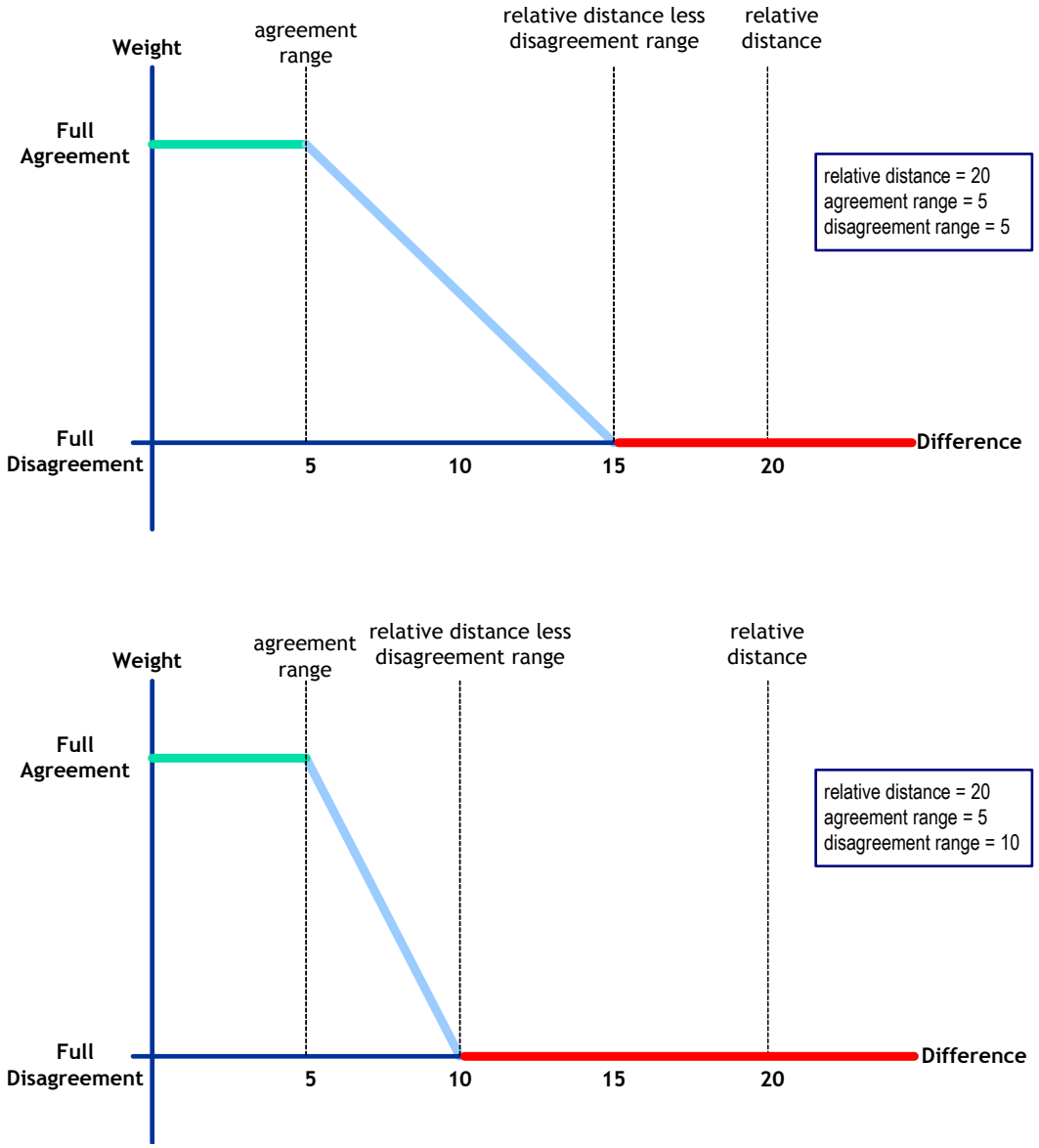
This date comparison function takes the second, minute, hour, day, month, and year into account for matching. If relative distance calculation is specified, the relative distance is specified in seconds.

B.1.6 Prorated Comparator (p)

The prorated comparison function 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.

Figure 5 illustrates how weighting is adjusted per the parameters you define. In these diagrams, the green line indicates full agreement, the light blue line indicates prorated agreement, and the red line indicates full disagreement. The diagrams illustrate how increasing the disagreement weight causes the prorated agreement weight to decrease more sharply.

Figure 5 Prorated Linear Adjustment Comparison



The prorated comparison functions takes the parameters listed in Table 45.

Table 45 Prorated Comparison Function Parameters

Parameter	Description
relative-distance	The greatest difference between two numbers at which they can still be considered a match or partial match.
agreement-range	The greatest difference between two numbers at which they are considered a full match. This number must be less than the relative distance.

Table 45 Prorated Comparison Function Parameters

Parameter	Description
disagreement-range	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).

B.2 Comparison Function Options

The options listed below can be used in conjunction with the above string comparison functions to give them more functionality. For example, you can use an 'ufI' string comparison function that refers to the first name comparison function with the possibility to switch fields if the first one does not match.

- **I** - This is a major inversion option that allows for field transpositions. If two compared fields do not match, this option lets the match engine know to compare the original field in the first record with the next field in the second record. If those fields agree, the match engine assigns the full agreement weight and switches the fields.
- **i** - This is a minor inversion option similar to the major inversion (I) described above. This option only assigns one-half of the full agreement weight if the transposed fields match.
- **x** - If two or more fields with this option match, their weight is doubled; but if any of the fields with this option disagree, the weight is not doubled.
- **k** - This option can be used with the 'x' option to give more importance to one field. Specifying this option on a field tells the match engine to double the match weight for a sub-group of fields with the 'x' option by doubling the weight as soon as it comes to the field with the 'k' option.

Glossary

agreement weight
A positive weight assigned to a match field if the values agree between two fields.

Blocking Query
Also known as a blocker query, this is used during matching to search the database for possible matches to a new or updated record. Blocking queries can also be used for searches done from the EDM.This query makes multiple passes against the database using different combinations of criteria, which are defined in the Candidate Select file.

Candidate Select file
The eView Studio configuration file that defines the queries you can perform from the Enterprise Data Manager (EDM) and the queries that are performed for matching.

candidate selection
The process of performing the blocking query for match processing. See *Blocking Query*.

candidate selection pool
The group of possible matching records returned by the blocking query. These records are weighed against the new or updated record to determine the probability of a match.

comparison function
A command specific to the SeeBeyond Match Engine that specifies how two fields are compared. Comparison functions are specified for each match field in the match configuration file.

disagreement weight
A negative weight assigned to a match field if the field values disagree between two fields.

duplicate threshold
The matching probability weight at or above which two records are considered to potentially represent the same entity. See also *matching threshold*.

enterprise object
A complete object representing a specific entity, including the SBR and all associated system objects.

ePath
A definition of the location of a field in an eView Studio object. Also known as the *element path*.

EUID
The enterprise-wide unique identification number assigned to each object profile in the master index. This number is used to cross-reference objects and to uniquely identify each object throughout your organization.

eView Studio Manager Service
An eView Studio component that provides an interface to all eView Studio components and includes the primary functions of the master index. This component is configured by the Threshold file.

field IDs
An identifier for each field that is defined in the standardization engine and referenced from the Match Field file.

master index
A database application that centralizes and cross-references information on specific objects in a business organization.

Match Field File
An eView Studio configuration file that defines normalization, parsing, phonetic encoding, and the match string for an instance of eView Studio. The information in this file is dependent on the type of data being standardized and matched.

match pass
During matching several queries are performed in turn against the database to retrieve a set of possible matches to an incoming record. Each query execution is called a match pass.

match string
The data string that is sent to the match engine for probabilistic weighting. This string is defined by the match system object defined in the Match Field file and must match the string defined in the match engine configuration files.

match type
An indicator specified in the **MatchingConfig** section of the Match Field file that tells the match engine which rules in the match configuration file to use for determine matching weights between records.

matching probability weight
An indicator of how closely two records match one another. The weight is generated using matching algorithm logic, and is used to determine whether two records represent the same object. See also *duplicate threshold* and *matching threshold*.

Matching Service
An eView Studio component that defines the matching process. This component is configured by the Match Field file.

matching threshold
The lowest matching probability weight at which two records can be considered a match of one another. See also *duplicate threshold* and *matching probability weight*.

matching weight *or* match weight
See *matching probability weight*.

normalization
A standardization process by which the value of a field is converted to a standard version, such as changing a nickname to a common name.

object
A component of an object profile, such as a company object, which contains all of the demographic data about a company, or an address object, which contains information about a specific address type for the company.

object profile
A set of information that describes characteristics of one enterprise object. A profile includes identification and other information about an object and contains a single best record and one or more system records.

parsing
A component of the standardization process by which a freeform text field is separated into its individual components, such as separating a street address field into house number, street name, and street type fields.

phonetic encoding
A standardization process by which the value of a field is converted to its phonetic version.

phonetic search
A search that returns phonetic variations of the entered search criteria, allowing room for misspellings and typographic errors.

potential duplicates
Two different enterprise objects that have a high probability of representing the same entity. The probability is determined using matching algorithm logic.

probabilistic weighting
A process during which two records are compared for similarities and differences, and a matching probability weight is assigned based on the fields in the match string. The higher the weight, the higher the likelihood that two records match.

probability weight
See *matching probability weight*.

Query Builder
An eView Studio component that defines how queries are processed. The user-configured logic for this component is contained in the Candidate Select file.

SBR
See *single best record*.

single best record
Also known as the SBR, this is the best representation of an entity's information. The SBR is populated with information from all source systems based on the survivor strategies defined for each field and child object. It is a part of an entity's enterprise object and is recalculated each time a system record is updated.

standardization
The process of parsing, normalizing, or phonetically encoding data in an incoming or updated record. Also see *normalization*, *parsing*, and *phonetic encoding*.

standardization type
An indicator specified in the **StandardizationConfig** section of the Match Field file that tells the SeeBeyond Match Engine how to standardize information.

survivor calculator
The logic that determines which field values or child objects from the available source systems are used to populate the SBR.

survivorship
Refers to the logic that determines which field values are used to populate the SBR. The survivor calculator defines survivorship.

system
A computer application within an organization where information is entered about objects and that shares information with the master index (such as a registration system). Also known as a source system, local system, or external system.

system object
A record received from a local system. The fields contained in system objects are used in combination to populate the SBR. The system objects for one entity are part of that entity's enterprise object.

Threshold file
An eView Studio configuration file that specifies duplicate and match thresholds, EUID generator parameters, and which blocking query defined in the Candidate Select file to use for matching.

Index

Numerics

A

B

C

D

E

F

G

H

I

N

O

P

R

S

T

U

V

W

X