MATCHING AND LINKING
USER GUIDE

STEP Trailblazer 8.1

Table of Contents

Matching and Linking

The STEP Matching and Linking component offers powerful functionality for identifying and handling duplicate product, entity, asset, and classification objects in STEP.

The matching and linking functionality is most commonly used for:

- Cleanup operations (during data migration, for example)
- Matching of the same item from multiple suppliers
- Matching of internally enriched records with data from external data suppliers
- Consolidation of information from different systems

Strategy for Identifying Duplicates

Before configuring the matching and linking functionality you must define what qualifies two or more objects as duplicates.

- For products, you could be looking for objects that have the same EAN number, or the same or similar manufacturer and manufacturer part number information.
- For customer data, you could be looking for people that have the same or similar names, combined with the same or similar addresses.

There are also many cases far more complex than those listed above.

When dealing with datasets of thousands or millions of objects, comparing each and every object with every other object is not a viable solution. In order to limit the number of comparisons, **Match Codes** must be generated for the applicable objects and then evaluated and matched together via a **Matching Algorithm**. The system can be configured to handle those matches in two ways: by simply merging matching records, or by generating **Golden Records**, which combines the data of all confirmed matching objects into one 'true' record.

For more information on match codes and matching algorithms, see the **Matching and Linking Components** section of the **Matching and Linking** documentation.

For more information of golden records, see the **Golden Records Match Action** section of the **Matching and Linking** documentation.

Matching and Linking in Web UI

Several different matching and linking tasks can be completed in Web UI. See the list below for relevant Web UI topics:

- **Potential Duplicates List** - Found in the **Web UI Setup and User Guide / Web User Interfaces** documentation.
- **Merging Confirmed Matches** - Found in the **Web UI Setup and User Guide / Web User Interfaces** documentation.
- **Configuring a Deduplication Clerical Review** - Found below in the **Matching and Linking** documentation.

Note: This list will be updated periodically as additional Web UI topics are developed.

Matching and Linking Components

The Matching and Linking functionality relies on three underlying components: Match Codes, Matching Algorithms, and an Event Processor. Together, these components can evaluate a dataset for duplicate objects and initiate necessary maintenance actions. To accomplish this, match codes are generated for objects in a dataset and the matching algorithm evaluates those match codes for potential duplicates.

How those duplicates are handled depends on how the matching algorithm is configured. The main two methods of handling duplicates are:

- **Identify Duplicates Match Action** - For more information, see the **Identify Duplicates Match Action** documentation.
- **Golden Records Match Action** - For more information, see the **Golden Records Match Action** documentation.

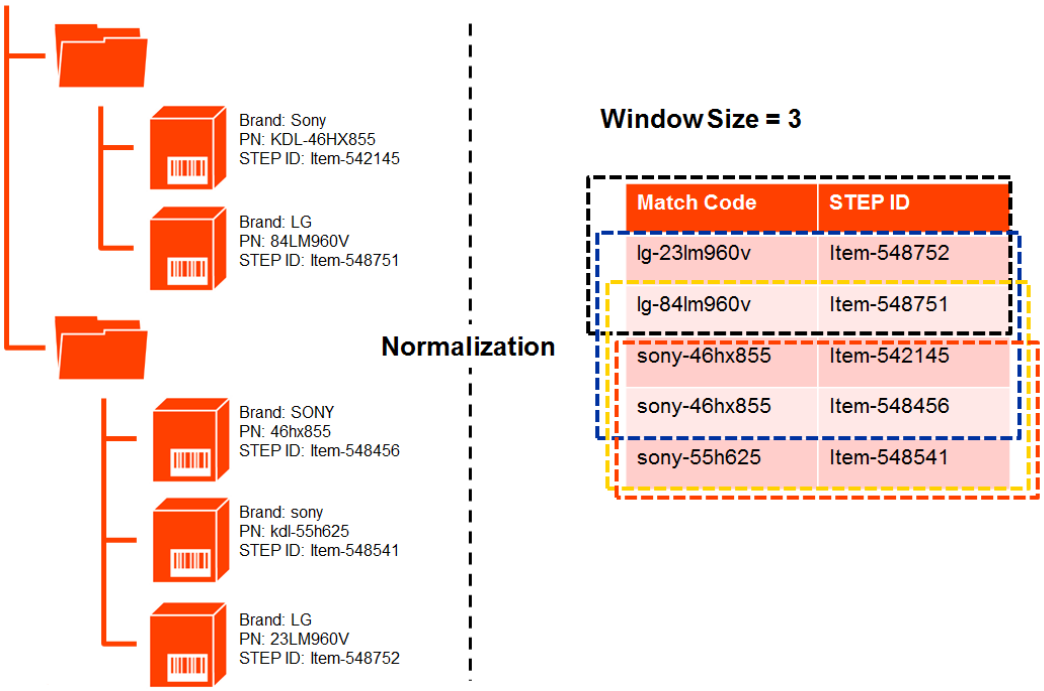
The event processor triggers events which can be acted upon by a business rule or an outbound integration endpoint.

Match Codes

A match code is essentially a string (i.e., a text) representing an object. The string is derived using either STEP functions or JavaScript (drawing upon the functionality exposed in the public Java API). As an example, a match code could simply be composed of the values of two attributes concatenated.

Once generated, these match codes populate an alphabetically sorted table in the system. Rather than comparing every object with every other object in the dataset, only objects with match codes close to each other in the sorted list will be compared. This dramatically limits the number of comparisons required as they are linearly proportional with the number of objects. The number of comparisons an individual object can make is based on the Window Size setting.

Example



Using the data from the image above, with a window size of '3', every product object would be compared to the object with the match code immediately prior to / following it in the list – the exception being the two products for which identical match codes are generated ('Item-542145' and 'Item-548456'). As the window size setting is for unique match codes, here 'Item-542145' would be compared to both 'Item-548751', 'Item-548456' and 'Item-548541' while 'Item-548456' would be compared to 'Item-548751', 'Item-542145', and 'Item-548541'.

Disregarding that match code values can be identical, the function for calculating the number of comparisons with this approach can be approximated to:

Total comparisons required = ((Window Size -1) / 2) * Number of objects

With this approach, you have to be very careful when defining the match code. Objects with match codes that are alphabetically far from one another are not likely to be compared (unless the window size is set very high, which then defeats the purpose).

For more information on match codes and how they are configured, see the **Configuring Match Codes** documentation.

Matching Algorithm

Match codes simply limit the number of comparisons that the matching algorithm has to make, and once the codes have be generated, the matching algorithm can compare objects in the dataset. Comparing two objects results in a number between 0% and 100%, indicating how similar the objects are to each other.

There are many ways to arrive at this metric. In some cases, you could only be interested in exact matches, and the algorithm would be fairly straightforward. For example, if the social security number for two customer objects is

the same, then it is very likely these are duplicates and the matching algorithm should return 100% (or 0% if the numbers are not the same).

In many cases, however, you cannot work with exact matches, but instead, will have to deal with approximate matches, or a combination of exact and approximate matches. It could be that for the customer object mentioned above, you don't have a social security number available and will have to identify duplicates based on names, mail addresses, phone numbers, and street addresses. These pieces of data can have variations, even in objects that represent the same real world entity. Names and addresses could be spelled differently, middle names could be left out, abbreviations could be used in names and addresses, the customers could be registered with different phone numbers or mail addresses, and so on.

For more information on matching algorithms and how they are configured, see the **Configuring Matching Algorithms** documentation.

Event Processor

While the match code and matching algorithm define the data and handling that is needed, an event processor is required to launch the associated business rule or OIEP.

Once set up, an Event Processor can be configured to monitor the system for actionable events on specified objects and then regenerate match codes and/or run matching algorithms in response. For example, consider an object that is subject to a matching algorithm. When the match code assignment or data on that object is approved, the approval can trigger the event processor to regenerate the match code for that object and run the algorithm. Alternatively, events could be passed to the event processor via a republish business rule as part of a workflow or integration.

Event processors keep a background process log, so you can determine when events were processed and what actions were taken in response. Additionally, event processor performance measurements are available on the Statistics tab for both Matching Algorithms and Match Code configurations.

For more information on matching event processors and how they are configured, see the **Configuring Matching Event Processor** documentation.

Golden Records Match Action

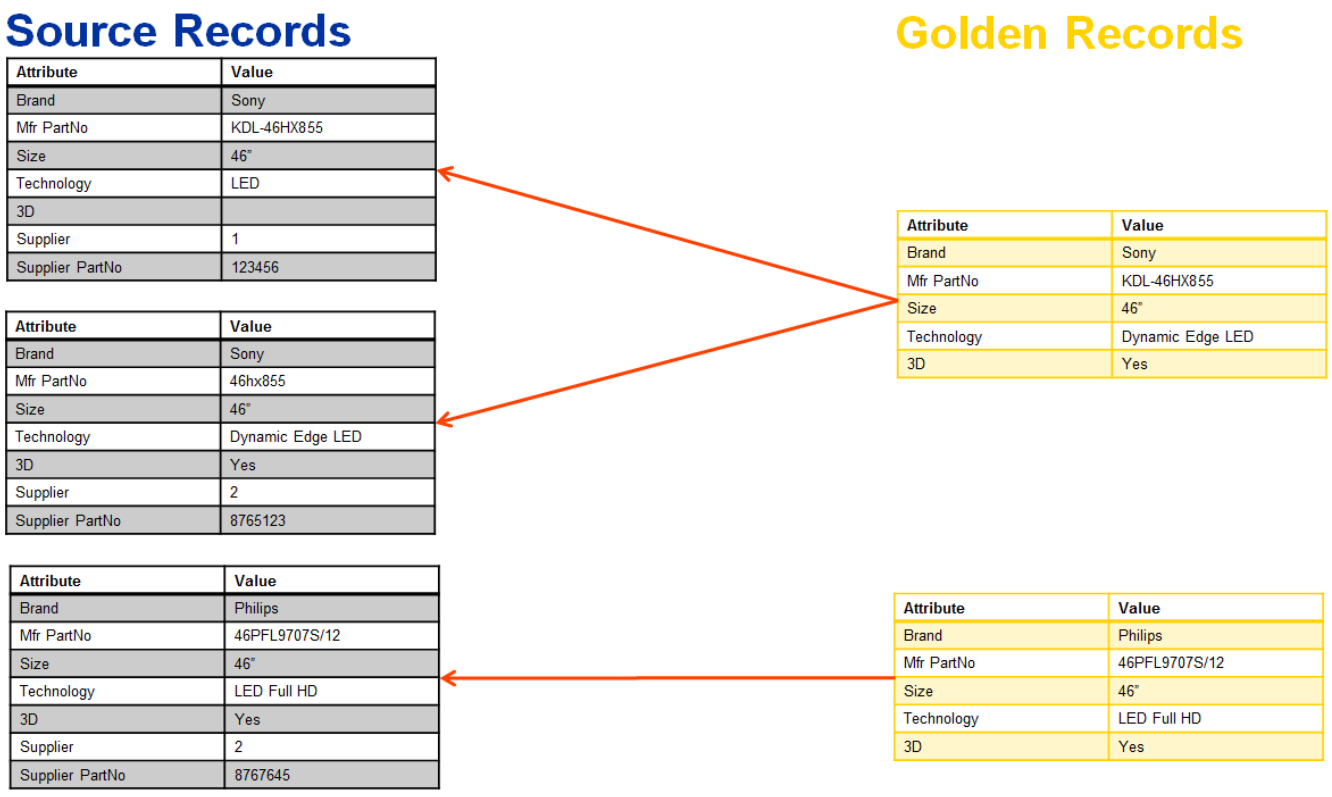
Instead of merging duplicate objects, a matching algorithm can be configured to create golden records using the most reliable data from those objects. Any duplicate objects that contribute data to the golden record are called 'Source Objects', and are referenced by the golden record. Once a golden record is created, the source objects remain unchanged.

After a matching algorithm has been applied, a source object will always have a golden record referencing it. For example, if you have 100 source objects and find that two of the objects are duplicates, running a matching algorithm creates 99 golden records. 98 of the golden records will have a reference to a single source object each, while the last one will reference two source objects.

Note: Any golden record that only references a single object is called a 'singleton'.

Because golden records simply represent the source objects, their data should never be maintained manually. Furthermore, golden records are updated every time the matching logic is applied, and may be deleted or changed to point to other objects as part of the process. Given their temporary nature, IDs of golden records should never have any significance in STEP or in external systems. Instead, golden records must be considered temporary representations of the source objects in the dataset.

In the diagram below, two golden records are created based on three source objects (two are duplicates):



With this functionality, only golden records will be made available to external systems – not a mixture of golden records and source objects.

Note that there may be times when more than two objects are identified as duplicates. Since the system only compares two objects at a time, transitive closure is used:

If A = B, and B = C, then A = C

Had A, B, and C been source objects, they would all have been referenced from the same golden record.

For more information of golden records and their configuration options, see the **Configuring Golden Records** documentation.

Matching and Linking Configuration

Considering the robust nature of the matching and linking functionality, configuration can be complex. Before beginning the configuration it is necessary to develop a complete deduplication strategy. To do so requires intimate knowledge of the data in question. To that end, data profiles are a great way to examine your data and can aid in developing your strategy.

For more information about deduplication strategies, see the **Configuration Example - Basic** and **Configuration Example - Advanced** sections of the **Matching and Linking** documentation.

For more information about data profiles, see the **About Data Profiling** section of the **Data Profiling** documentation.

Once a strategy has been developed, users will have to configure underlying attributes for use in the configuration, as well as potentially writing JavaScript for dictating the logic of the matching algorithm. Typical configuration steps include:

- Configuring the Component Model
- Configuring the Setup Group
- Configuring the Match Codes
- Configuring the Matching Algorithm
  - May include configuring Golden Records
- Configuring the Event Processor
  - Including Matching Algorithm / Match Code maintenance tasks

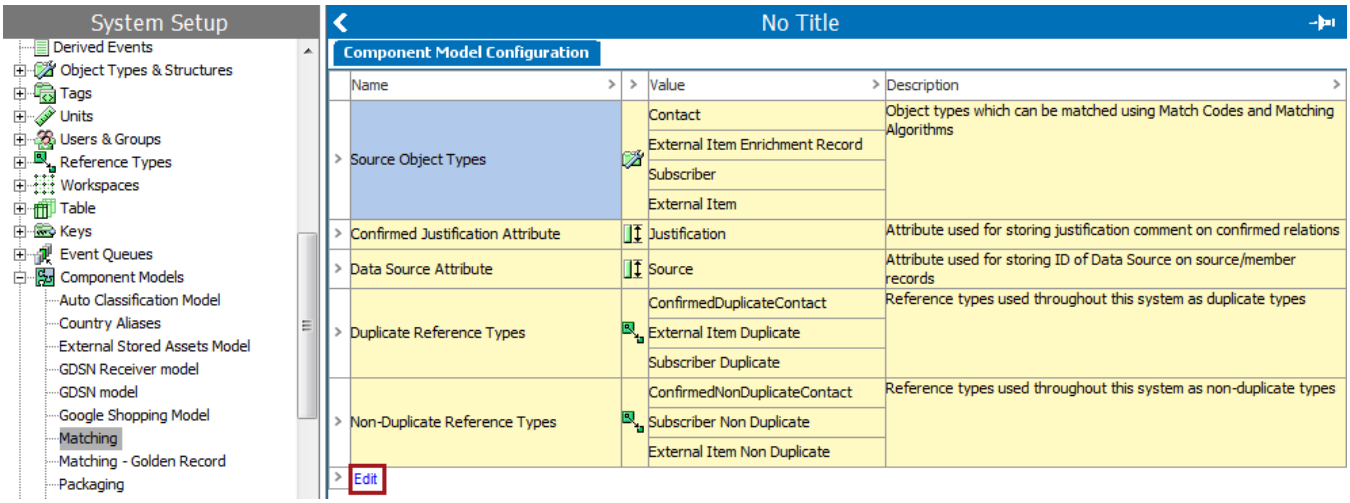
Matching and Linking Component Model Configuration

Before match codes can be generated and matching algorithms applied, the **Matching Component Model** must be configured. The component model determines which objects, attributes, and references are relevant to your configuration, and how they apply. There are two component models available for matching and linking: 'Matching' and 'Matching - Golden Record'.

Matching Component Model

The Matching component model defines all the source objects that are allowed to be matched.

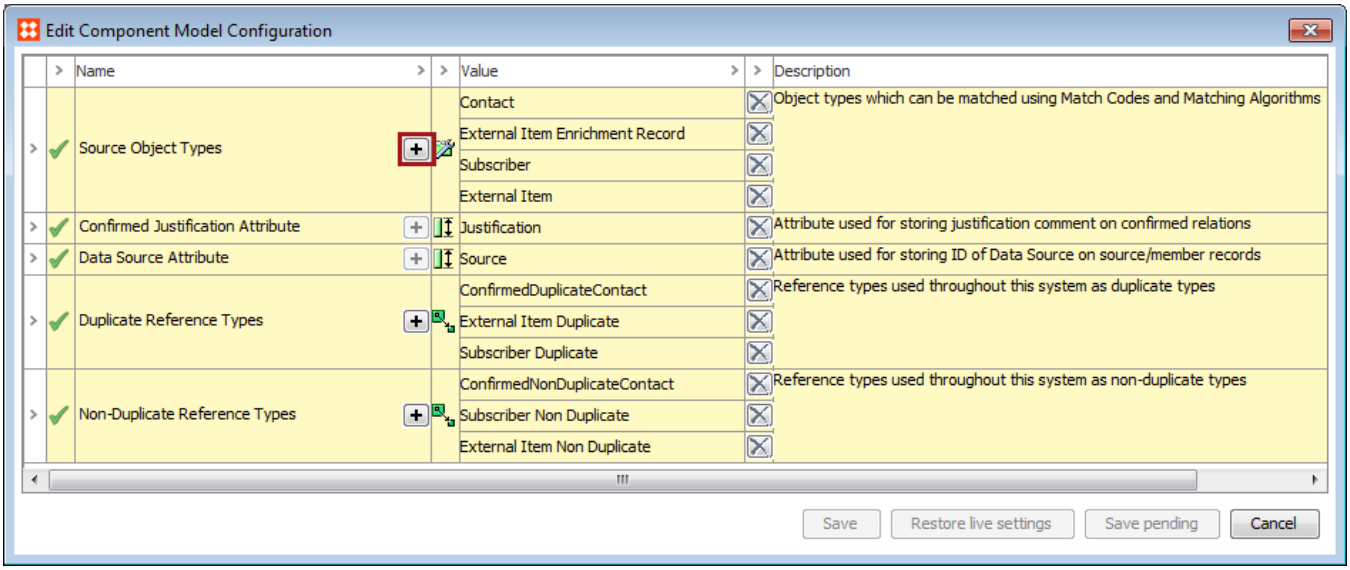
1. In **System Setup**, expand 'Component Models', and click on the 'Matching' node.
2. On the 'Component Model Configuration' tab, click the **Edit** link.



3. Click the 'plus' button for the relevant component aspect to display the selection dialog, and then choose to add an object, attribute, or reference:

   - **Source Object Types** – Select the object types that need to be matched. Only the object types configured here can be used as object types for match codes. On objects of these types, the 'Matching' tab is automatically enabled. The 'Matching' tab shows match code values, potential duplicates, and confirmed relations for the selected object.
   - **Confirmed Justification Attribute** – Select a description attribute that is valid for all reference types specified in the 'Duplicate Reference Types' and 'Non-Duplicate Reference Types' fields. This attribute stores a description explaining why two objects are marked as duplicates or non-duplicates.
   - **Data Source Attribute** – Select one or more description attributes that are valid for all source object types specified in the 'Source Object Types' field. This attribute contains the source ID of the source objects. If you select more than one attribute in this field, then exactly one of these attributes must be valid per source object type selected in the 'Source Object Types' field.
   - **Duplicate Reference Types** – Select one or more reference types to be used to store the manually maintained confirmed duplicate references. These references store the reason for confirming two source objects as duplicates as specified in the attribute selected in the 'Confirmed Justification Attribute' field. All the selected reference types must have exactly one valid attribute from the 'Confirmed Justification Attribute' field. Only the duplicate reference types you select here can be used as 'Duplicate Type' on a matching algorithm. In a typical scenario, you will have different duplicate reference types for different matching algorithms. If you reuse duplicate reference type between algorithms, then the confirmed duplicates will be reused between those algorithms as well.
   - **Non-Duplicate Reference Types** - Select one or more reference types used by the system for storing the manually maintained confirmed non-duplicate references. These references store the reason for confirming two source objects as non-duplicates as specified in the attribute selected in the 'Confirmed Justification Attribute' field. All the selected reference types must have exactly one valid attribute from the 'Confirmed Justification Attribute' field. Only reference types selected here can be used as 'Non-Duplicate Type' on a matching algorithm. In a typical scenario, you will have different duplicate reference types for different

matching algorithms. If you reuse non-duplicate reference type between algorithms, then the confirmed non-duplicates will be reused between those algorithms as well.



Click the 'X' button to remove the relevant object, attribute, or reference from the component model.

A green check mark will appear if the applicable row has a valid configuration.

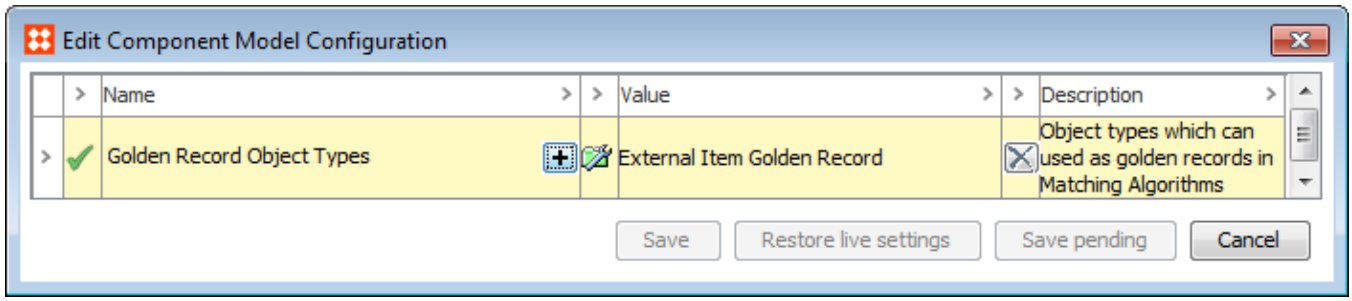
4. Click **Save** to save changes.

Note: If you need to navigate away from the configuration dialog and some of the rows are not yet valid (they have an 'X' instead of a check mark), click **Save pending** to save your work.

Matching - Golden Record Component Model

The Matching - Golden Record component model lists all the golden record object types in the system.

1. In **System Setup**, expand 'Component Models' and click on the 'Matching - Golden Records' node.
2. On the 'Component Model Configuration' tab, click the **Edit** link.
3. Click the 'plus' button in the 'Golden Record Object Types' field, and in the selection dialog, choose the object types you want to use for golden records in the system. Only the object types that you select here can be used as object types for golden records. Furthermore, on objects of these object types the Golden Record tab is automatically enabled. The Golden Record tab shows the golden record together with its member records.



Click the 'X' button to remove the relevant value from the component model.

A green check mark will appear if the component model has a valid configuration.

4. Click **Save** to save changes.

Note: If you need to navigate away from the configuration dialog and the component model is not yet valid (it has an 'X' instead of a check mark), click **Save pending** to save your work.

Configuring a Match Codes and Matching Algorithms Setup Group

Match codes and matching algorithms are first class objects in **System Setup**, living below 'Setup Groups' in the workbench. In order to create new match codes and matching algorithms, however, a setup group must exist to house them.

On a system where the setup is not in place, first define a new setup group object type, make match codes and matching algorithms legal children, and then create an instance of the setup group object type in the System Setup.

For more information, see the **Setup Group** section of the **System Setup / STEP Super User Guide** documentation.

Configuring Match Codes

Before configuring match codes there are several things to consider:

- Closely examine the data before configuring a match code. The data profiling tool can provide a lot of valuable information, and if you are planning to use a specific attribute in the match code, always check to which degree the attribute is populated – if values are missing on a lot of objects, it is likely not a good candidate, or at least should not be used alone, as objects with 'empty' match codes will not be included in the database table.
- When working with match codes composed from several pieces of data, always put the most significant data first. For instance, if deduplicating address objects, put the ZIP code before street and street number, as ZIP codes are geographic, standardized, and mutually exclusive – which most effectively separates your addresses into discrete objects.
- Be sure to normalize the data used in match codes. If, for instance, a manufacturer name is often abbreviated, your match code definition should handle this so that the name is represented the same way in the match codes, regardless of whether it is abbreviated on the source object or not.
- The match code can be just a single piece of data like an EAN number. Furthermore, if you are only interested in comparing objects that have identical EAN numbers, a Window Size of 1 can be used. This means that only objects with identical Match Code values will be compared.
- Several match codes can be generated per source object. STEP functions can resolve to a list of multiple match codes, and in JavaScript, an array can be returned. In both cases, each element will be a separate match code. As a simple example, this could be useful if you were to identify duplicates among customer entity objects, each having a name and an address attribute. Here, the match code could be a concatenation of address and name, but with this approach, you would not be able to find duplicates for customers who

have moved, as the match codes would likely be placed too far from each other. Instead, each object could be represented with two match codes: one for 'Name' and one for 'Address', meaning that the objects could be compared both due to having similar names – and having similar addresses (a hardcoded prefix should be added first to prevent comparisons across the two domains).

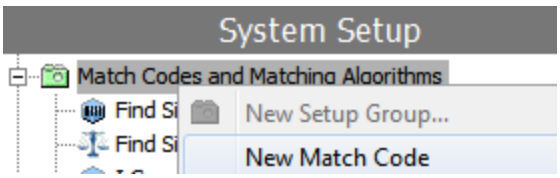
- Ideally, you should generate match codes that allow you to perform matching with a Window Size of 1, but where there are still not too many objects that share the same match code.

For more information on match codes, see the **Matching and Linking Components** section of the **Matching and Linking** documentation.

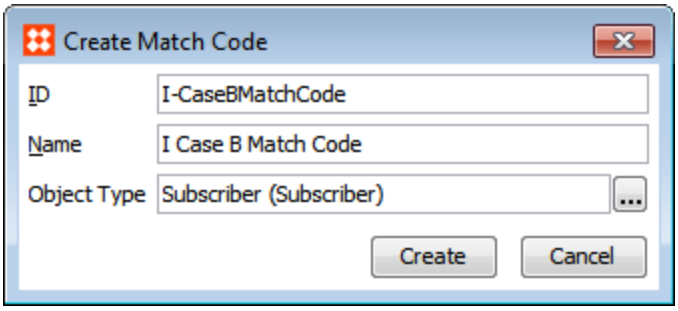
For more information on data profiling, see the **Data Profiles** section of the **Data Profiling** documentation.

Configuration

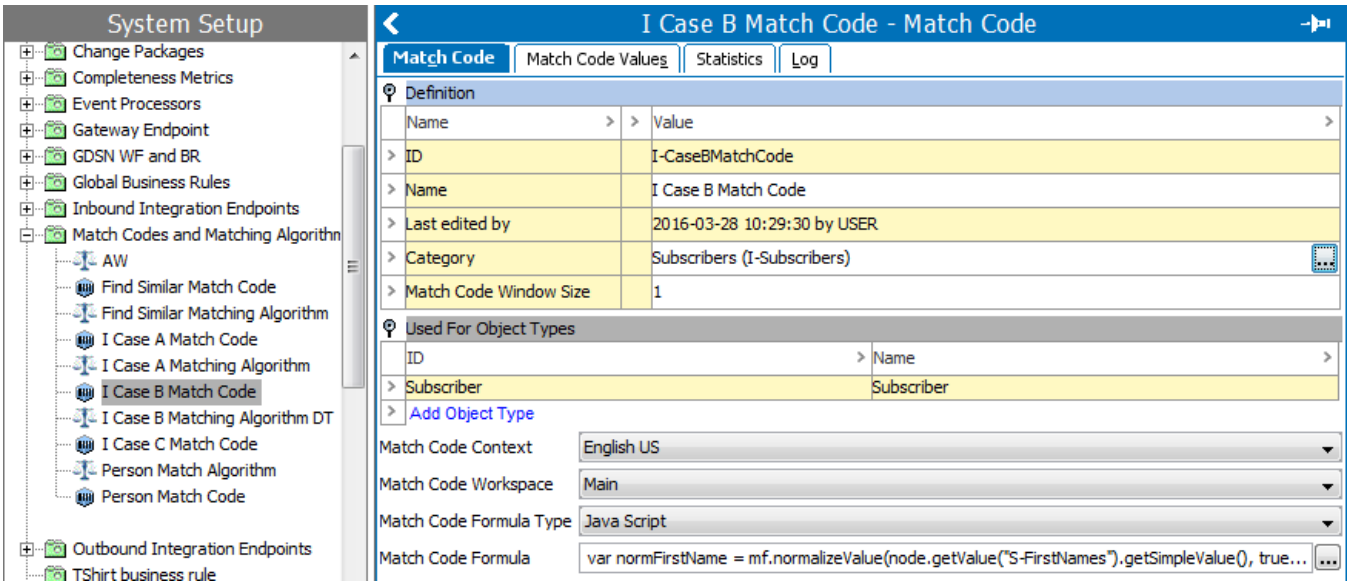
1. In **System Setup**, right-click the node configured to house match codes and select 'New Match Code'.



2. In the 'Create Match Code' dialog, define an ID and name for the match code, specify an object type for which this match code applies, and click **Create**. Additional object types can be identified in the Match Code editor (see details below).



3. On the new Match Code editor, navigate to the 'Match Code' tab and click the ellipsis button **(…)** in the 'Category' field. In the selector that appears, select a node to indicate which objects will have match codes generated.



4. In the 'Match Code Window Size' field, specify the window size to be used by the matching algorithm.
5. If additional object types are required, in the 'Used For Object Types' section, use the **Add Object Type** link and selector to identify more object types for the match code.
6. In the 'Match Code Context' field, specify in which context to run the match code formula. This is only required if the data is dimension dependent.
7. In the 'Match Code Workspace' field, specify in which workspace to run the match code formula.
8. In the 'Match Code Formula Type' field, specify 'Java Script' or 'Calculated' as the format
9. In the 'Match Code Formula' field, then click the ellipsis button **(...)** to open up the formula editor and add your match code formula.

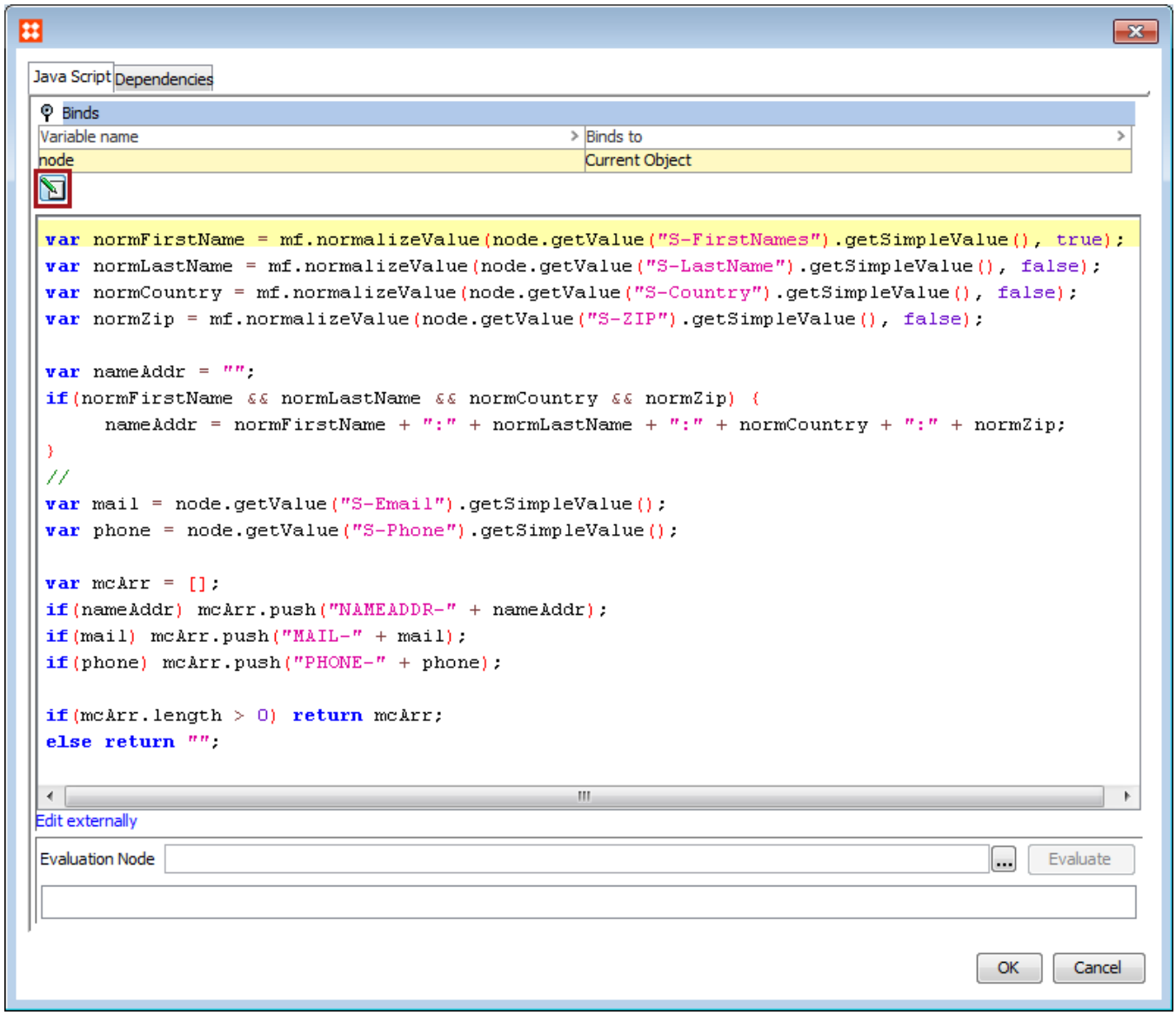
Binds for Match Code Formulas

It is also possible to make use of attributes and values that are created offline, by binding them in the match code formula. This is used in cases of offline matching or matching records on import. Once inside the 'Match Code Formula' editor, open the 'Binds' flipper, and click the 'Edit Binds' button. You can declare variables and bind them to a variety of STEP elements / objects, as determined by the selected formula type.

JavaScript Match Code Formula

When using JavaScript, the current object should be bound to a variable. The ultimate goal should be to return the match code value of an object from the JavaScript. If a string is returned, it will be used as a match code value. If a JavaScript array is returned, all values in the array will be used as match code values for that object. Additional utility functions for match codes can be accessed by binding 'Matching Functions' to, for example, the context variable in JavaScript or by binding 'Lookup Table Home' to, for example, 'lth':

Method	Description
`context.soundex('Stibo')`	Returns the Soundex. For more information, see **Text Functions** in the **Calculated Attributes** documentation.
`context.metaphone3('Stibo')`	Returns the primary value for the Metaphone 3. For more information, see **Text Functions** in the **Calculated Attributes** documentation.
`context.metaphone3alternate ('Stibo')`	Returns the alternate value for the Metaphone 3. For more information, see **Text Functions** in the **Calculated Attributes** documentation.
`lth.getLookupTableValue ('<asset-id>', 'LookupValue')`	Returns the converted value from asset-id lookup table. If this is not found, it returns 'LookupValue'. For more information, see the **Creating a Transformation Lookup Table** section of the **Data Quality** documentation.



Calculated Attribute Match Code Formula

When defining the formula via the calculated attribute language, all functions are available. An object's match code value can be a single string derived from the value of the formula, or it can be a list where all the values in the list are used as match code values for that object. For more information on the available STEP functions, see the **Calculated Attribute Functions** section of the **System Setup / Super User Guide** documentation.

Configuring Matching Algorithms

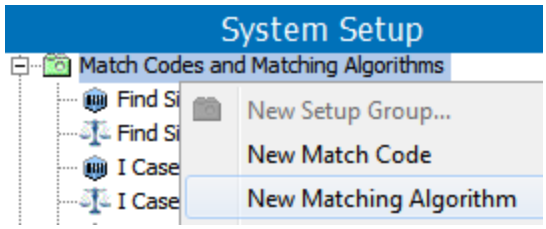
Before configuring a matching algorithm, ensure that you have developed a thorough matching and linking strategy. Among the many choices available, you must determine which string comparison method to use and what criteria it should follow.

For more information, see the **Match Criteria** section of the **Matching and Linking** documentation.

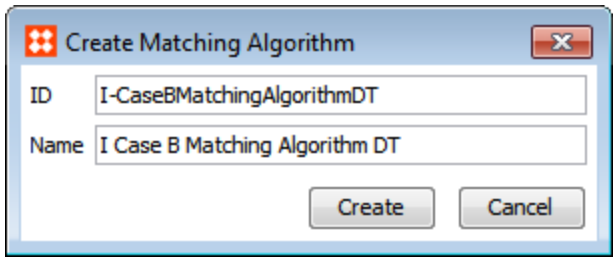
For additional information on configuring the accompanying matching algorithms and match codes, see the **Matching and Linking Components** and **Configuring Match Codes** documentation.

Configuration

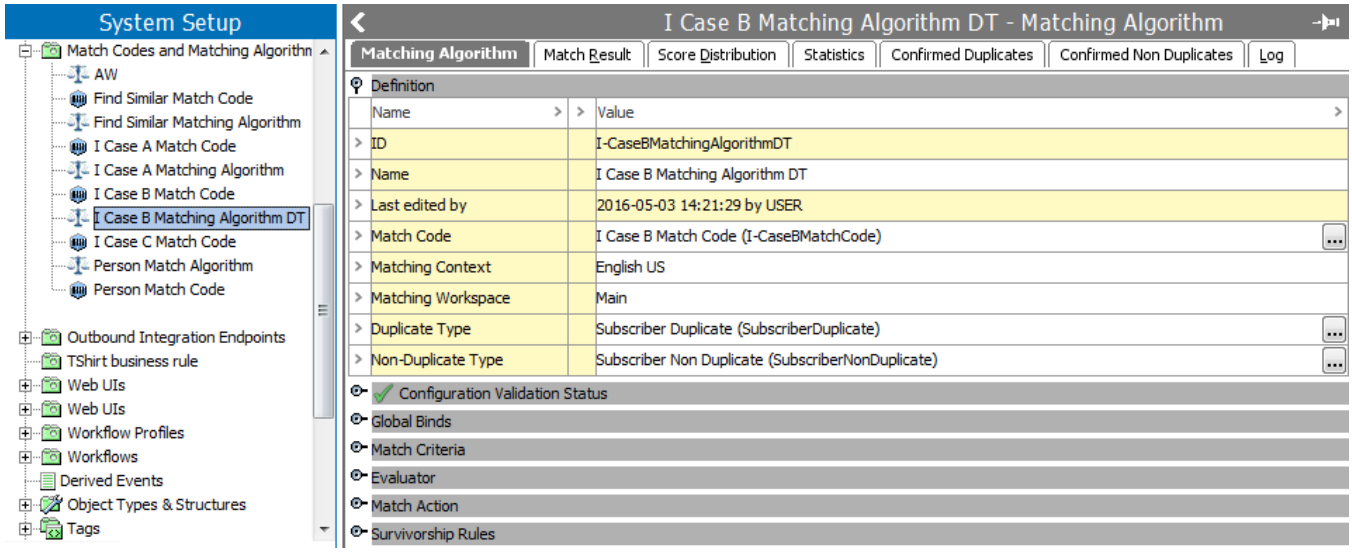
1. In **System Setup**, right-click the node configured to house matching algorithms and select 'New Matching Algorithm'.



2. In the 'Create Matching Algorithm' dialog, define an ID and name for the matching algorithm, and click **Create**.



3. On the Matching Algorithm editor, navigate to the 'Matching Algorithm' tab and click the ellipsis button **(…)** in the 'Match Code' field. Choose the applicable match code, and then click **Select**.

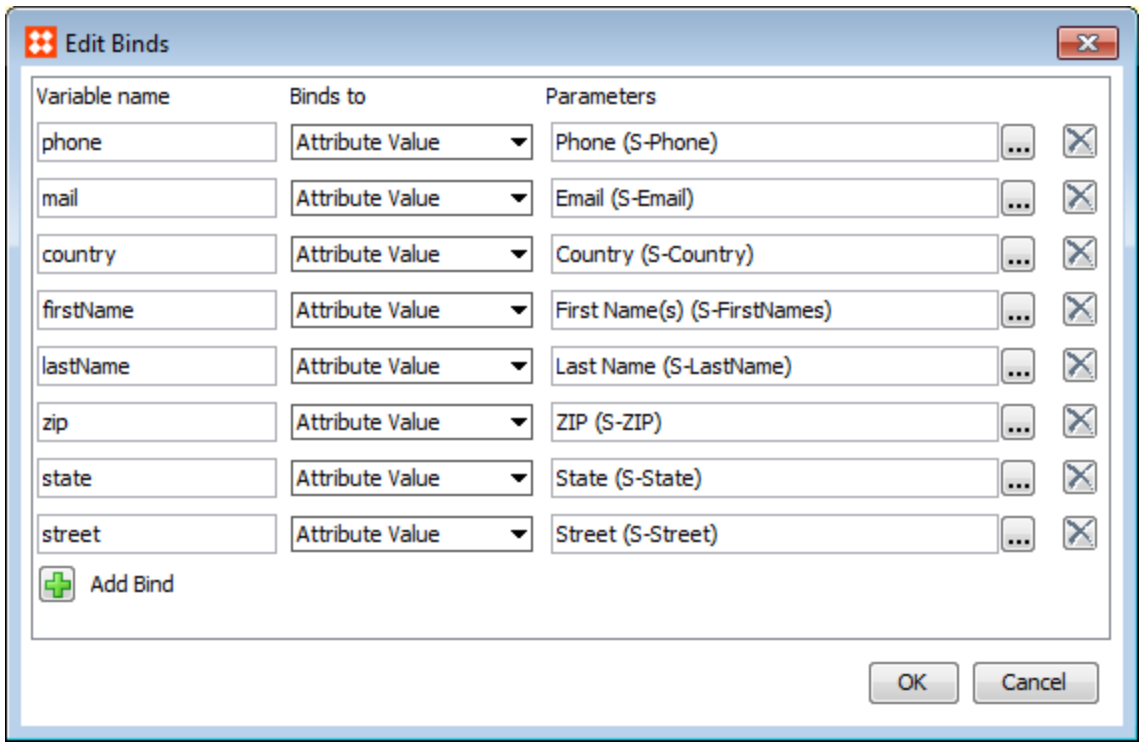


Note: The 'Configuration Validation Status' area displays a green check mark if the matching algorithm has a valid configuration.

4. In the 'Matching Context' and 'Matching Workspace' fields, specify in which context and workspace to run the matching algorithm. This does not have to be the same context and workspace combination as used for the corresponding match code.
5. In the 'Duplicate Type' field, click the ellipsis button **(…)**. In the selector that appears, select the applicable reference type. Next, do the same for the 'Non-Duplicate Type' field. The reference types selected must correspond with those mapped in the component model.

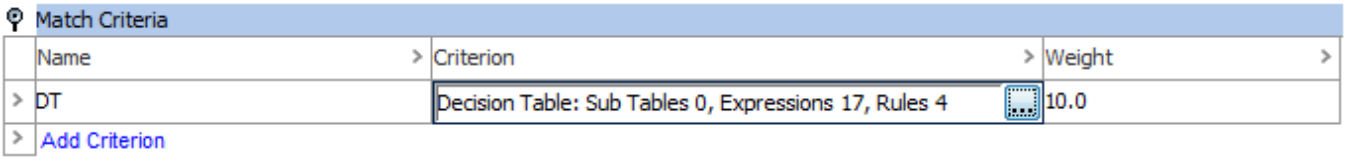
   For more information, see the **Component Model Configuration** documentation.

6. Open the 'Global Binds' flipper and click the **Edit** link. In the 'Edit Binds' dialog, click the Add Binds button to create a new bind, and then use the 'Binds to' dropdown to select a bind (some are displayed within a bind group). Next, if required, under 'Parameters', click the ellipsis button **(…)**to specify an object to bind via the selector dialog that appears. Finally, under 'Variable Name', specify a variable name for the bind. Click **OK** when finished.



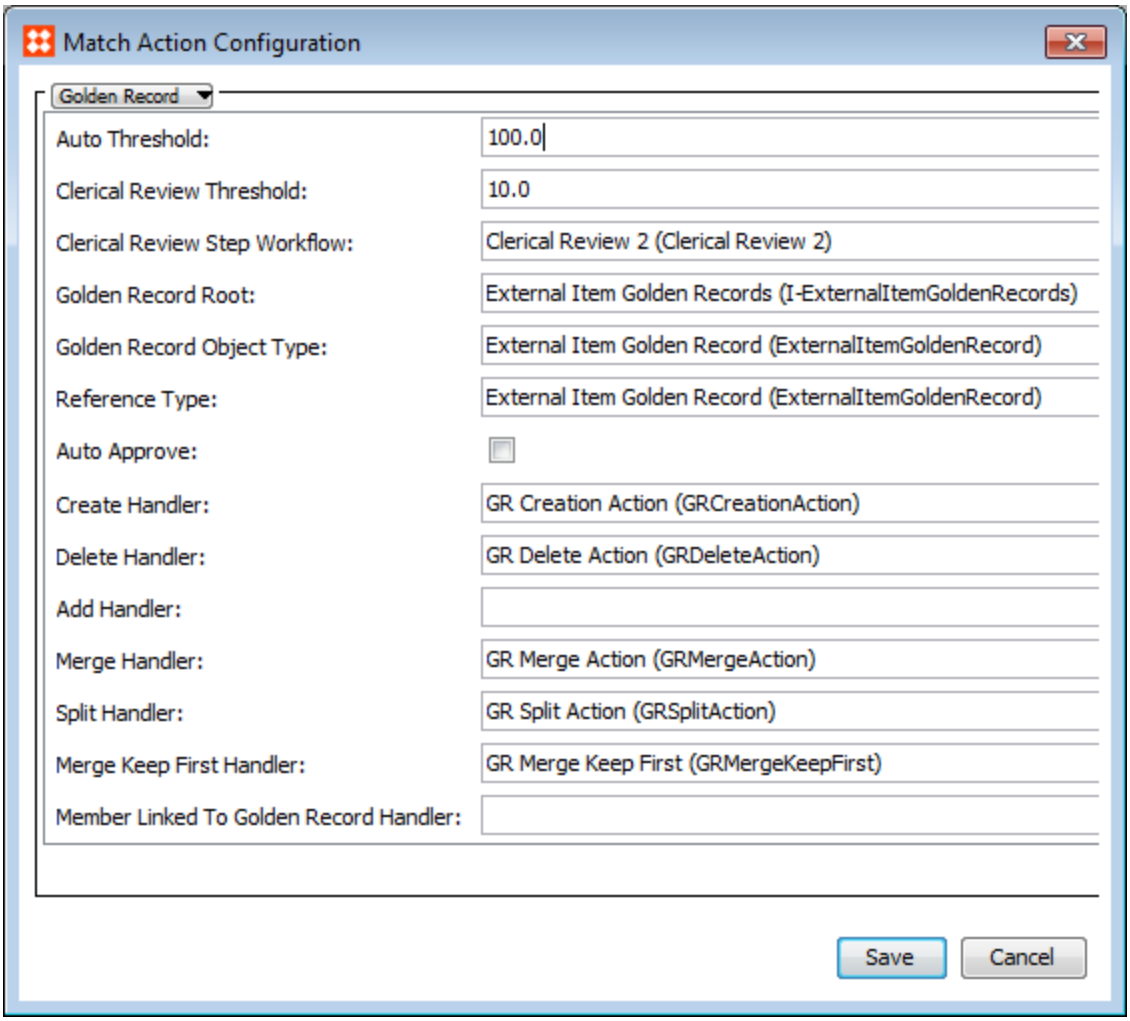
   For more information on using binds with matching algorithms, see the **Match Criteria** documentation.

7. Open the 'Match Criteria' flipper, click the **Add Criterion** link to create a new criterion for the matching algorithm, then specify a name and match criterion in the dialog that appears. In the 'Criterion' field, click the ellipsis button **(…)** to open the JavaScript editor and create the matching criterion. Once complete, specify a weight for the criterion via the 'Weight' field.

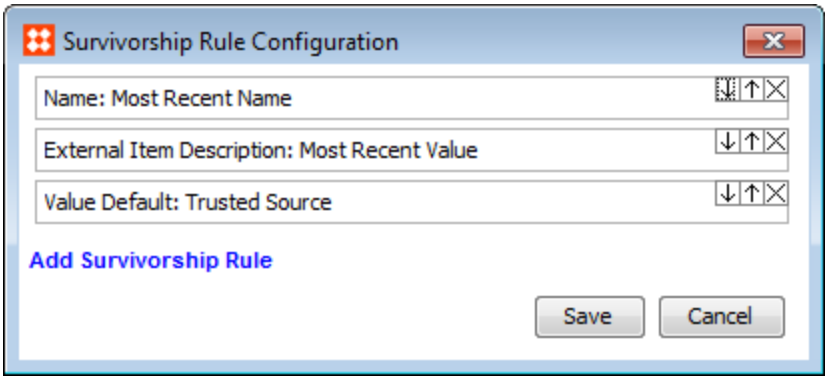


For more information on match criteria, see the **Match Criteria** documentation.

8. Once the match criteria has been configured, open the 'Evaluator' flipper to test the criteria on selected data.
9. Open the 'Match Action' flipper and click the **Edit** link. In the dialog that appears, specify a match action and click **Save** when finished.
   - For **Identify Duplicates**, specify the threshold via the 'Create Threshold' field. For more information, see the **Identify Duplicates Match Action** documentation.
   - For a **Golden Record** setup, several other fields need to be populated. For more information, see the **Configuring Golden Records** documentation.



10. If configuring a golden record match action, open the 'Survivorship Rules' flipper and click the **Edit** link to open the configuration dialog. Click the **Add Survivorship Rule** link and select a rule. Rules can be re-ordered using the arrow buttons and deleted with the 'X' button. Survivorship rules are not used to identify duplicates.

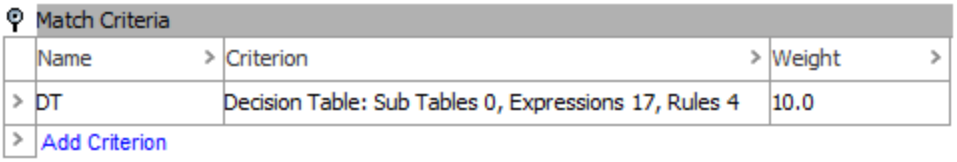


Note: If a survivorship rule is added but has not been configured, the Configuration Validation Status of the matching algorithm will display an 'X'.

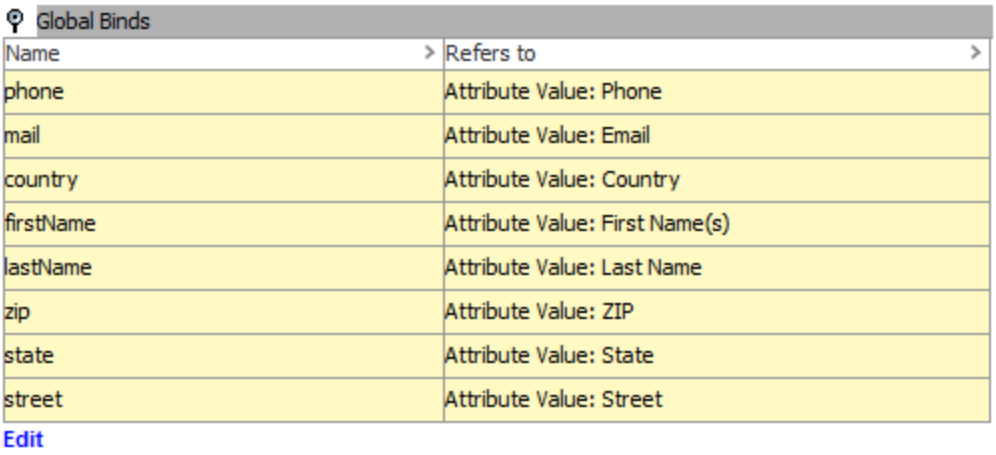
For more information, see the **Survivorship Rules** documentation.

Match Criteria

When configuring a matching algorithm the actual matching logic is defined under the 'Match Criteria' flipper of the matching algorithm.



Binds used in the matching logic can be defined under the 'Global Binds' flipper.



String Comparison Algorithms

While developing your matching and linking strategy, a string comparison algorithm must be chosen to serve as the foundation for the matching process. The available string comparison algorithms include:

- **Levenshtein distance** - A metric for how many edits (substitution, insertion, deletion) it takes to make one string look like another. For example, the Levenshtein distance between the strings 'AXR55487' and '8XRT5487' is 2 because the first and fourth digits are different. In STEP terms, the strings would be 75 percent alike (6/8*100).
- **Damerau-Levenshtein distance** - Similar to Levenshtein distance except that the transposition of two adjacent characters counts only as one edit, not two. For example, the Levenshtein distance between the strings 'AA67' and 'A6A7' is 2 while the Damerau Levenshtein distance is 1.
- **Jaro / Jaro-Winkler distance** - Outputs 0 or 1 where 0 is no similarity and 1 an exact match. Note that these algorithms are available and can be made accessible in STEP via JavaScript, but are not currently included in the STEP core.

Note: The Levenshtein / Damerau-Levenshtein distance has to be converted into a percentage manually.

Matching Algorithm Criteria

As is often the case, it may be that the preferred string comparison algorithm is not sufficient. To compensate for this, it is possible to define criteria that apply the Levenshtein / Damerau-Levenshtein distance directly to strings you build using STEP functions and automatically output an equality metric. Several criteria can be added and given weights that are used when calculating the total equality. The available criterion types are described bellow.

Multi Word Damerau-Levenshtein Distance

The Multi Word Damerau-Levenshtein distance is equal to the Damerau-Levenshtein distance except that transposition of two words does not count as an edit. For example, the distance between 'Paul Johnson' and 'Johnson Paul' is 0. This criterion can come in handy when working with names where first name and surname are in the same attribute value yet the order differs from object to object.

Number Distance

The Number Distance criterion returns the relative distance between two numbers expressed as a percentage: lowest number / highest number * 100. For example, with this simple way of calculating a difference, the numbers 1 and 2 will be as different or equal as 50 and 100.

Special cases:

- If one or both strings are not numerical values, the criterion returns '0'
- If only one of the strings is '0', the criterion returns '0'
- If both are '0', the criterion returns '100'
- If both strings are negative the calculation is the highest number / lowest number * 100
- If one value is positive and the other negative, the criterion returns '0'.

The data to apply the number distance calculation to is generated via STEP functions.

JavaScript

The JavaScript criterion allows you to define your own algorithm for comparing objects. The only requirement is that the result is a number between 0 and 100 (as before, representing the percentage of equality).

From the JavaScript criterion you can draw upon functions defined in business libraries in addition to six objects made available via bindings.

For more information, see the **JavaScript Binds** section of the **Matching and Linking** documentation.

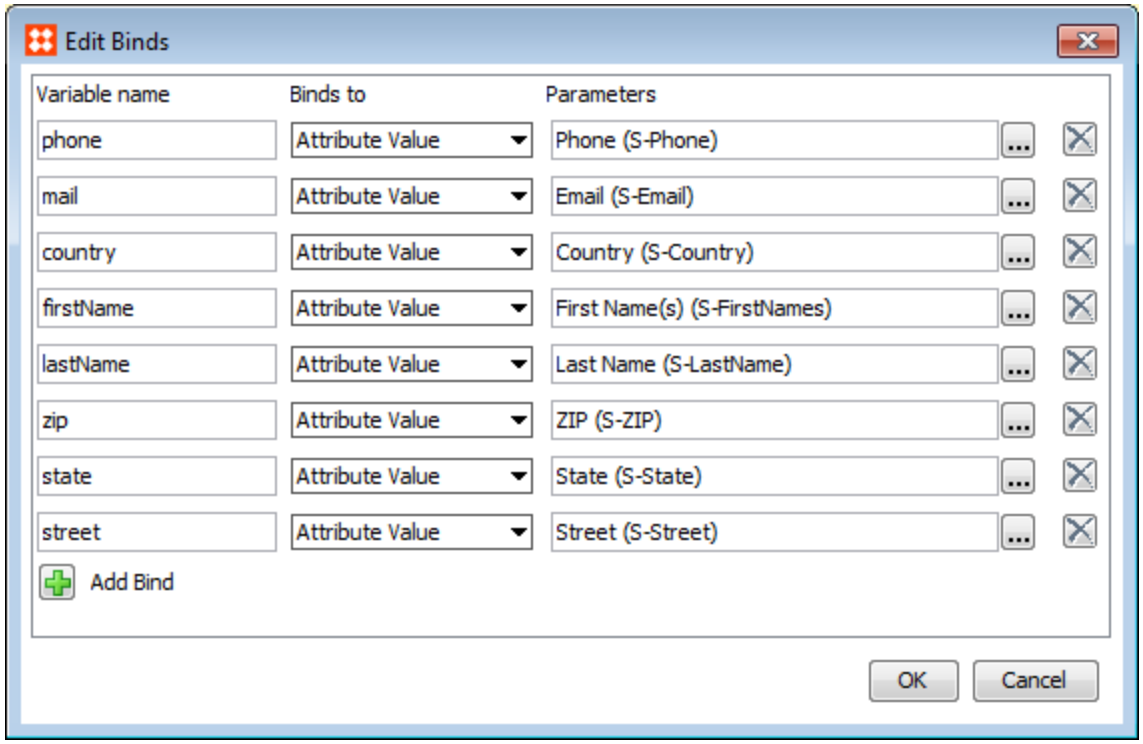
Decision Table

Decision Tables let you break the complexity of matching objects into smaller, more manageable parts. The algorithm is defined using various comparisons between the two objects and a set of rules that govern the conditions for the outcome.

For more information on decision tables, see the **Decision Tables** documentation.

Global Binds

The matching process can strain performance, and when large sets of data are to be processed, there is potentially a significant performance gain if the matching functionality can pre-fetch the values it has to work on. This way, the system will not have to build complete Java objects for the pairs being compared. This is possible via global binds configured on the matching algorithm, where attributes used in the matching algorithm logic can be bound to specific variable names. Values for the attributes will then be pre-fetched before the matching logic is applied, and can be referenced from both JavaScript and STEP functions. For these reasons, it is common setup to use global binds.



When working in JavaScript it is optimal to use only global binds, rather than the 'First Match Object' and 'Second Match Object' bindings which are designed for Find Similar. For more information on Find Similar functionality, see the **Find Similar** section of the **Using a Web UI** documentation.

Decision Tables

The Decision Table match criterion is a flexible method of defining a matching algorithm. Like the other available criteria, a decision table can compare two objects and indicate to what degree they are similar. The algorithm is defined using various comparisons between two objects and a set of rules governing the conditions for the outcome.

In the example table below, separate comparisons are being handled simultaneously via a single decision table.

phoneMatch	emailMatch	nameDistance	Result
TRUE	TRUE		100
TRUE		< 1	95
	TRUE	< 1	95
TRUE		< 3	85
	TRUE	< 3	85

In the above case, the phone numbers and emails of customers, as well as the Damerau-Levenshtein distance of their names are all compared using the same table. The rows in a decision table represent different 'rules' and are evaluated from top to bottom. In the above example, the top rule states that if both phone and email return true (meaning they match in this case) it should return '100' (100% chance of a match). If 'phoneMatch' returns true but 'emailMatch' does not, the result would be '95'. If no matches are found after running through all of the rules in the table it will result in a '0'.

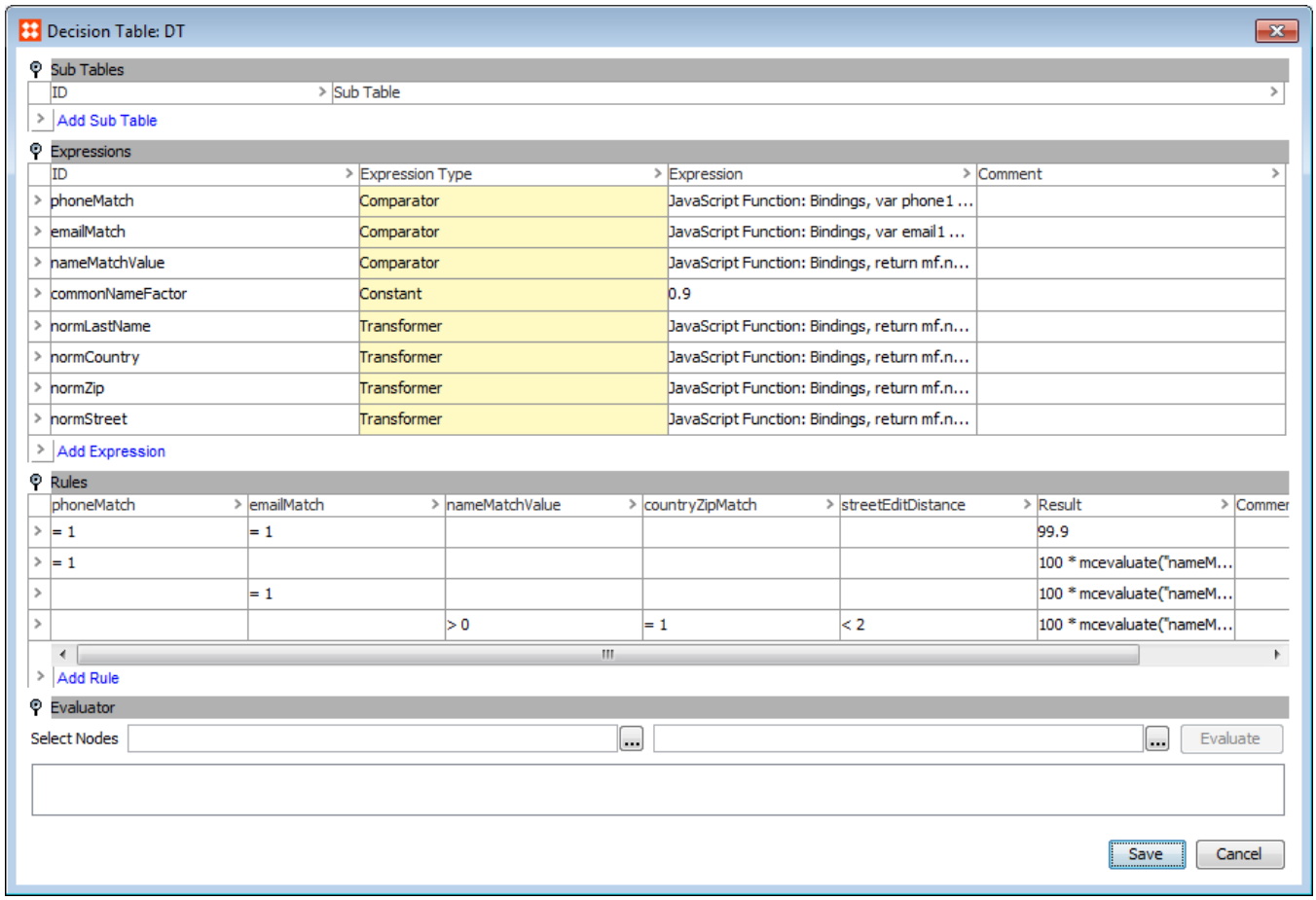
Decision tables in STEP cannot handle Boolean values, so the logic must be expressed using 0's and 1's instead (e.g., if there is a match on phone, 1 is returned, otherwise 0). Thus, the above table would look like this in STEP:

phoneMatch	emailMatch	nameDistance	Result
= 1	= 1		100
= 1		< 1	95
	= 1	< 1	95

phoneMatch	emailMatch	nameDistance	Result
= 1		< 3	85
	= 1	< 3	85

Decision Table Configuration

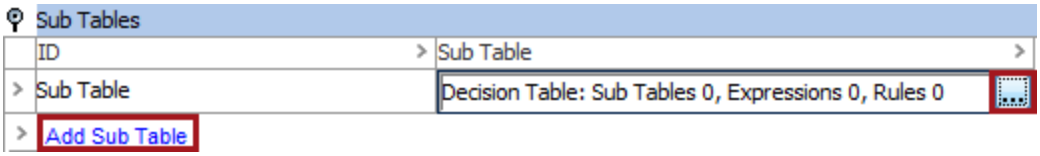
A decision table consists of four major sections: Sub Tables, Expressions, Rules, and Evaluator.



Sub Tables

Sub Tables are optional decision tables used for breaking a complex algorithm down into smaller, more manageable parts. A decision table within a decision table, essentially. A matching algorithm for a customer record can, for example, be split into sub tables for name, address, and phone number, leaving the details of matching these aspects in the sub tables.

To create a sub table, click the **Add Sub Table** link in the 'Sub Tables' area. Next to the newly created sub table, click the ellipsis button **(...)**. A configurable decision table will appear.

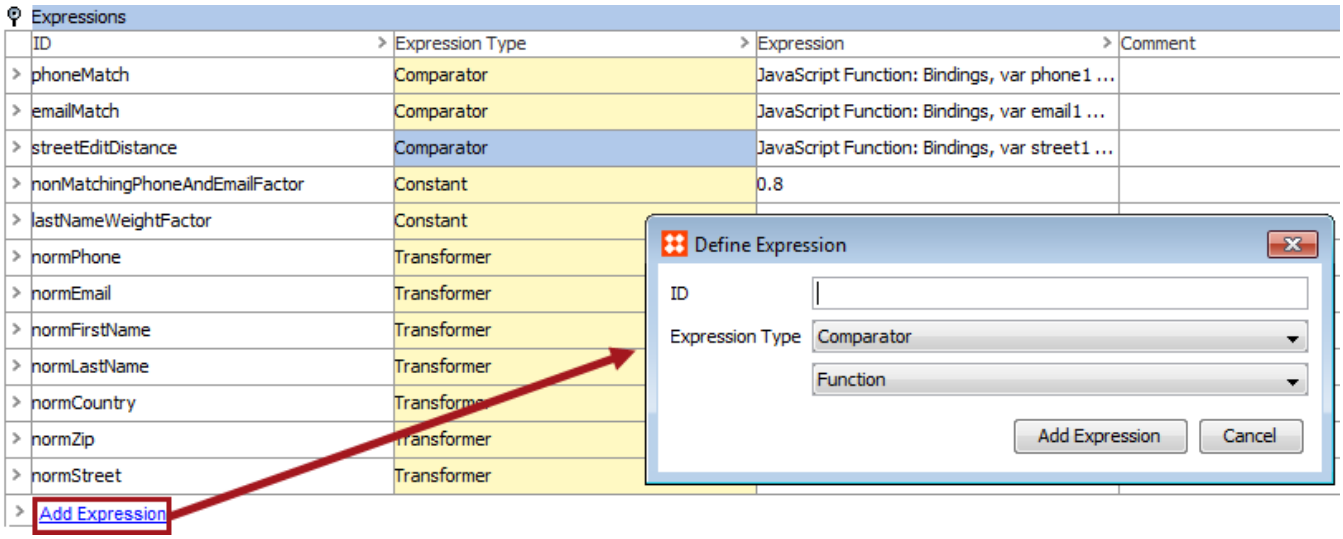


The output from sub tables can be referenced via the `mcevaluate` (STEP function) and `evaluate` (JavaScript) methods. Furthermore, the result value for each rule can be a STEP function.

Expressions

Expressions form the foundation of a decision table and can be used to evaluate one or two specific objects.

To create an expression, click the **Add Expression** link in the 'Expressions' area. In the dialog that appears, specify an ID for the expression, as well as it's expression type (see expression types below), and whether it is to be written as a JavaScript function or a STEP function.



There are three kinds of expressions: 'Transformers' that operate on a single object, 'Comparators' that evaluate multiple objects, and 'Constants' which always return a constant value.

Transformers

Transformers are primarily used for retrieving and normalizing attribute values and only focus on one object at a time. They can be configured using either JavaScript, STEP functions, or an option called 'Attribute Value' that simply lets you reference the value for a specific attribute.

Transformer expressions can work with the results from other transformer expressions. With STEP functions, the result of another transformer or a global bind can be obtained via the `mcevaluate ('ExpressionID')` function. In JavaScript, a 'Match Expression Context' object can be bound using the function `evaluate ('ExpressionID')`, allowing you to reference the result of other transformers or global binds in a similar way.

Comparators

A Comparator Expression works on two objects at a time and is, as the name implies, the piece of functionality doing the actual comparison. Comparators can be formulated using either STEP Functions or JavaScript. In STEP Functions, you can again, reference the output of transformers using the `mcevaluate` function, but will have to specify which of the two objects being compared you want the value for.

For comparators, to get the value from a transformer expression, `mcevaluate` takes two arguments: the first being the ID of a transformer expression, and the second, a string 'first' or 'second', specifying from which of the two objects being compared you want the expression rendered.

Constants

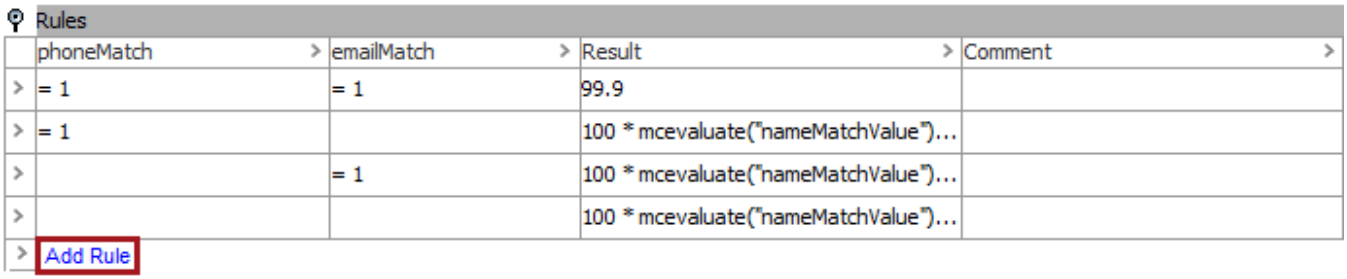
A constant expression is an expression that returns a constant value. Constants can be used to externalize constant values for use in other match expressions. Fine-tuning the matching algorithm can then be done by modifying the constant values, without the need to change or even understand the inner working JavaScript and functional based expressions.

Rules

Rules determine the comparison result for the decision table based on any existing sub tables, as well as the comparators defined in the expressions table. Each sub table and comparator is represented with a column in the rules table. As mentioned earlier, a row in this table represents a rule. A rule consists of two major parts:

- An optional set of conditions on the defined sub tables and comparators
- A definition of the result to be returned if all of the conditions are met

To add rules to the table, click the **Add Rule** link in the 'Rules' area. A new row will appear in the table, allowing for the creation of a new rule by populating the relevant fields in the new row.



Conditions

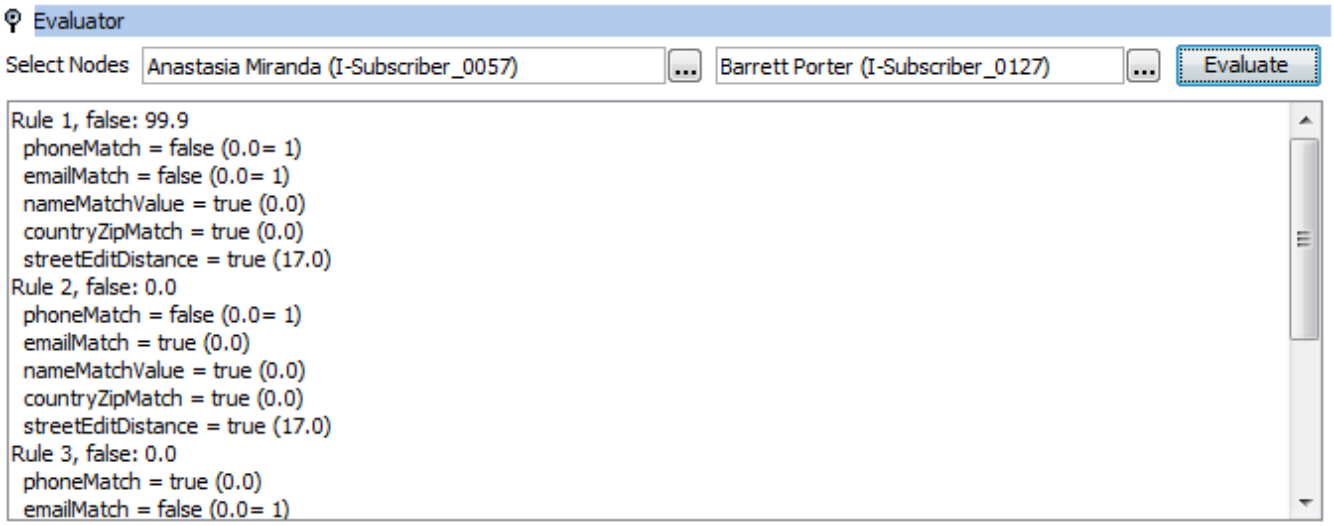
A condition applies to the result of a sub table or a comparator. For example, if a rule column is defined by a comparator that is the textual edit distance between a value in the two compared objects, a condition for that rule could be that the edit distance is less than five. This is specified by writing '<5' in the column of that comparator for this rule. If the edit distance is greater than five, the rule will not have fulfilled the condition. If no condition is specified for a given rule, the sub table or comparator for that column is not taken into account for that rule.

Result

The result column of a rule is either a number or a function that calculates the result. If the result is based on a function, it can be based on the results of sub tables and comparators. To do this, apply the same mechanism as for function-based comparators.

Evaluator

The evaluator is a tool for diagnosing unexpected results that may be encountered. In the evaluator, select two objects that you want to compare. It reports the results and provides detailed information about how the result was obtained. If additional details are required, the evaluators of the sub components can be used.



The evaluator is also available in expressions. If a transformer is evaluated, select one or two objects for evaluation. If you select two objects, a result is displayed for each selected object.

Decision Table Example

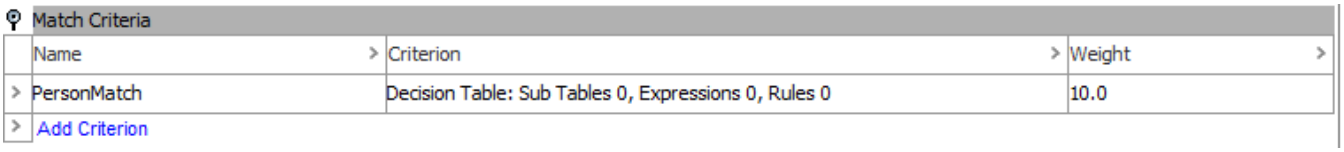
This example uses a decision table match criterion to match person objects. Note that this example has several simplifications and is used to illustrate the principles of a decision table. The example cannot be compared to a production case. Simplifications include both the number of attributes and the complexity of the algorithm. Additionally, only the setup of the match criteria is shown in this example. No other setup is included.

Before you get started with the example, create a Person object with the following attributes:

- FirstName
- LastName
- SSN
- Phone

Create the PersonMatch Match Criterion

To get started, create a matching algorithm in **System Setup**, and then on the **Matching Algorithm** tab, click the **Add Criterion** link and create a match criterion named **PersonMatch**, selecting Decision Table from the dropdown. The value in the **Weight** field is not relevant since only one criterion is used. You can now start adding the sub-tables.

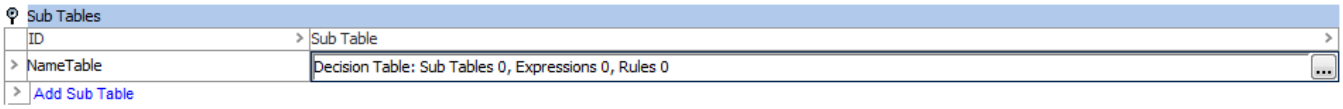


You are now ready to create the sub tables and the relevant rules and expressions.

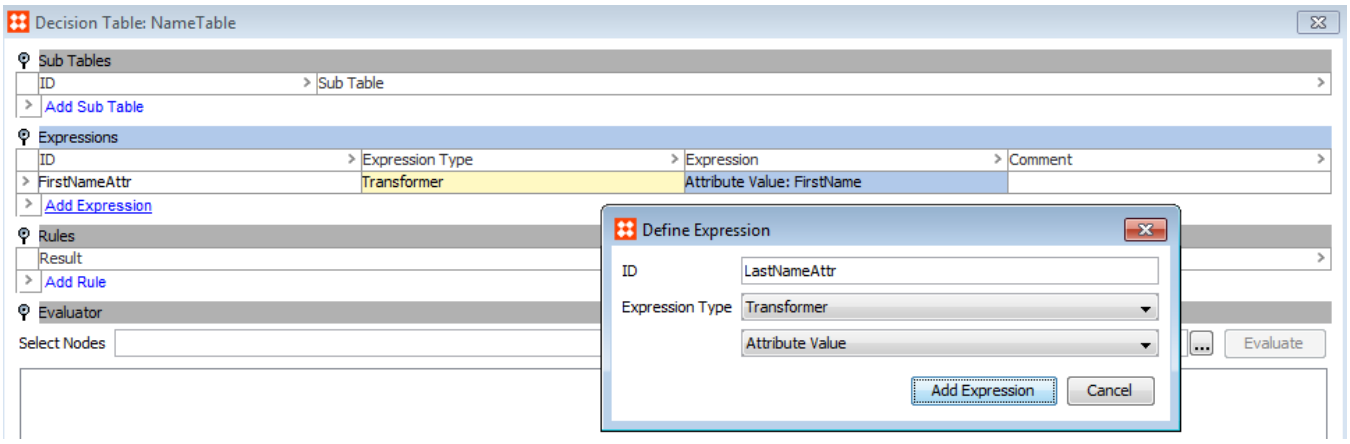
Part 1: Adding the NameTable Sub Table

To break down the complexity of comparing names, the attributes **FirstName** and **LastName** are handled in a sub-decision table with the ID **NameTable**. Likewise, the attributes **SSN** and **Phone** are also handled in sub-tables of the main decision table.

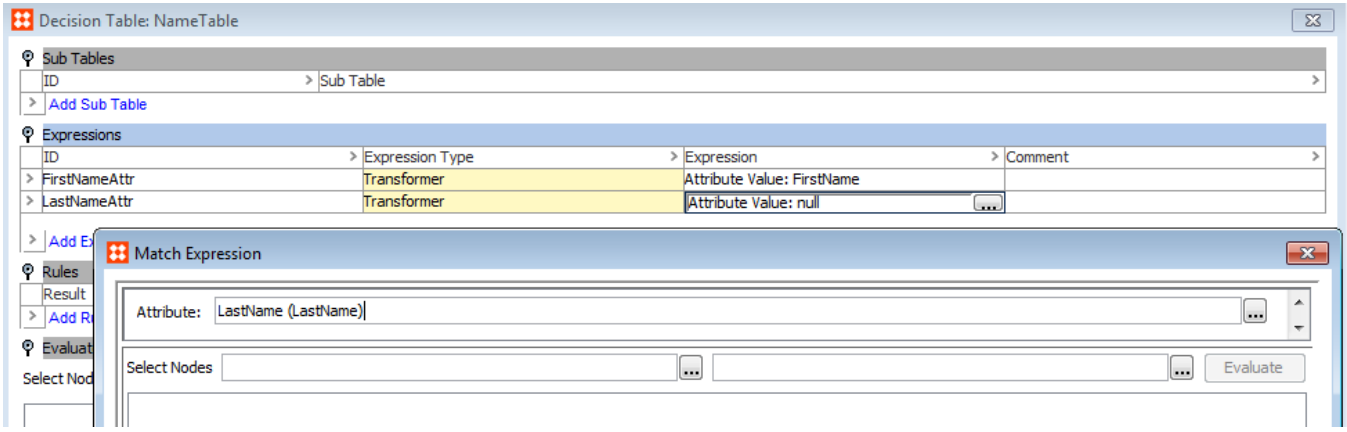
1. In the Match Criteria section **Criterion** column, click the ellipsis button **(…)** next to **PersonMatch** to open the decision table. Click the **Add Sub Table** link, and then add a sub table with the ID **NameTable**.



2. To open **NameTable**, click the ellipsis button **(…)** in the Sub Table column. Click **Add Expression** and then add two transformer expressions, one with the ID **FirstNameAttr** and one with the ID **LastNameAttr**. In both cases select **Transformer** and **Attribute Value** in the **Expression Type** list.

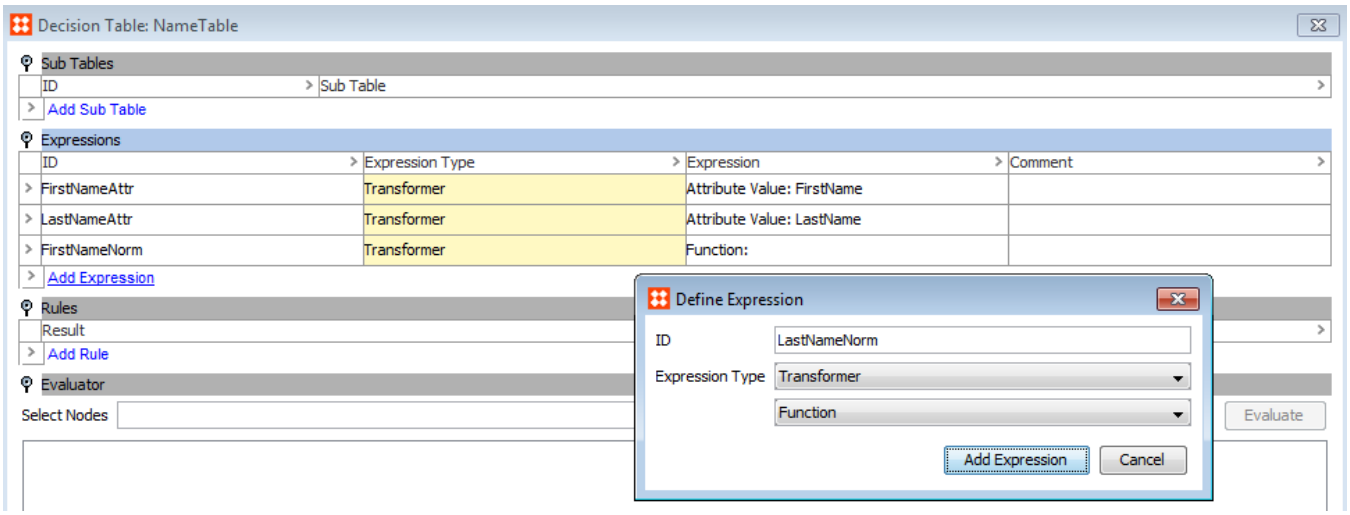


3. In the **Expression column**, click the ellipsis button **(…)** in the **FirstNameAttr** row and in the **LastNameAttr** row, and then browse or search for the relevant attributes.



4. Next, perform a simple normalization of the **FirstName** and **LastName** attributes. Normalizations are often complex, but in this example only leading and trailing spaces will be removed, and the value of the attributes will be converted to lower case if they are in upper case. To do this, apply a transformer on top of each of the attribute transformers.

   Open **NameTable**, click **Add Expression** and then add two transformer expressions, one with the ID **FirstNameNorm** and one with the ID **LastNameNorm**. In both cases select **Transformer** and **Function** in the **Expression Type** list.



- Enter the following formula for 'FirstNameNorm'.

  ```
  lower(trim(mcevaluate('FirstNameAttr')))
  ```

- Enter the following formula for 'LastNameNorm'

  ```
  lower(trim(mcevaluate('LastNameAttr')))
  ```

  This code does the following: The `mcevaluate()` function retrieves the value of the attribute, the `trim()` `function` removes leading and trailing white space, and finally the `lower()` function converts to lowercase.

5. Now, you have to validate that the algorithm works as expected. For this purpose, you create and use the following two test objects:

FirstName	LastName	SSN	Phone
LENNY	BERRY	123-45-6789	(555) 384-7614
Lenny	BERY	123-45-6789	(555) 384-7614

In the sub decision table, in the **Evaluator** area, select the two objects in the **Select Nodes** fields. If you click the **Evaluate** button, it returns `0` since there are currently no rules.

Therefore, In the **Expression** column of the Expressions section, open the expression dialog for **FirstNameAttr** and then click the **Evaluate** button. The result is the FirstName as shown in the table above.

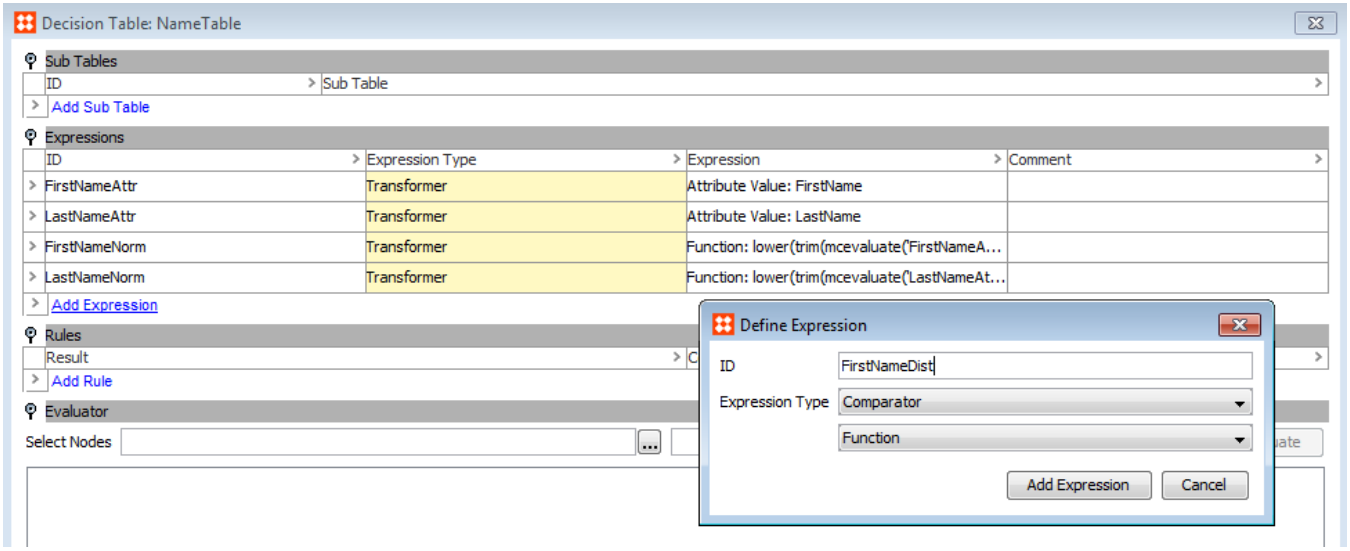
Then open the expression dialog for **FirstNameNorm** and then click the **Evaluate** button. The result is that both values are converted to 'lenny', which is suitable for comparison.

Check the **LastName** attribute in the same way. It should result in the values being converted to 'berry' and 'bery'.

6. The next step is to make comparators on the normalized names. First, compare the normalized first name with the normalized first name of the other object and then compare the normalized last name with the normalized last name of the second object.

   To take typing errors into account, use an edit distance function (levenshtein distance) instead of a simple exact match. This results in a number that indicates how many edit operations are required to change the first name of one object to the first name of the second object. A result of 0 means there is an exact match, 1 indicates a small difference, 2 a bigger difference, and so on.

   Open **NameTable**, click **Add Expression** and then add a comparator expression with the ID **FirstNameDist**. Select **Comparator** and **Function** in the **Expression Type** list.



   On the **NameTable**, in the **Expression** column, click the ellipsis button **(…)**next to **FirstNameDist** to open the function editor. Enter the following formula:

```
matchingLevenshteinDistance(mcevaluate('FirstNameNorm', 'first'),
mcevaluate('FirstNameNorm', 'second'))
```

   Add a second comparator like the one just created, and name it **LastNameDist**. Enter the following expression in its formula editor:

```
matchingLevenshteinDistance(mcevaluate('LastNameNorm', 'first'),
mcevaluate('LastNameNorm', 'second'))
```

   While still on the Match Expression formula dialog, evaluate the new expressions with Lenny Berry. FirstNameDist should return a distance of 0, and LastNameDist should return a distance of 1.

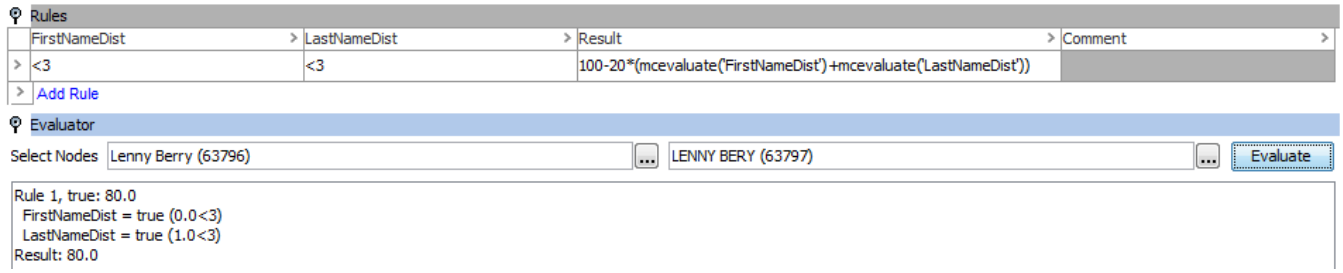
   On the **NameTable** in the **Rules** table note the two new columns that correspond to the two new comparators. To use the comparisons you just created, you need to create at least one rule.

   Click **Add Rule** below the rules table to add a new row to the table.

In this example, an edit distance of up to 2 is acceptable in both the first name and the last name. For each typographical error, 20 points out of 100 are deducted. Enter <3 in the **FirstNameDist** and the **LastNameDist** columns, and then enter the following expression in the **Result** column:

```
100-20*(mcevaluate('FirstNameDist')+mcevaluate('LastNameDist'))
```

Click **Evaluate** to evaluate the two test objects and verify that 80 is returned as expected.



7. Next, check for cross matches to find out if the first name and last name have been swapped. In principle, this is the same procedure as in step 6, except that you need to create two new comparators. The comparators are needed to compare the first name of the first object with the last name of the second object, and vice versa.

   Open **NameTable**, click **Add Expression** and then add a comparator expression with the ID **FirstNameSwitchDist**. Select **Comparator** and **Function** in the **Expression Type** list.

   Enter the following expression in the function editor:

   ```
   matchingLevenshteinDistance(mcevaluate('FirstNameNorm', 'first'),
   mcevaluate('LastNameNorm', 'second'))
   ```

   And another comparator called **LastNameSwitchDist**, and then enter the following expression:

   ```
   matchingLevenshteinDistance(mcevaluate('LastNameNorm', 'first'),
   mcevaluate('FirstNameNorm', 'second'))
   ```

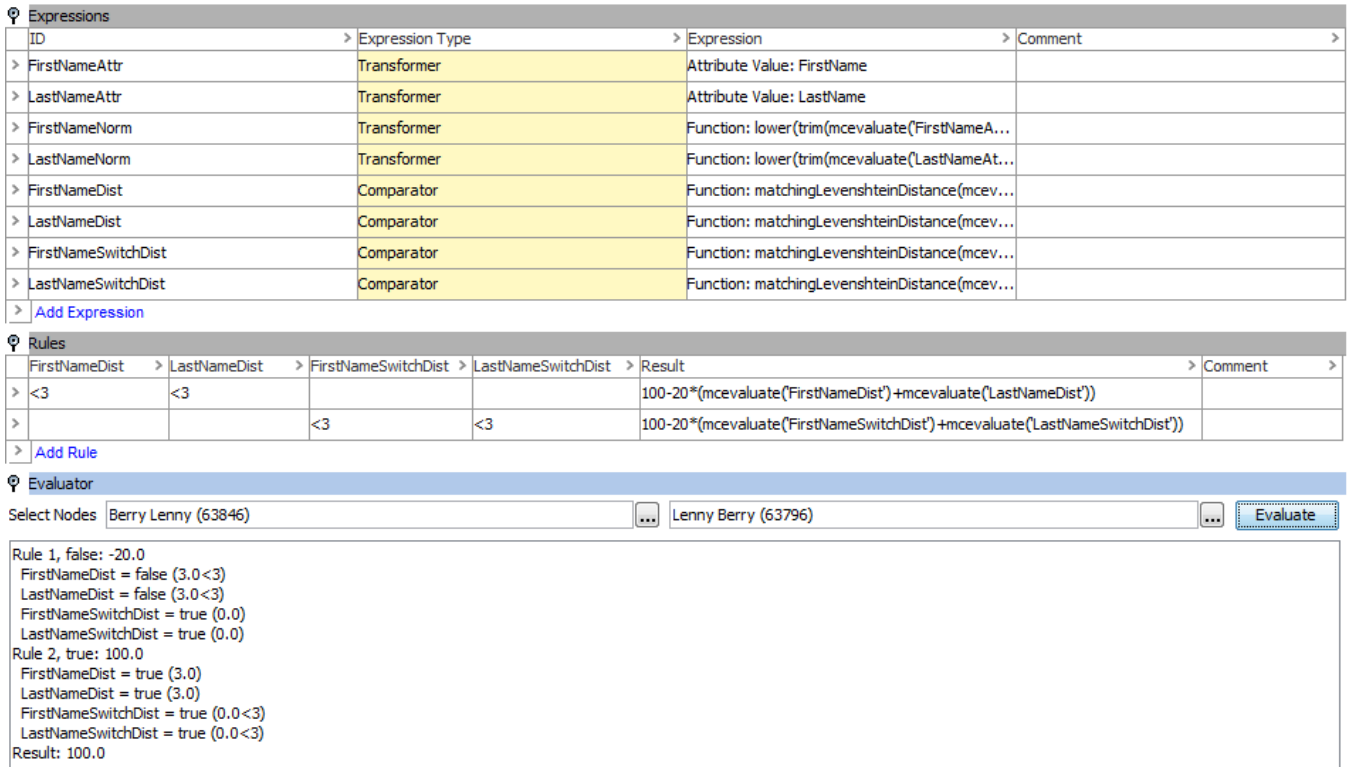
   Click **Add Rule**, and in the **FirstNameSwitchDist** and **LastNameSwitchDist** columns enter `<=3`, and then in the **Result** column, enter the following expression:

   ```
   100-20*(mcevaluate('FirstNameSwitchDist')+mcevaluate
   ('LastNameSwitchDist'))
   ```

   To test this rule, add another test object. Note that FirstName and LastName have been switched and that SSN and Phone contain typographical errors:

FirstName	LastName	SSN	Phone
Berry	Lenny	1234567891	(555) 384-7615

Evaluate and observe that the rule returns the expected result:



Expressions

ID	Expression Type	Expression	Comment
FirstNameAttr	Transformer	Attribute Value: FirstName	
LastNameAttr	Transformer	Attribute Value: LastName	
FirstNameNorm	Transformer	Function: lower(trim(mcevaluate('FirstNameA...	
LastNameNorm	Transformer	Function: lower(trim(mcevaluate('LastNameAt...	
FirstNameDist	Comparator	Function: matchingLevenshteinDistance(mcev...	
LastNameDist	Comparator	Function: matchingLevenshteinDistance(mcev...	
FirstNameSwitchDist	Comparator	Function: matchingLevenshteinDistance(mcev...	
LastNameSwitchDist	Comparator	Function: matchingLevenshteinDistance(mcev...	

Add Expression

Rules

FirstNameDist	LastNameDist	FirstNameSwitchDist	LastNameSwitchDist	Result	Comment
<3	<3			100-20*(mcevaluate('FirstNameDist')+mcevaluate('LastNameDist'))	
		<3	<3	100-20*(mcevaluate('FirstNameSwitchDist')+mcevaluate('LastNameSwitchDist'))	

Add Rule

Evaluator

Select Nodes: Berry Lenny (63846) ... Lenny Berry (63796) ... Evaluate

```
Rule 1, false: -20.0
  FirstNameDist = false (3.0<3)
  LastNameDist = false (3.0<3)
  FirstNameSwitchDist = true (0.0)
  LastNameSwitchDist = true (0.0)
Rule 2, true: 100.0
  FirstNameDist = true (3.0)
  LastNameDist = true (3.0)
  FirstNameSwitchDist = true (0.0<3)
  LastNameSwitchDist = true (0.0<3)
Result: 100.0
```

In the evaluator output, you can see that Rule 2 is chosen and produces the result 100.0. This means that the switched first and last names can now be matched. While this table does not represent a full implementation, it is sufficient to illustrate the principles.

Note that no rules have been created in the parent **PersonMatch** table. This means that the **NameTable** is currently not used when the matching algorithm is applied. In this example, the rules will be added after the sub tables **SSNTable** and **PhoneTable** have been created.

Part 2: Adding the SSNTable Sub Table

1. In the **Criterion** column, click the ellipsis button **(…)** next to **PersonMatch** to open the decision table. Click the **Add Sub Table** link, and then add a sub table with the ID **SSNTable**.
2. Open **SSNTable**, click the **Add Expression** link and then add two expressions:
   - For ID **SSNAttr**, in the **Expression Type** list, select **Transformer** and **Attribute Value**, and then browse or search for the SSN attribute.
   - For ID **SSNDist**, in the **Expression Type** list, select **Comparator** and **Function**, and then enter the following expression:

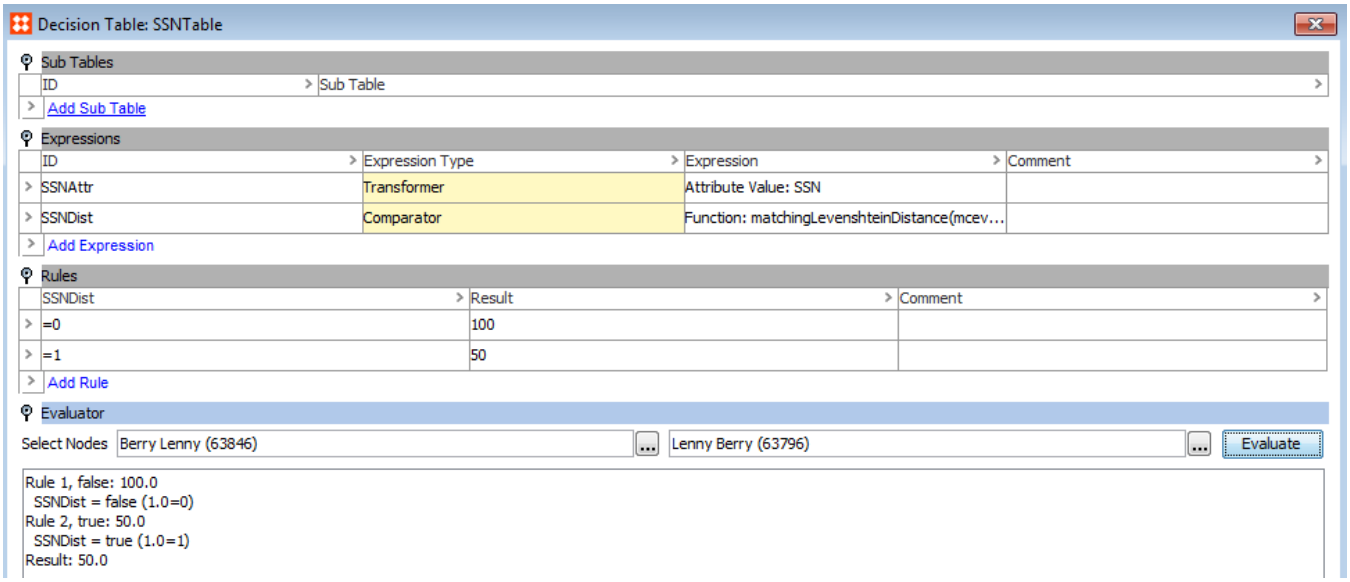
```
matchingLevenshteinDistance(mcevaluate('SSNAttr', 'first'),
mcevaluate('SSNAttr', 'second'))
```

3. In the **Rules** table, note the new column that corresponds to the comparator. To use the comparison you just created, add two rules.

   Click **Add Rule**, and in the **SSNDist** column enter `=0`, and then in the **Result** column, enter `100`.

   Add another rule, and in the **SSNDist** column enter `=1`, and then in the **Result** column enter `50`.

   In this example, only matches with an edit difference of 1 are accepted.



Part 3: Adding the PhoneTable Sub Table

1. In the **Criterion** column, click the ellipsis button **(…)** next to **PersonMatch** to open the decision table. Click **Add Sub Table**, and then add a sub table with the ID **PhoneTable**.
2. Open **PhoneTable**, click the **Add Expression** link and then add three transformer expressions as follow:
   - For ID **PhoneAttr**, in the **Expression Type** list, select **Transformer** and **Attribute Value**, and then browse or search for the Phone attribute.
   - For ID **PhoneNorm**, in the **Expression Type** list, select **Transformer** and **Function**, and then enter the following expression:

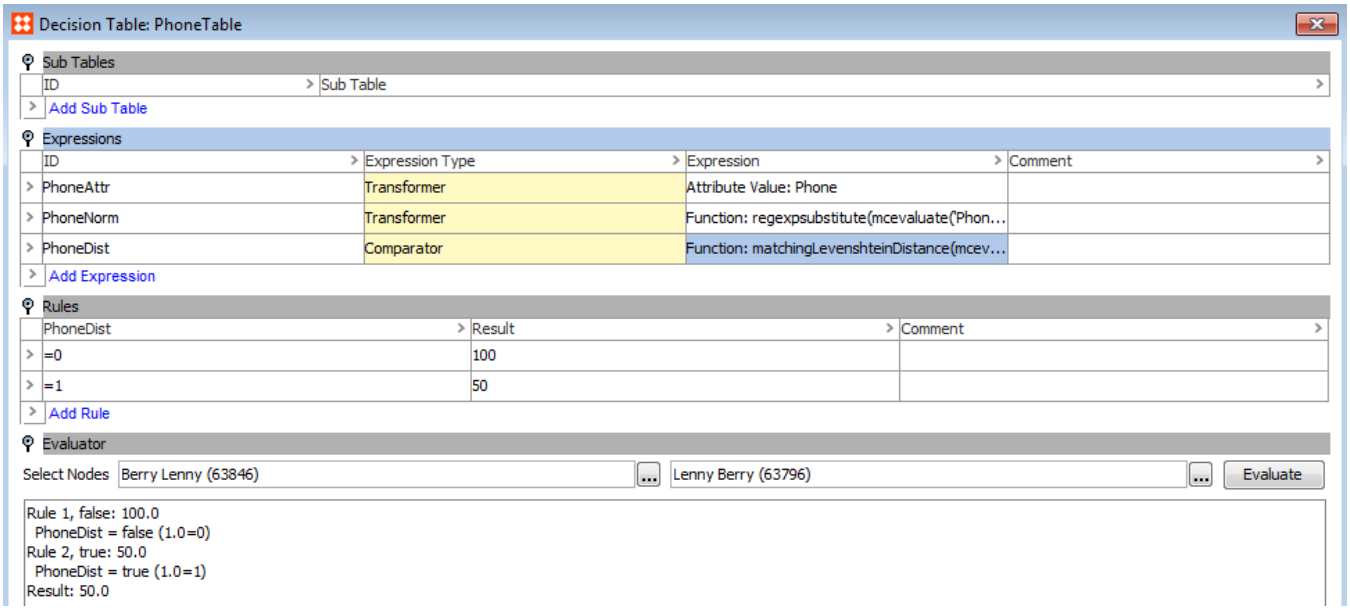
```
regexpsubstitute(mcevaluate('PhoneAttr'), '[- \(\)]', '')
```

   This expression removes brackets, dashes and spaces found in the source values.

   - For ID **PhoneDist**, in the **Expression Type** list, select **Comparator** and **Function**, and then enter the following expression:

```
matchingLevenshteinDistance(mcevaluate('PhoneNorm', 'first'),
mcevaluate('PhoneNorm', 'second'))
```

3. In the **Rules** table note the new column that corresponds to the comparator. To use the comparison you just created, add two rules.
   - Click **Add Rule**, and in the **PhoneDist** column enter `=0`, and then in the **Result** column, enter `100`.
   - Add another rule, and in the **PhoneDist** column enter `=1`, and then in the **Result** column enter `50`.
4. Select the two Nodes with different phone numbers for comparison.
5. Click **Evaluate** to verify that the rules work as expected.



Part 4: Adding the PersonMatch Rules

To use the sub tables in a matching algorithm, the sub tables must be referenced in rules from the top-level table MatchPerson. This example illustrates one way to do this.

1. In the **Criterion** column, click the ellipsis button **(...)** next to **PersonMatch** to open the decision table.
2. Click **Add Rule**, and in the **SSNTable** column enter `=100`, and then in the **Result** column, enter `100`.

   If there is a perfect match, this rule bypasses the matching on name and phone number. The rule assumes that data is valid for the SSN attribute (almost like a database primary key) so the likelihood of a false positive is very low.

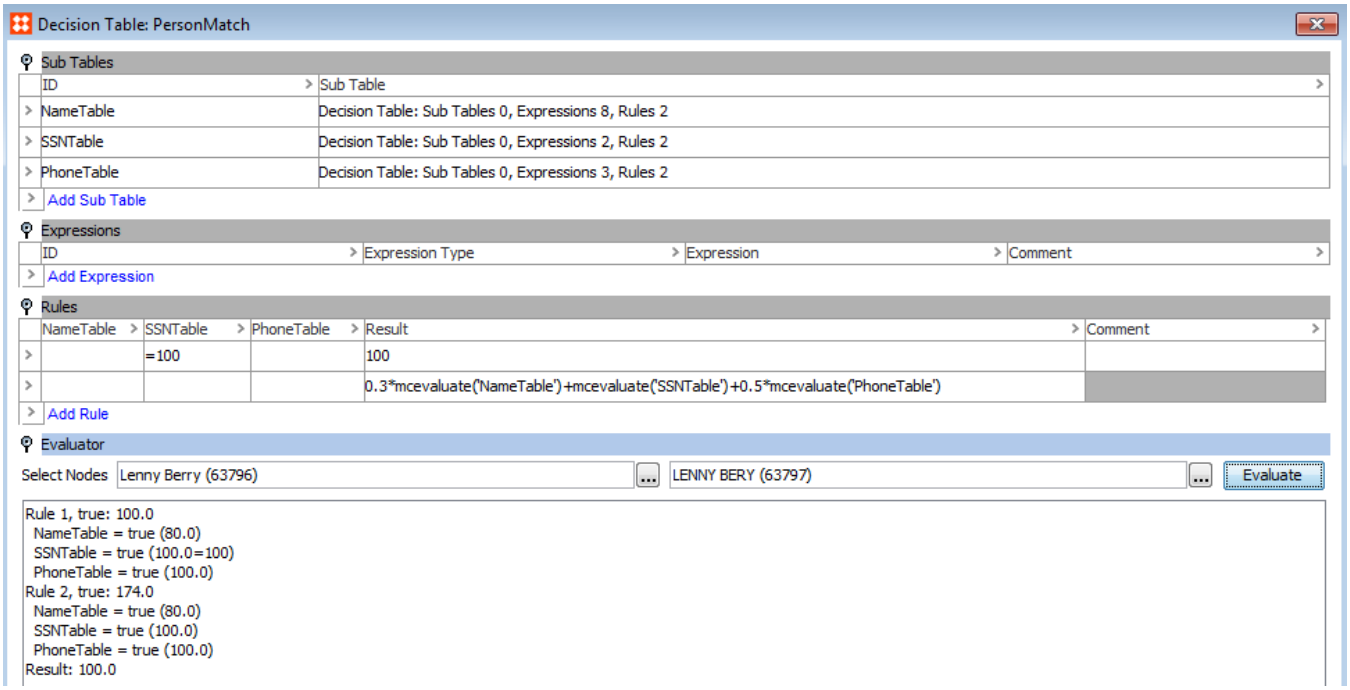
   Add another rule and then in the Result column enter the following expression:

   ```
   0.3*mcevaluate('NameTable') + mcevaluate('SSNTable') +
   0.5*mcevaluate('PhoneTable')
   ```

   Because there are no conditions, the rule is evaluated unconditionally if the first rule does not match. The result is a weighted sum of the results of the three sub tables. The weights have been chosen to reflect highest confidence in the SSNTable, lower confidence in the PhoneTable, and lowest confidence in the NameTable.

You can adjust the weights to place higher or lower emphasis on either Name matching, SSN matching, or Phone matching.

3. Select the first two objects you added, where the first name are spelled the same, but the last names differ.



Matching and Linking JavaScript Binds

First Match Object and Second Match Object

When used in a Find Similar matching algorithm, the First Match Object and Second Match Object binds are used to search for similar objects across a reference. For example, when searching for a similar address on a contact, when addresses are linked to contacts by a reference.

Outside of Find Similar, for performance considerations avoid using this binding. Instead, use the 'Match Expression Context' to obtain values via global binds.

For more information on global binds, see the **Match Criteria** documentation.

For more information on Find Similar functionality, see the **Find Similar** section of the **Using a Web UI** documentation.

Matching Functions

When bound to a variable, the variable will hold an object that has the following member functions:

Matching Functions	Header Row
Levenshtein and Damerau-Levenshtein Distance Functions	• damerauLevenshteinDistance(string1, string2) • damerauLevenshteinDistanceLimited(string1, string2, limit) • levenshteinDistance(string1, string2) • levenshteinDistanceLimited(string1, string2 , limit)
Soundex	• soundex(string)
Metaphone3	• metaphone3(string) • metaphone3alternate(string)

The Levenshtein and Damerau-Levenshtein Distance Functions

The Levenshtein and Damerau-Levenshtein distance functions work as described in the **Match Criteria** topic. Additionally, each function has a variant which allows you to specify a maximum number of edits that can be returned (limit defined as an integer).

For more information on the Levenshtein and Damerau-Levenshtein distance functions, see the **Match Criteria** section of the **Matching and Linking** documentation.

Soundex

Soundex is a phonetic algorithm for indexing names by sound, as pronounced in English. In practice the function allows you to generate a four character code from a string, where the code is the phonetic representation.

For more information on Soundex, see the **Text Functions** section of the **System Setup / Super User Guide** documentation.

Metaphone3

Metaphone 3 is an improved version of Soundex.

For more information on Metaphone3, see the **Text Functions** section of the **System Setup / Super User Guide** documentation.

Match Expression Context

Can be used to reference global bind values from a pure JavaScript criterion.

For more information on global binds, see the **Match Criteria** section of the **Matching and Linking** documentation.

Lookup Table Home

When bound to a variable, the variable will hold an object that allows you to work with transformation lookup tables from the script.

The object has a single member function: `getLookupTableValue(asset_id, string)`

For more information, see the **Using JavaScript with Lookup Tables** section of the **Business Rules** documentation.

STEP Manager

When bound to a variable, the variable will hold a STEP Manager, i.e., the gateway to the public Java API.

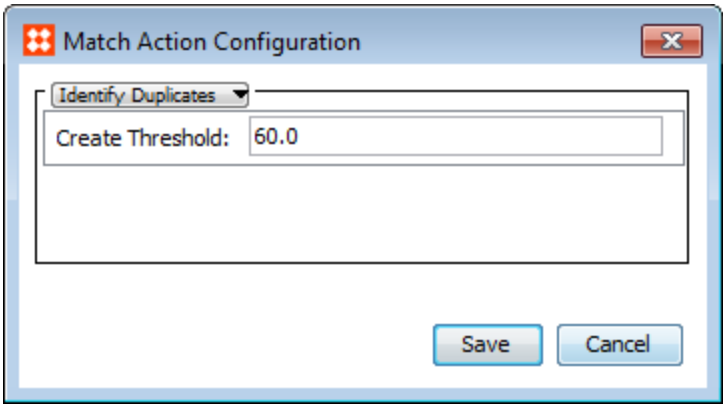
If desired, you could implement the entire algorithm for comparing objects using just the JavaScript criterion and have it as complex as required.

For more information, see the STEP Manager Bind section of the Business Rules documentation.

Identify Duplicates Match Action

The Identify Duplicates Match Action allows for the identification of duplicates without automatically applying any actions to those records.

The only parameter required when configuring this match action is a threshold specifying how equal objects have to be in order to be identified as possible duplicates.



For more information on configuring a matching algorithm with this match action, see the **Configuring Matching Algorithms Overview** documentation.

When a matching algorithm configured with this match action is applied, the possible matches can be inspected from the matching algorithm object.

If two objects are manually confirmed as being duplicates, a reference will be created between them. The existence of such a reference means that regardless of how the objects are modified, the matching algorithm will always see them as duplicates. The reference type configured as the 'Duplicate Type' on the matching algorithm is used in this case.

Similarly, possible duplicates can be rejected, creating a reference between them. With such a reference, the two objects will never be identified as duplicates by the matching algorithm regardless of how they are modified. The reference type configured as the 'Non-Duplicate Type' on the matching algorithm is used in this case.

In both of the above cases the references can be manually removed via the 'References' tab of the object in question.

If two objects are confirmed duplicates, it is possible to manually merge them into a single object. When performing a merge, you can decide which object will survive the merge and what data that object will inherit from the object being deleted.

For more information, see the **Handling Potential Duplicates** documentation.

Golden Records Survivorship Rules

A golden record's attribute and reference data is copied from its source objects automatically. What data is copied from which object is determined by 'Survivorship Rules' set on the applicable matching algorithm. These rules can be defined independently for an object's name, its references, and its attributes / attribute groups. A default comprehensive rule exists for attributes, so it is not necessary to define a rule for each and every attribute / attribute group.

Note: Survivorship rules are only applicable to golden record configurations.

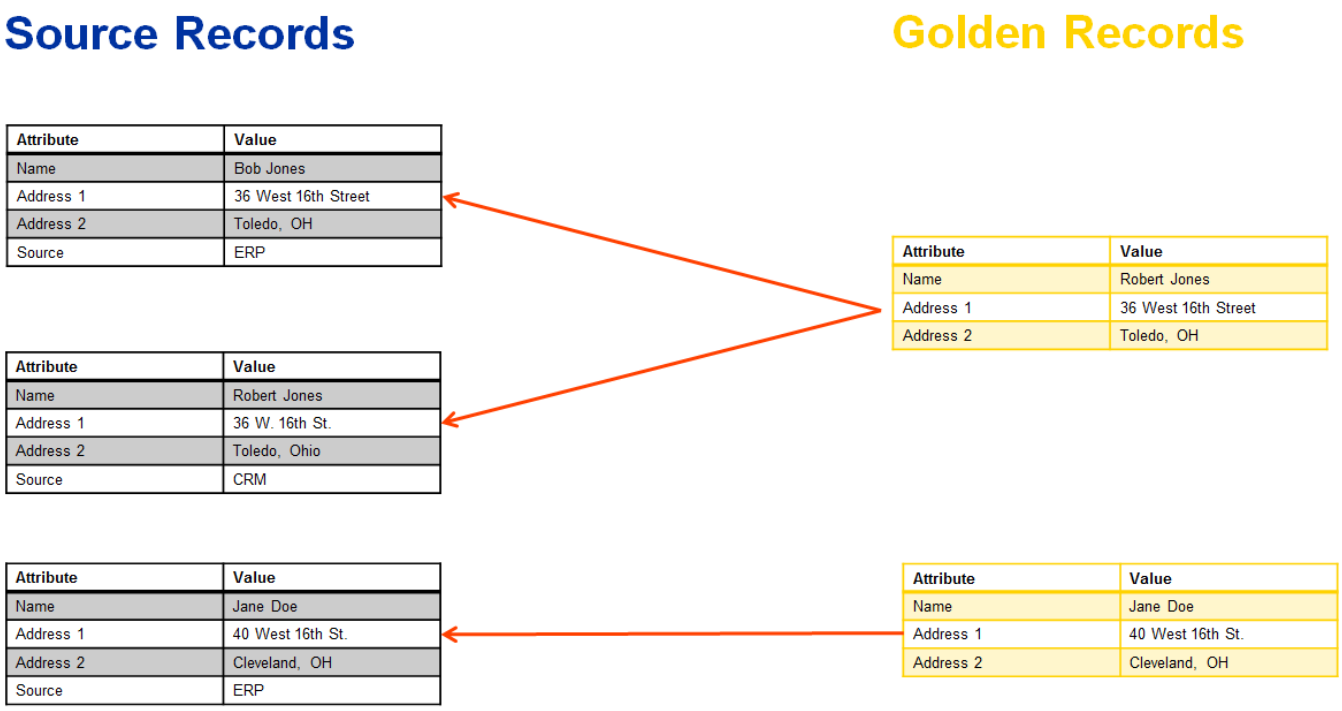
Two concepts are in play when determining which of multiple source objects to take data from:

- Trusted Source
- Most Recent

Trusted Source

Information about the source (such as what system / supplier the object originated from) must be available on the source objects when using the 'Trusted Source' option. Any attribute selected when configuring the matching component model must be populated on the source objects. Typically this should be a required LOV based attribute that does not allow users to add values.

With this information available, a survivorship rule can be defined that lists the possible sources in the preferred order. Take the example illustrated below where the source objects come from either an ERP or a CRM system:



In the image above, a single golden record is produced with data from the 'Bob Jones' / 'Robert Jones' duplicate pair. The 'Name' is taken from the object that comes from the CRM system while 'Address 1' and 'Address 2' are taken from the object that comes from the ERP system.

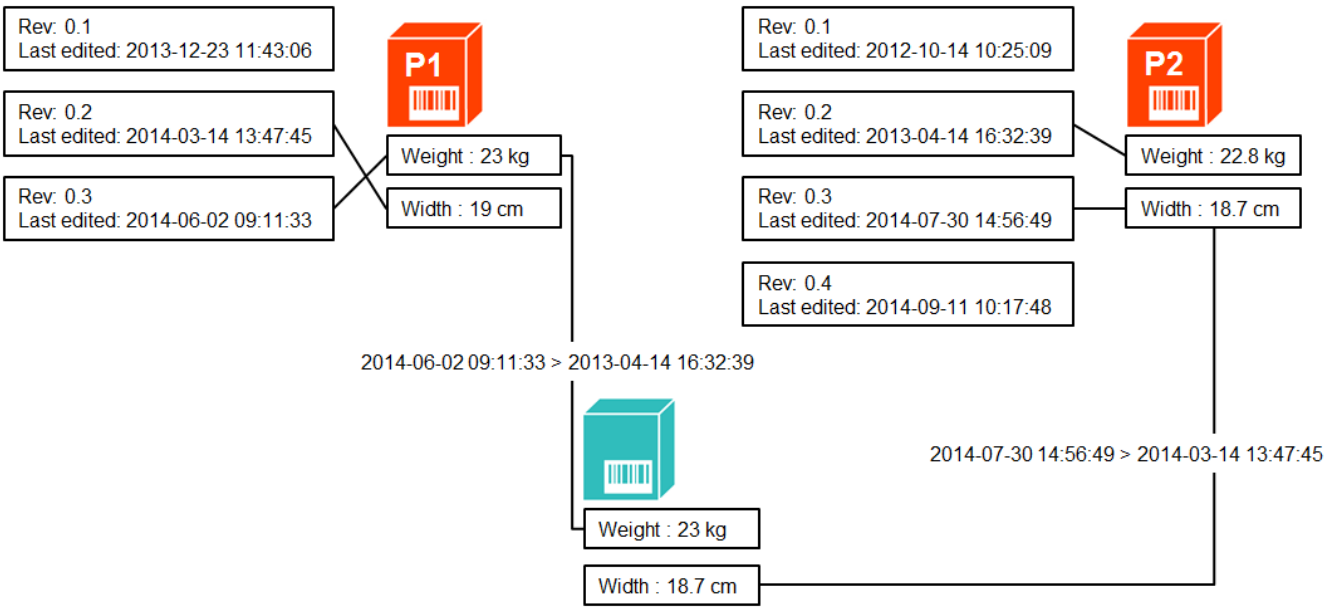
If only trusted source survivorship rules are used, this implies that a rule has been defined for 'Name' that favors the CRM source over the ERP source. Additionally, there is a rule(s) which state that 'Address 1' and 'Address 2' favor the ERP source over the CRM source (this could be either two separate rules or a single default comprehensive rule).

Golden records that only reference a single source object will get data from that object, but only if it is listed as a trusted source. Only data that has survivorship rules defined are copied to the golden record.

Most Recent

As the name implies, the 'Most Recent' survivorship rule strategy simply takes the most recent data from a golden record's source objects. While this sounds simple, it is important to note that this does not necessarily mean the source object with the newest revision. Instead, each relevant value is analyzed individually. Those values with the newest revisions (based on the 'Last Edited' date) are taken from their source objects.

In the image below, two products (P1 and P2) are combined to create one golden record.



Survivorship Rule Types

It is possible to use a combination of the 'Most Recent' and 'Trusted Source' strategies described above so that, for example, the value of one attribute could be copied to the golden record based on a trusted source setup while the other attributes are copied based on which ones were updated most recently.

The available survivorship rule types are listed below.

Name: Most Recent

Specifies that 'Name' should be taken from the source object with the most recent name. No configuration needed.

Name: Trusted Source

Allows you to enter a comma separated list of trusted sources starting with the most trusted source, then the next-most, and so on. 'Name' is taken from the most trusted source listed, unless a source object from this source does not exist. If one does not exist for the first source on the list, 'Name' is taken from the next-most trusted source, and so on. If 'Name' does not exist for any of the trusted sources, the golden record's name will be left blank.

Value: Most Recent

Same logic as with 'Name: Most Recent'. Allows you to select a single attribute or all attributes in a specific group for which the rule applies.

Value: Trusted Source

Same logic as with 'Name: Trusted Source'. Allows you to select a single attribute or all attributes in a specific group for which the rule applies.

Reference: Most Recent

For references and links, the 'Most Recent' logic is the same as with names and values. However, the UI allows you to select not just one reference / link type, but three:

- **Reference Type** - The only mandatory field. Specifies which of the reference / link types valid from the source objects you are handling. If this is the only field populated, the golden record will get a reference / link of the same type pointing to the same target.
- **Golden Record Reference Type** – If the objects the source objects are pointing to also have golden records, you can configure the new golden record to point to this golden record rather than the source object's original target. The reference type that links the target golden records and target source objects must be entered.
- **Mapping Reference Type** – If this field is not populated, the reference or link created for the golden record will be of the same type as the source object's reference / link. By specifying a reference / link type, you can map the reference / link to this type.

Reference: Trusted Source

Same options available as with the 'Reference: Most Recent' option, but in this case, the trusted source strategy is used to determine which source object to copy a given reference / link from.

Data Container: Most Recent

This rule allows the most recent data container instances and their attribute values to be promoted to the golden records. The following parameters must be configured:

- **Business Condition:** Click the ellipsis button **(…)** and select a business condition that is valid for the golden record object type. This condition must be a JavaScript rule that uses the 'Pairs of Attributes' bind to compare data container instances on source records and golden records.
- **Data Container Type:** Click the ellipsis button **(…)** and select the relevant data container type.

Data Container: Trusted Source

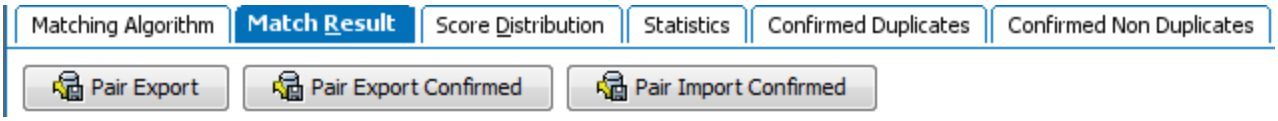
This rule allows data container instances and their attribute values that originate from the specified trusted source (s) to be promoted to the golden records. The following parameters must be configured:

- **Business Condition:** Click the ellipsis button **(…)** and select a business condition that is valid for the golden record object type. This condition must be a JavaScript rule that uses the 'Pairs of Attributes' bind to compare data container instances on source records and golden records.

- **Comma separated list of trusted sources:** Enter a comma separated list of all trusted sources.
- **Data Container Type:** Click the ellipsis button **(…)** and select the relevant data container type.

Adjusting a Matching Algorithm

When a matching algorithm has first been applied, the identified matches are displayed on the 'Match Result' tab of the matching algorithm. From here, three options related to fine-tuning the matching algorithm are available: 'Pair Export', 'Pair Import Confirmed', and 'Pair Export Confirmed'.



In order to determine how well different versions of a matching algorithm work, you will need a set of confirmed duplicates and confirmed non-duplicates that can function as a truth table that the algorithms can be tested against. That is, pairs where human users have inspected the data, and for each pair determined, whether they are duplicates or not. This truth table can be built directly from the 'Match Result' tab by confirming or rejecting matched pairs.

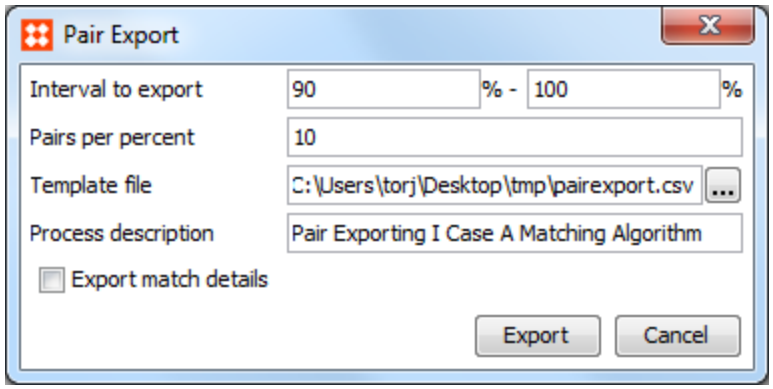
Alternately, the 'Pair Export' and 'Pair Import Confirmed' options can be used.

Pair Export

With the 'Pair Export' option, a CSV file is produced that can be used for manual, offline confirmation / rejection of matched pairs. The file has a header and the following standard columns:

- **<Pair>** - One row per source object and the 'Pair' info is used to indicate which objects belong together. The first two rows will have the value '1', the next two rows will have '2', and so on.
- **<Match y n>** - Column used to indicate whether pairs are matches or not. A value is only required for the first object in a pair.
- **<Equality>** - The calculated equality between the two objects.
- **<ID>** - ID of the object in the current row.
- **<Name>** - Name of the object in the current row.
- **<URL>** - STEP URL of the object in the current row.

Additionally, for people to work with the data offline, attribute values should be included in the file. For this purpose, a template file with a semicolon-separated list of attribute IDs must be prepared in advance and selected in the 'Pair Export' dialog as shown below.



In the Pair Export dialog, specify the following:

- **Interval to export:** Specify an interval that includes pairs expected to be both matches and non-matches, as well as pairs that are not clear matches or non-matches. Only pairs with scores within this interval are exported.
- **Pairs per percent:** Specify the maximum number of pairs to be exported for each percentage point.
- **Template file:** Select the file (created beforehand) that contains the required attribute values. The format of the file is the attribute IDs separated by semicolons (;).
- **Process description:** Provide a description for the background process.
- **Export Match Details:** Add additional columns with part scores from decision table comparators and sub decision tables.

The exported file can be opened in Excel and the decisions can be entered in the <Match y n> column, as shown below:

	A	B	C	D	E	F	G	H
1	<Pair>	<Match y n>	<Equality>	<ID>	<Name>	<URL>	OEM	OEMPartNumber
2	1	y	100	I-EI00042	I EI00042	step://product?id=I-EI00042	Weller	d421881
3	1		100	I-EI00055	I EI00055	step://product?id=I-EI00055	WELLER INC.	d4-21881
4	2	n	100	I-EI00042	I EI00042	step://product?id=I-EI00042	Weller	d421881
5	2		100	I-EI00142	I EI00142	step://product?id=I-EI00142	Weller	D421881
6	3		100	I-EI00055	I EI00055	step://product?id=I-EI00055	WELLER INC.	d4-21881
7	3		100	I-EI00142	I EI00142	step://product?id=I-EI00142	Weller	D421881
8	4		100	I-EI00058	I EI00058	step://product?id=I-EI00058	OSP Manufacturing	yzo41241
9	4		100	I-EI00134	I EI00134	step://product?id=I-EI00134	OSP Manufacturing	YZO-41241

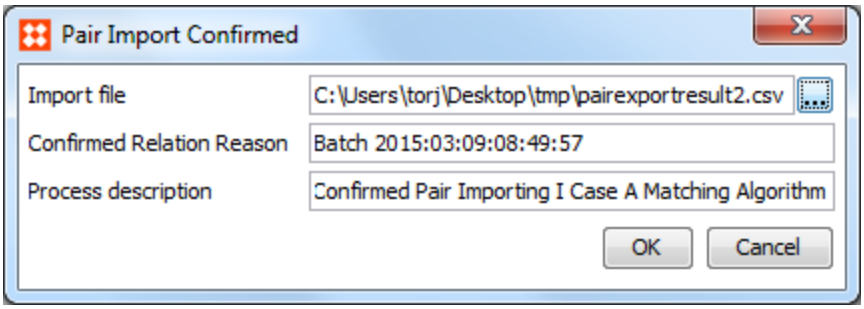
Pair Import Confirmed

Once the file exported via the pair export option has been populated with matches, it can be imported via the 'Pair Import Confirmed' option.

Rather than use the match column, the 'Pair Import Confirmed' process uses its own columns for identification purposes, but does not import any other data. This way you can avoid reverting values updated elsewhere since the pair export was performed.

Before starting the import process, consider whether to import directly into the original matching algorithm or into a copy of the original algorithm. Common setup is to import into a copy so that your original can be used for reference if needed.

To import into a copy of the algorithm, either create a new reference for 'Duplicate Type' and 'Non-Duplicate Type', or you can share the reference type between the original and the copy. If the reference types are shared between the original and the copy, the confirmed duplicates and confirmed non-duplicates will be shared as well.



In the Pair Import Confirmed dialog, specify the following:

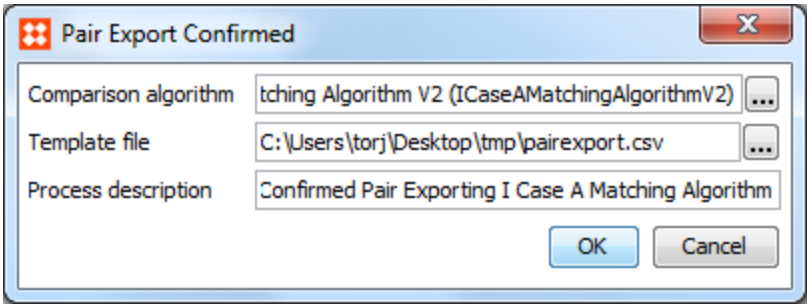
- **Import File:** Select the file that you want to import. This must be a CSV file produced by a pair export process, using a semicolon delimiter. Keep the header line.
- **Confirmed Relation Reason:** This reason is added to all confirmed relations that are created as a result of this import.
- **Process description:** Provide a description for the background process.

The file import will create 'Confirmed Duplicate' / 'Confirmed Non Duplicate' references between the pair objects.

Pair Export Confirmed

The 'Pair Export' option is used when you want to compare versions of a matching algorithm against each other and against the confirmed duplicates /non duplicates truth table constructed manually or via the steps described above.

When using this functionality, it is assumed that you have a duplicated and fine-tunedversion of your matching algorithm. Along with a template file similar to the one produced for the 'Pair Export' option, the fine-tunedmatching algorithm must be selected in the 'Pair Export Confirmed' dialog as shown below.



In the 'Pair Export Confirmed' dialog, specify the following:

- **Comparison Algorithm:** Select the matching algorithm that you want to compare the selected algorithm to.
- **Template File:** Select the file (created beforehand) that contains the required attribute values. The format of the file is the attribute IDs, separated by semicolons (;).
- **Process description:** Provide a description for the background process.

Confirmed Matches Distribution Tool

Once the background process has finished, a CSV file with the comparison results will be produced. More importantly, the background process instance will also let you inspect statistics for the comparison via a 'Show Distribution' button as shown below.

In the 'Confirmed Matches Distribution' dialog that opens when you click 'Show Distribution' (shown below), you can see for each of the matching algorithms, how well they work against the truth table. For example, how many 'True Positives', 'True Negatives', 'False Positives', and 'False Negatives' there are for each matching algorithm.

If the fine-tunedversion of the matching algorithm produces fewer 'False Positives' and 'False Negatives', you can either copy the logic to the original matching algorithm or let the fine-tunedversion replace the original one.

To access the Distribution Tool:

1. On **BG Processes**, expand **Matching Pair Export**, and then select the relevant confirmed export process.
2. At the bottom right corner, click **Show Distribution**. The **Confirmed Matches Distribution** window opens.
3. Select the checkbox to view the algorithm data in the chart.
4. From the list, select whether to view the data in a bar chart or an accumulated chart.

The table shows the following information about each algorithm:

- **Algorithm:** The ID of the algorithm.
- **Threshold:** The threshold that is used to distinguish between positives and negatives.
- **True Negative:** The number of comparisons that were classified as a non-match, both manually, and by the algorithm.
- **False Negative:** Count of comparisons that were manually classified as a match but were classified as a non-match by the algorithm because the scores were below the threshold.
- **False Positive:** Count of comparisons that were manually classified as a non-match but were classified as a match by the algorithm because the scores were above the threshold.
- **True Positive:** Count of comparisons that were classified as a match both manually and by the algorithm.

The false negatives and false positives are the errors produced by the algorithm compared to the manually reviewed pairs. Ideally, the count should be 0, which is the goal of fine-tuning the algorithms. However, even if the count is 0, it does not mean that the algorithm is perfect. The reliability of the result depends on the amount of data in the data set and the representativeness of the data.

The Bar Chart and the Accumulated Chart

The colors used in the charts are unique and identified in the chart legend. Common for both charts, green represents relations that have been manually confirmed as duplicates and red represents relations that have been manually confirmed as non-duplicates.

The threshold of the algorithm is shown as a vertical line.

Generally, the red bars are displayed to the left of the threshold indicator and the green bars to the right. If green bars are displayed to the left of the threshold indicator, they represent false negatives, and if red bars are displayed to the right of the threshold indicator, they represents false positives.

The bars only have a resolution of 1% point. Therefore, you cannot always read the exact number of false positives and false negatives directly from the graph. However, the exact number is listed in the table above it.

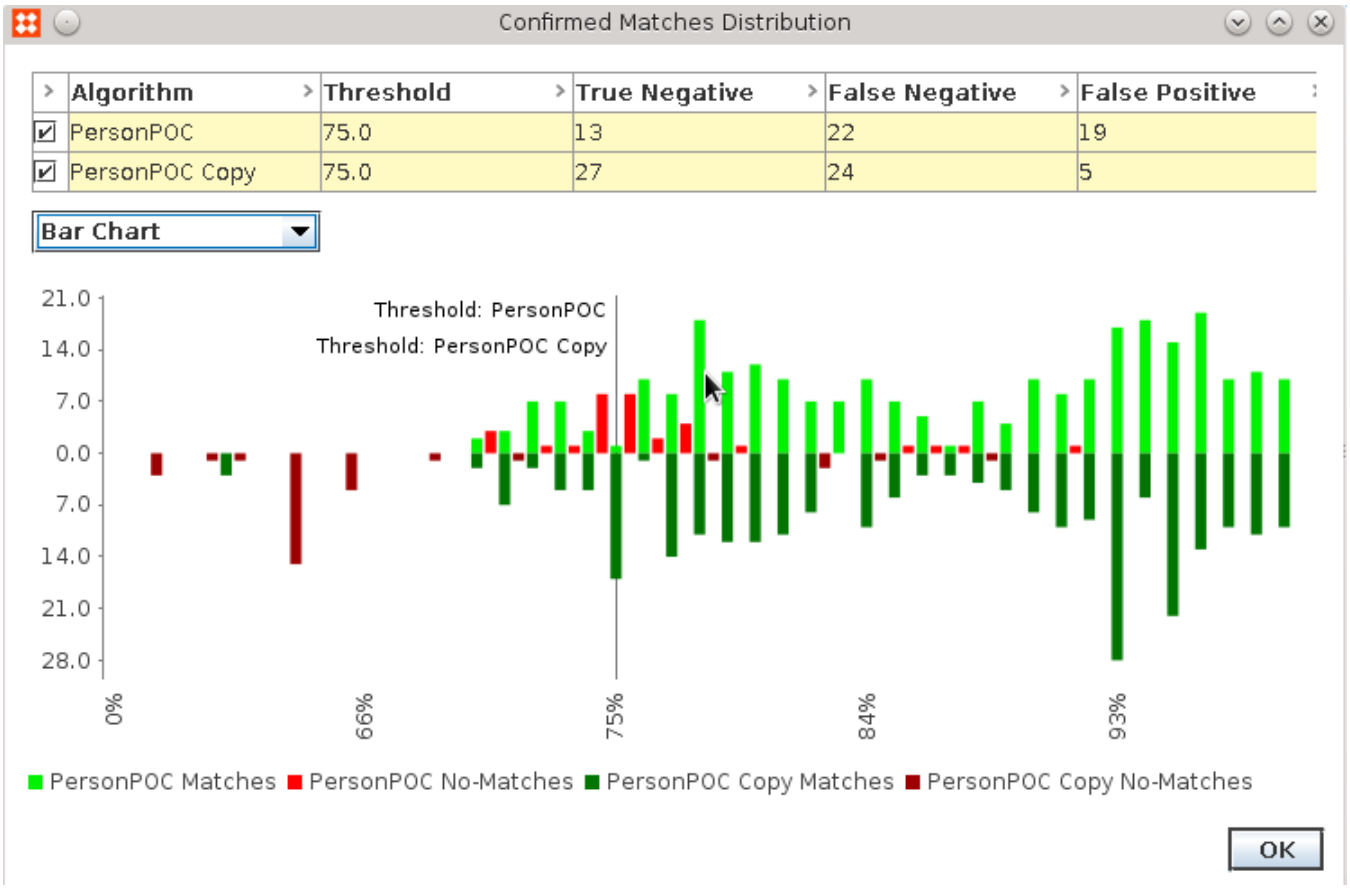
- **The Bar Chart**

  The bars in the chart show the frequency of the scores of the selected algorithm. The bar chart can either show a single algorithm or two algorithms in a special compare mode that enables a detailed comparison of the two algorithms.

  When clicking a bar, you can view a table that contains an extract of the corresponding data from the CSV file. This enables you to perform a quick inspection of the attribute values of the pairs.

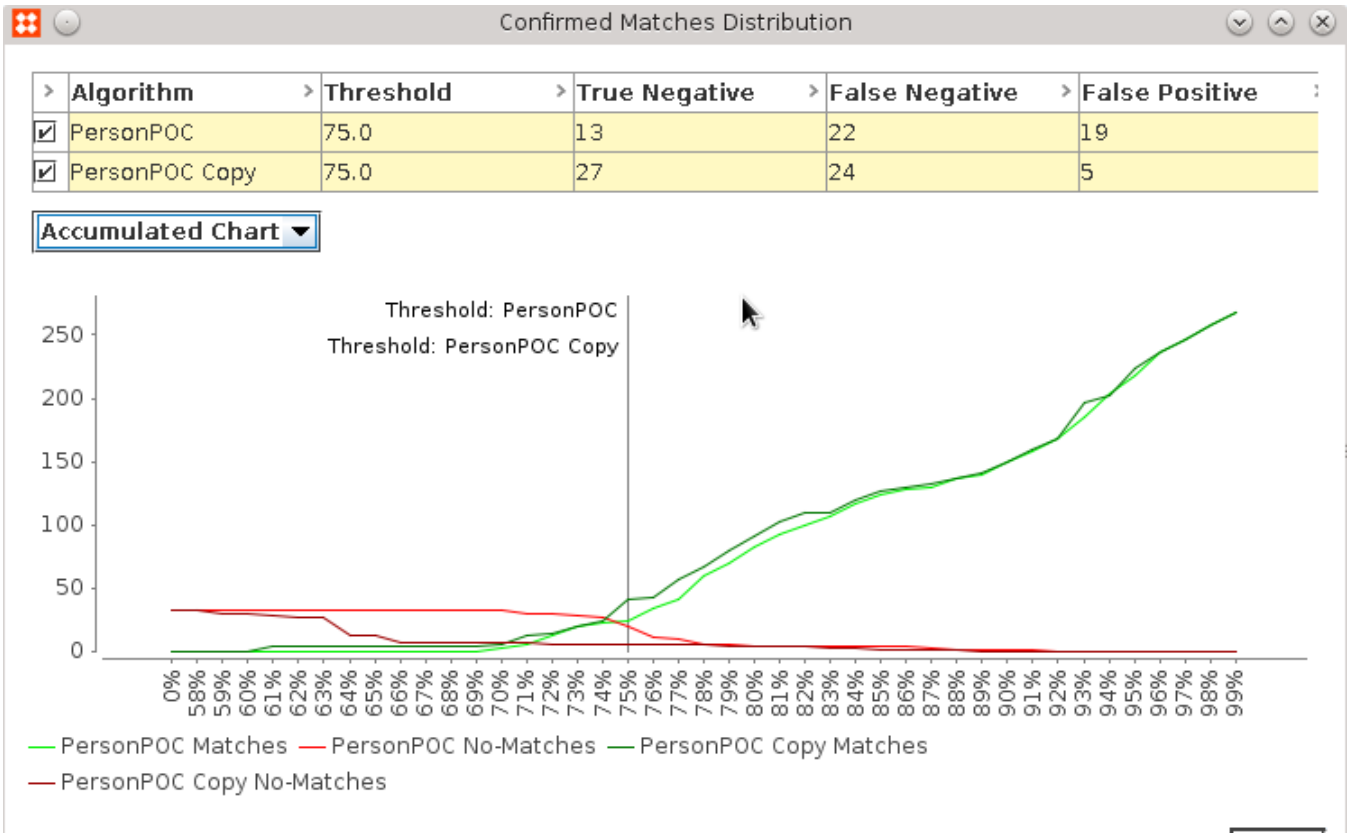
  If you need to investigate the algorithm behavior further for a given pair, click the binocular icon  . The matching algorithm editor is opened with the relevant pair selected in **System Setup**.

  You can then use the Evaluator or open the individual criteria to view more detailed information about the pair.



- **The Accumulated Chart**

  The accumulated chart shows the accumulated score frequency for the algorithms. The manually classified matches are green and accumulated to the right of the threshold line. The manually classified no-matches are red and are accumulated to the left. The accumulated chart is useful when you want to compare the matching abilities of two algorithms because it is easy to evaluate the number of scores up to a certain point. The chart is also useful for identifying a good threshold value.



Configuring Matching Event Processor

The final step in Matching and Linking Configuration is to configure a matching event processor. For information on the other required setup steps, see the **Matching and Linking Configuration** documentation.

1. Create a matching event processor by following the steps outlined in the **Creating Event Processors** section of the **Event Processors** documentation.

   This section guides you to select the type of processing required (match code generate and update and/or run matching algorithm), the matching algorithm(s) if necessary, establish the schedule, set the queue status, and set the event triggering definitions.

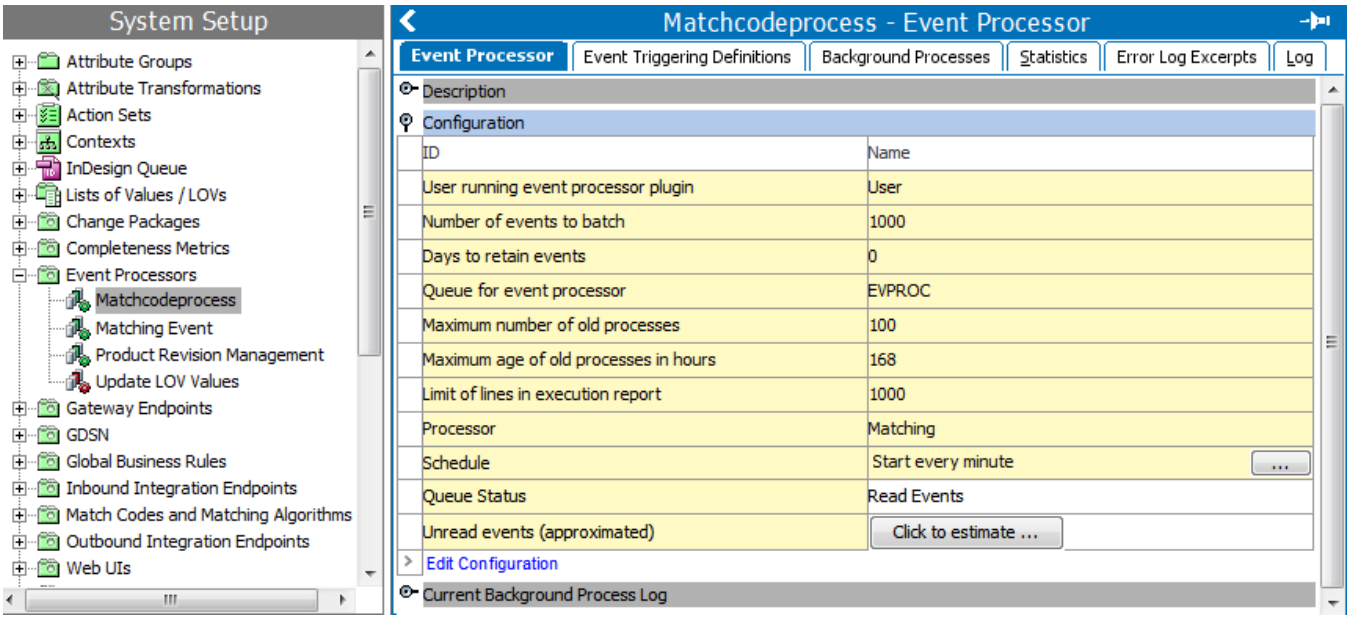
2. Enable the matching event processor by following the steps outlined in the **Enabling Event Processors** section of the **Event Processors** documentation.
3. The matching event processor is ready to run.

Matching Event Processor Example

The following example event processor has been configured to:

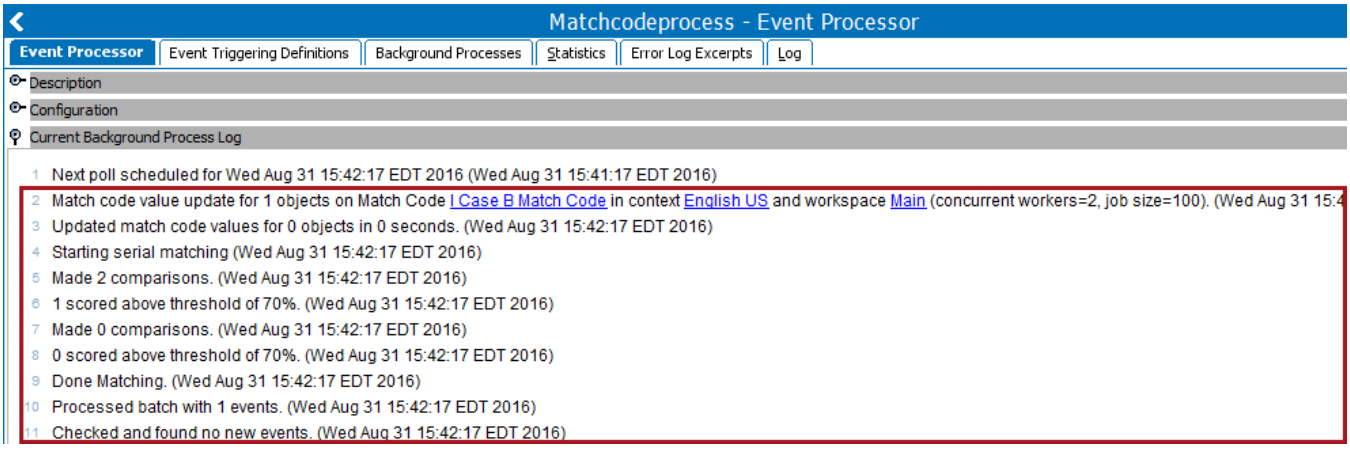
- Run every minute
- Be triggered by changes on the object types and attributes defined in the match code 'I Case B Match Code'

- Regenerate match codes using 'I Case B Match Code'
- Run the matching algorithm 'I Case B Matching Algorithm DT'

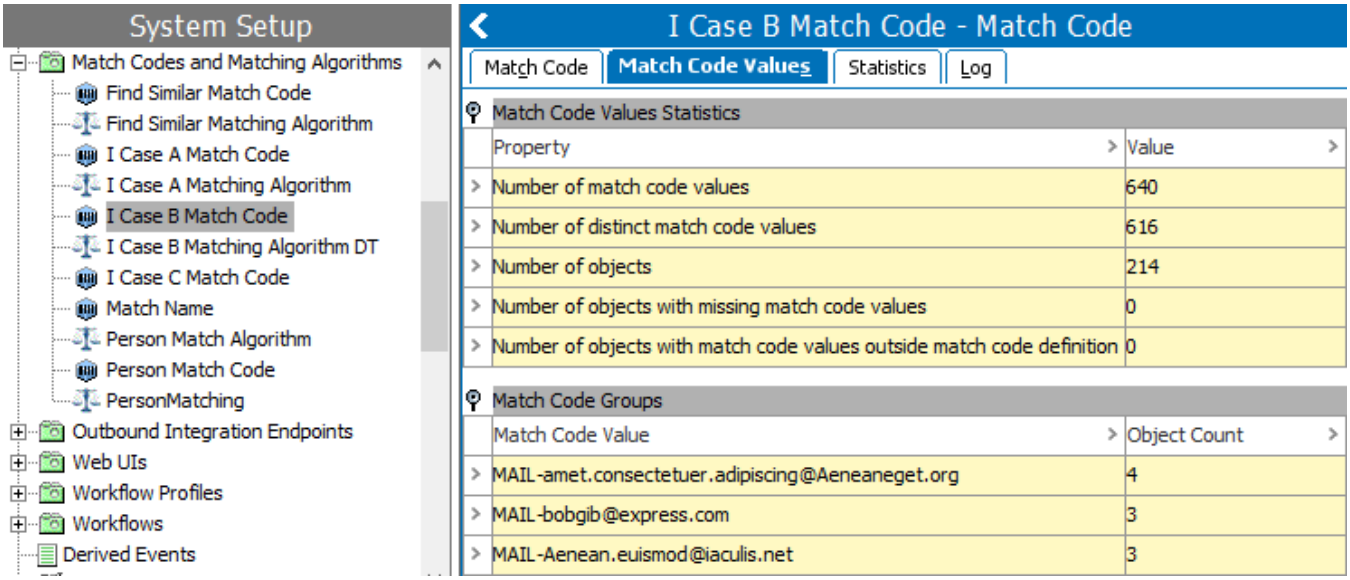


When an entity valid for the match code is updated, the event processor detects the event. On the 'Event Processor' tab, clicking the 'Click to estimate' button reports that an event exists but has not yet been read.

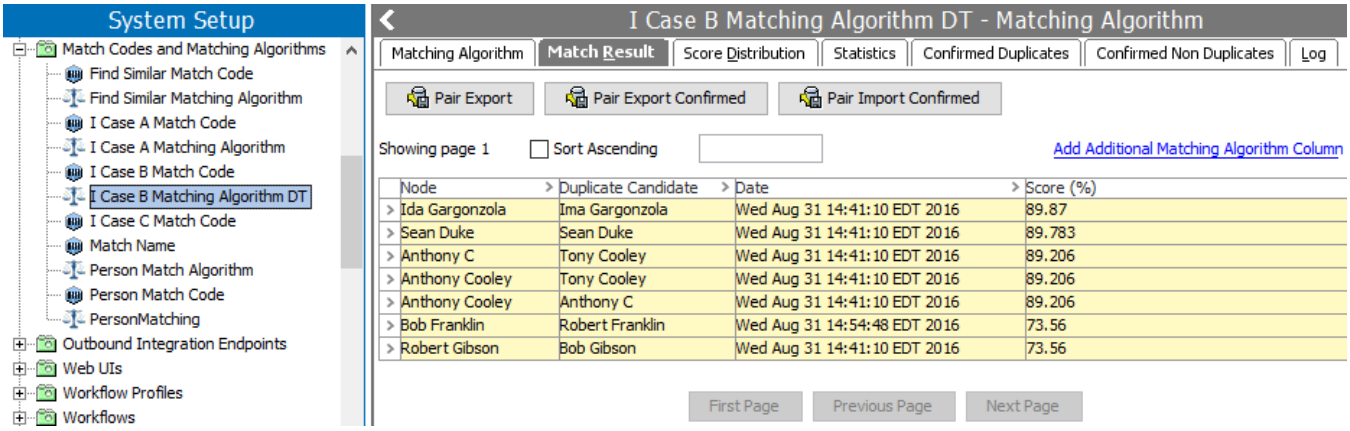
After one minute, the event processor responds to the event, updates the match code values, and then runs the algorithm to determine possible duplicates among the affected object types, based on the defined algorithm threshold.



Once complete, you can view the 'I Case B Match Code' match code. On the Match Code Values tab, open the Match Code Values Statistics flipper, and click the Yes button to calculate the statistics and display information about the existing match codes values.



You can also check which objects scored above the Match Action threshold. These objects are now defined as potential duplicates and are displayed on the 'Match Result' tab of the 'I Case B Matching Algorithm DT'.



Configuring Golden Records

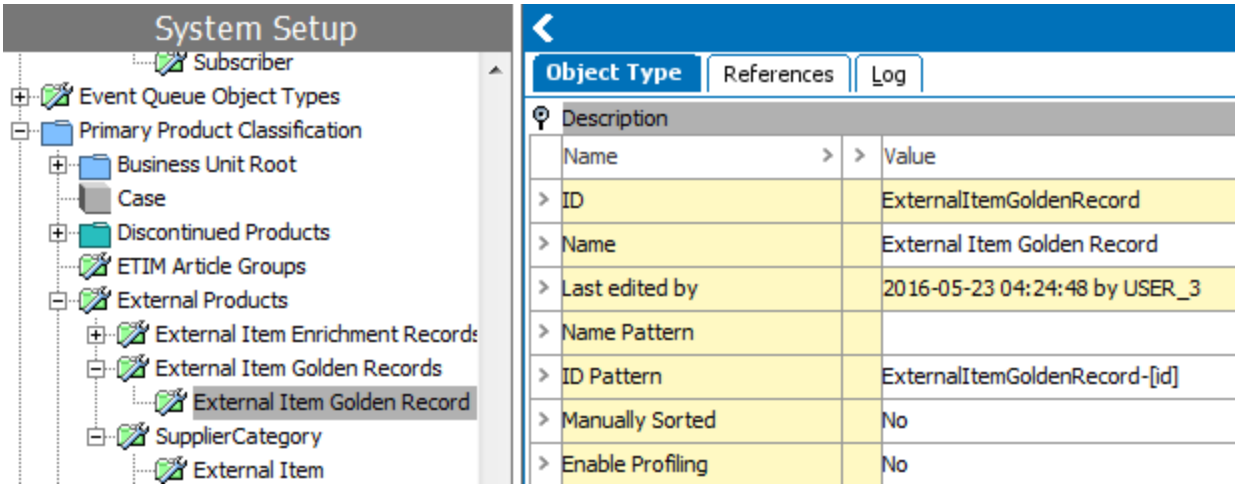
Before golden records can be generated, several setup tasks must be completed:

- A golden record object type and reference type must be created.
- A golden record root node must be created. This is where all golden records are initially populated.
- The golden record object type must be specified in the Matching Golden Record component model. For more information, see the **Component Model Configuration** section of the **Matching and Linking** documentation.

- The golden record object type, reference type, and root node must all be specified for the Golden Record Match Action of the applicable matching algorithm.
- The auto threshold and clerical review threshold must be specified in the Golden Record Match Action.
- A clerical review workflow may be configured. For more information, see the **Clerical Review** section of the **Matching and Linking** documentation.

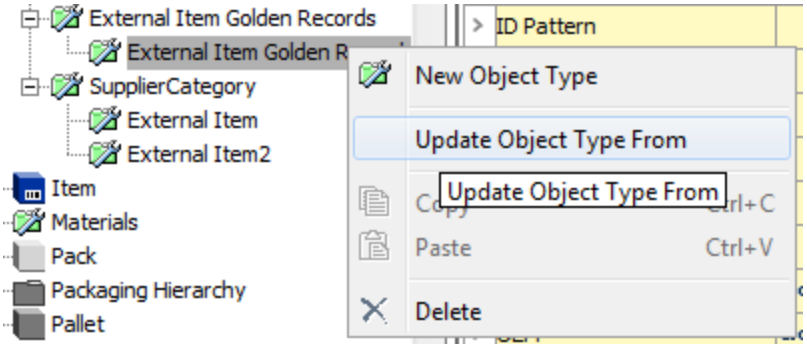
Golden Record Object Type

Golden records cannot share the same object type as their source objects, so it is necessary to create an object type for golden records specifically. Considering how golden records are generated, this object type must have an auto ID pattern configuration.

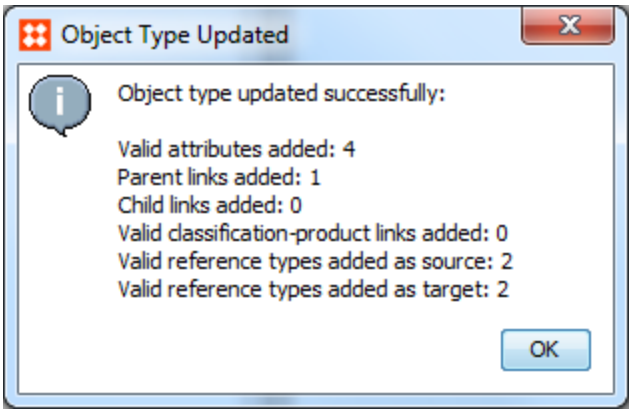


If you intend to copy all data (attribute values and references) from the source objects, the golden record object type should be valid for the same attributes and be a valid source for the same reference / link types. The 'Update Object Type From' operation can be performed to simply copy this information from the source object:

1. Navigate to the golden record object type node and right-click it.
2. In the dropdown, click 'Update Object Type From'



3. Pick the applicable object type from the selector screen and then click **Select**.



If you choose to copy the information from the source object, some cleanup will be necessary. For example, the golden record object type should not be a legal target for the Duplicate and Non-Duplicate Reference Types. Additionally, the golden record object type should not be valid below the same object types as the source objects.

For more information on creating object types, see the **Object Types and Structures** section of the **System Setup / STEP Super User Guide** documentation.

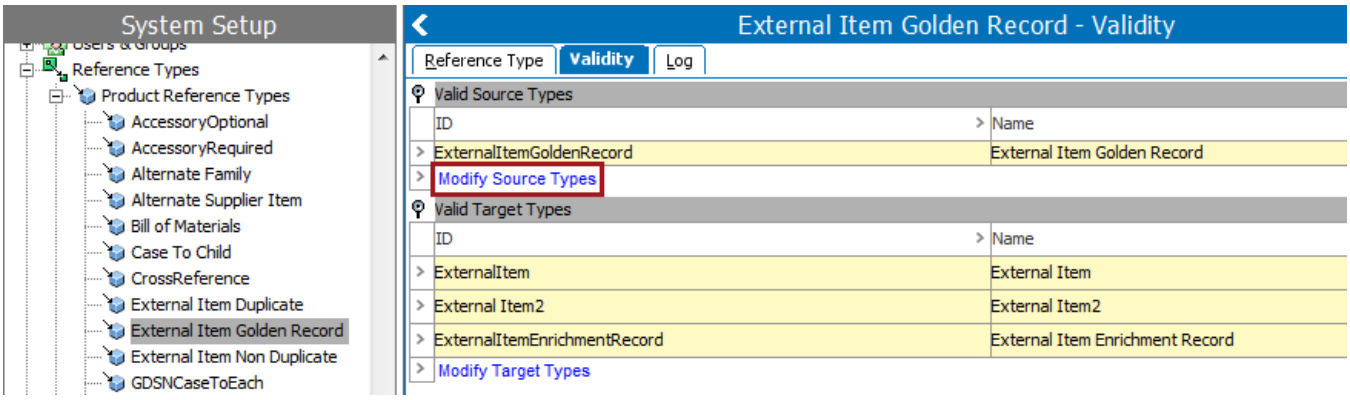
Once created, the golden record object type must be selected in the Matching Golden Record component model.

For more information on the component model configuration, see the **Component Model Configuration** section of the **Matching and Linking** documentation.

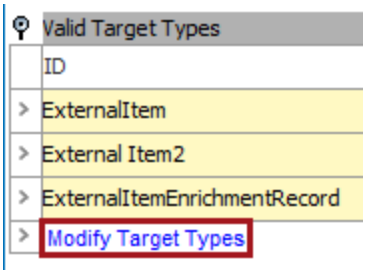
Golden Record Reference Type Validity

In order for golden records to reference back to their source objects, a golden record reference type must be created. The reference type must allow for multiple targets and should be valid from the golden record object type to the source record object type. To configure the validity:

1. On the reference type node, navigate to the 'Validity' tab.
2. In the 'Valid Source Types' area, click the **Modify Source Types** link.



3. In the selector that appears, choose the golden record object type. Then click **OK**.
4. In the 'Valid Target Types' area, click the **Modify Target Types** link.



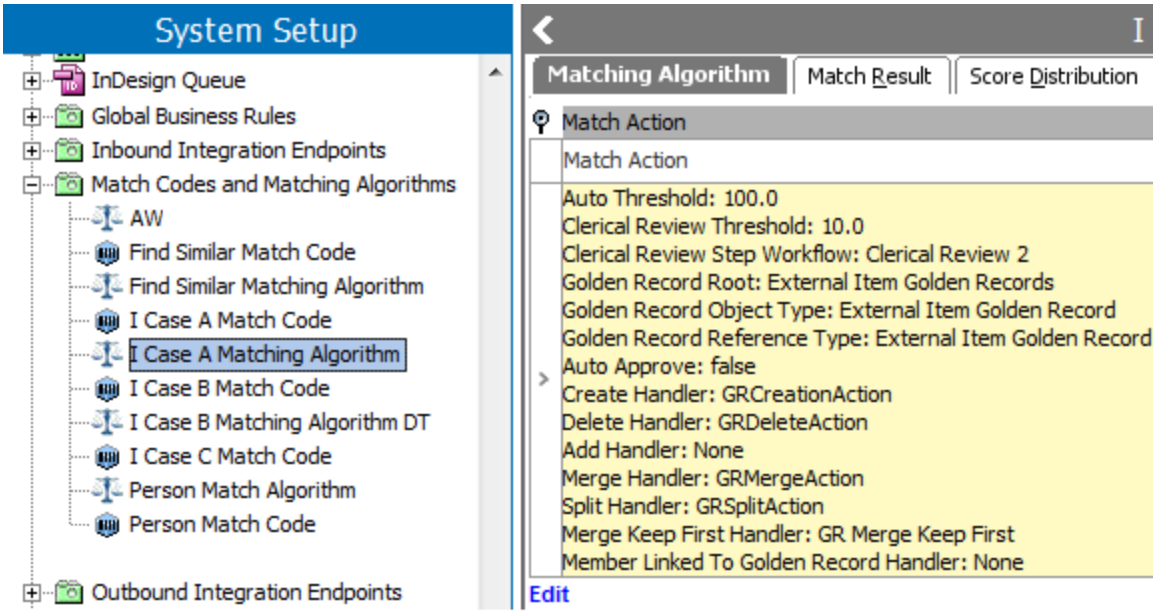
5. In the selector that appears, choose the object types the golden records should reference back to. Then click **OK**.

For more information on creating reference types, see the **About Reference Types** section of the **System Setup / STEP Super User Guide** documentation.

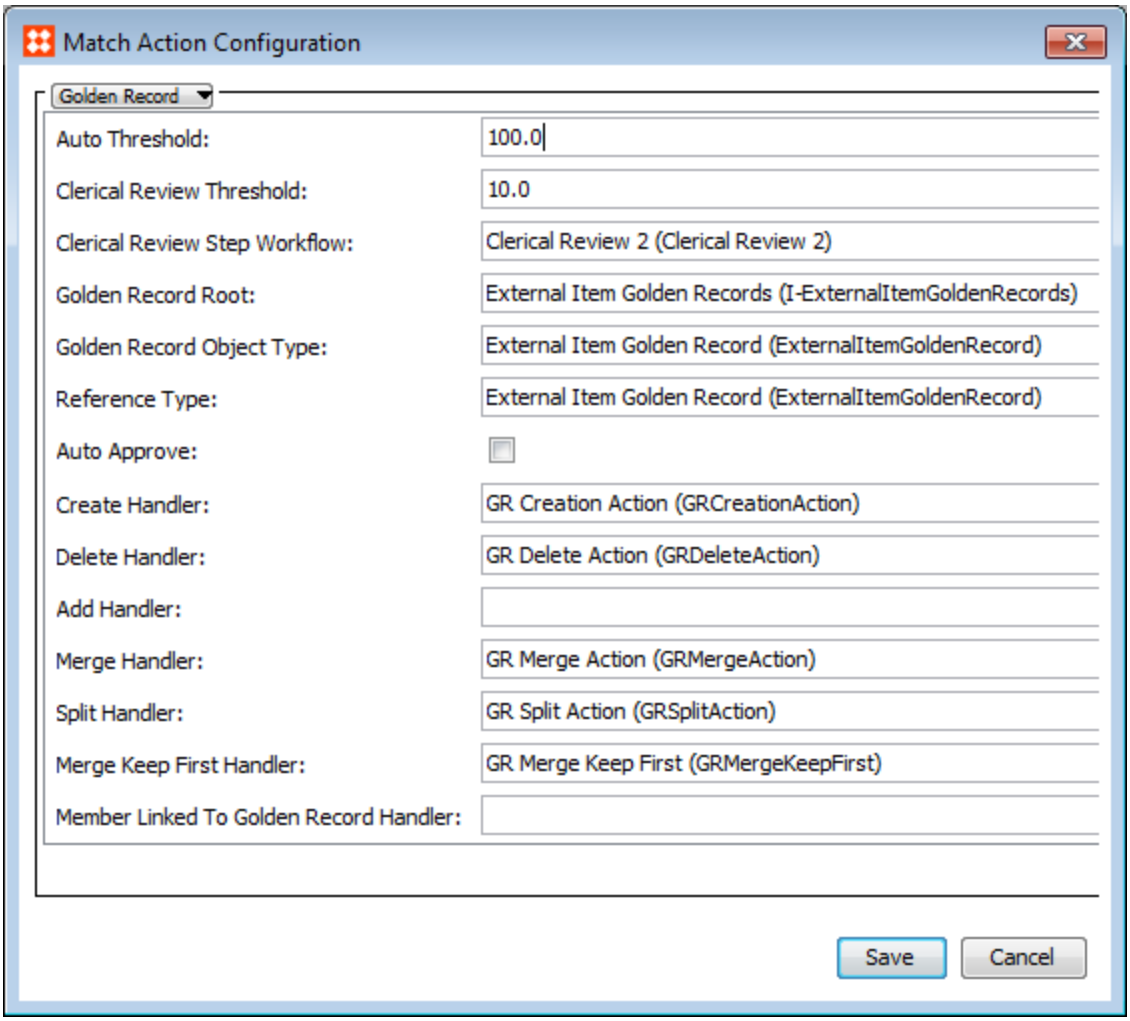
Golden Record Match Action

In order to generate golden records from the desired matching algorithm, the golden record match action must be configured:

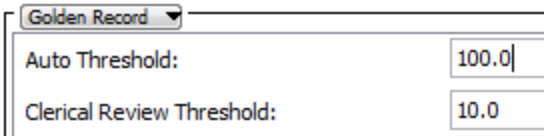
1. On the matching algorithm node, navigate to the 'Matching Algorithm' tab.
2. In the 'Match Action' area, click the **Edit** link.



Once on the 'Match Action Configuration' screen, several parameters must be configured.

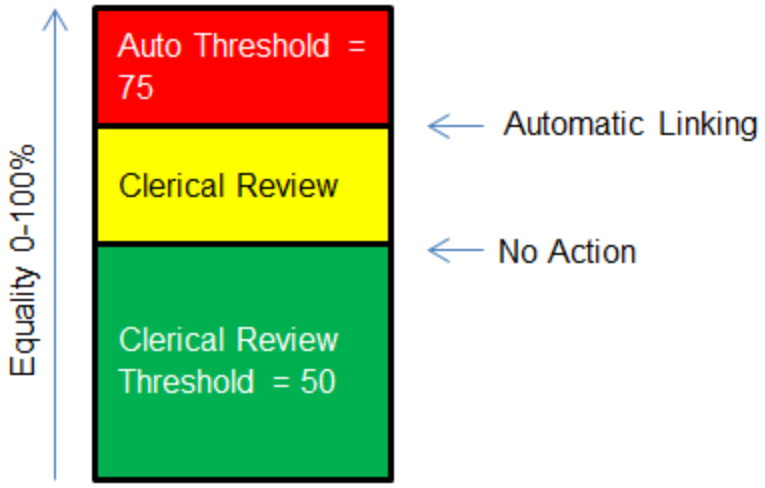


3. For the 'Auto Threshold' and 'Clerical Review Threshold' parameters, specify how matches will be evaluated via an equality measurement.



- The 'Auto Threshold' specifies how equal two source objects have to be in order to automatically have them share the same golden record.
- The 'Clerical Review Threshold' should be equal to or less than the 'Auto Threshold'. It specifies how equal two objects must be to be considered possible duplicates.
- Objects with an equality metric between the 'Clerical Review Threshold' and the 'Auto Threshold' will initially be referenced from separate golden records. If a user then confirms that the objects are in fact duplicates, a

matching algorithm-specific reference will be created between them and they will be made to share the same golden record.



5. In the 'Clerical Review STEP Workflow' parameter, if a clerical review workflow was created, select it.

   For more information, see the **Clerical Review** section of the **Matching and Linking** documentation.

6. For the 'Golden Record Root', 'Golden Record Object Type', and 'Reference Type' parameters, specify the applicable golden record root node, golden record root object, and golden record reference type respectively.

For more information of configuring the match action handlers, see the **Updating Golden Records** section of the **Matching and Linking** documentation.

Clerical Review

After running a matching algorithm, some paired objects may need to be manually reviewed to determine whether or not they are actual duplicates. An object is flagged for clerical review if its equality metric falls between the 'Clerical Review Threshold' and the 'Auto Threshold', as defined by the algorithm's golden record match action. It is subsequently placed into a workflow for review by an end user.

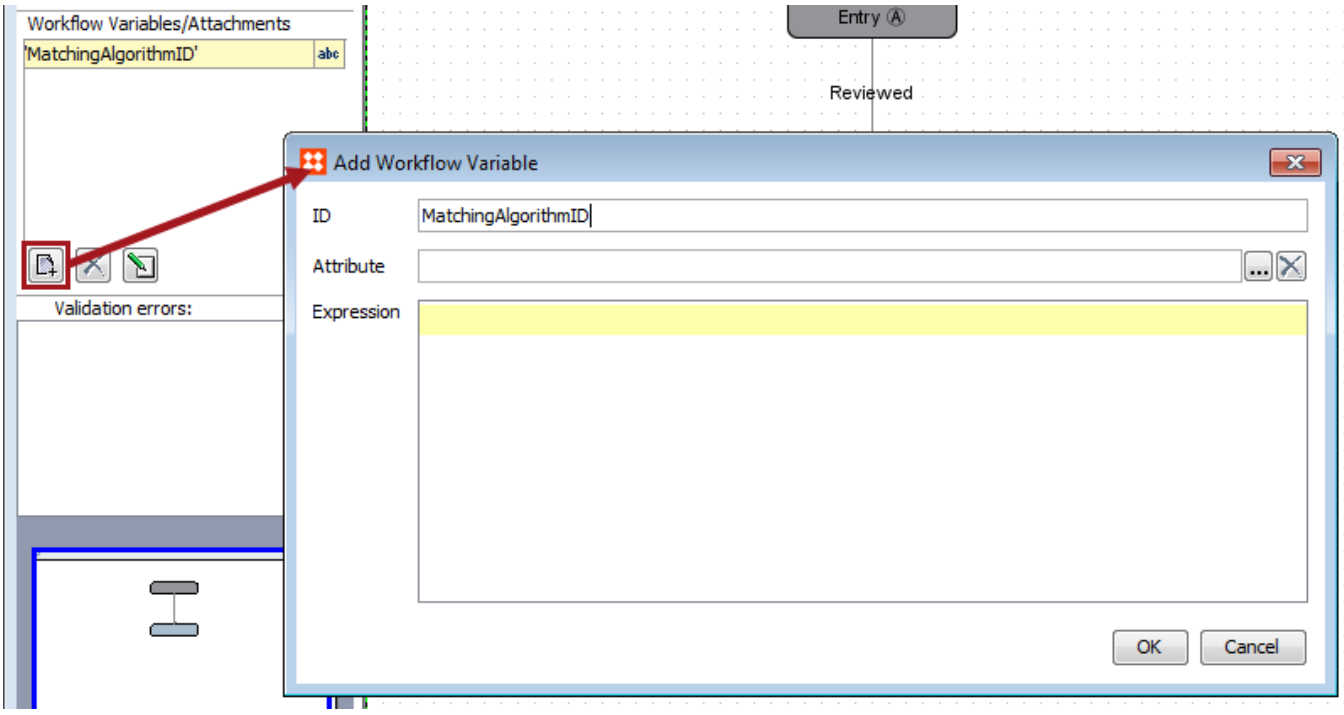
When configured for clerical review, a workflow allows users to easily review and confirm or reject potential duplicate objects.

For more information about the golden record match action, see the **Configuring Golden Records** section of the **Matching and Linking** documentation.

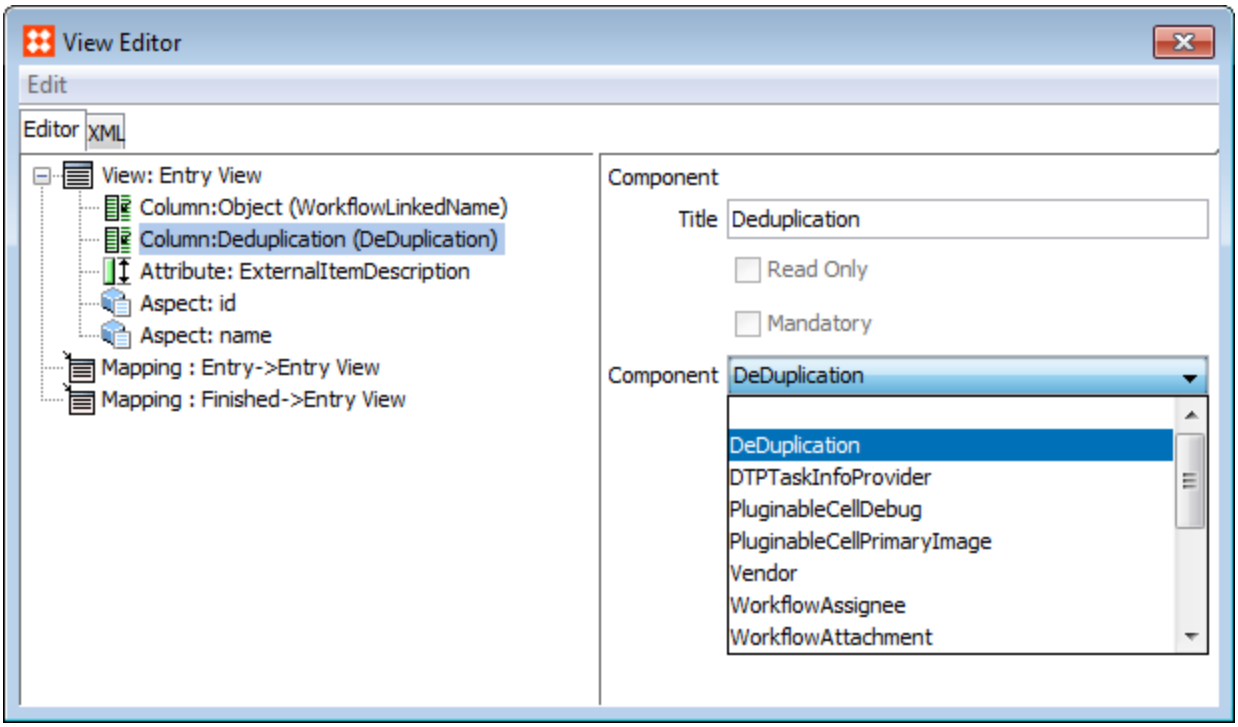
Configuring a Workflow for Clerical Review

To configured a workflow that can handle clerical review, the following steps need to be followed:

1. Create the workflow that will handle the clerical review. See **Creating a Workflow** in the **Workflows** documentation for more information.
2. Once configured, in the 'Workflow Variables / Attachments ' area of the workflow editor, click the 'Add' icon and select 'Add a Workflow Variable'. In the 'ID' field, enter 'MatchingAlgorithmID', then click **OK**.



3. Under 'Edit' > 'Edit Workbench Views and Mappings', right-click the relevant view in the editor tab and select 'Add Column' > 'Component'. In the 'Component' dropdown selector, select 'DeDuplication'.



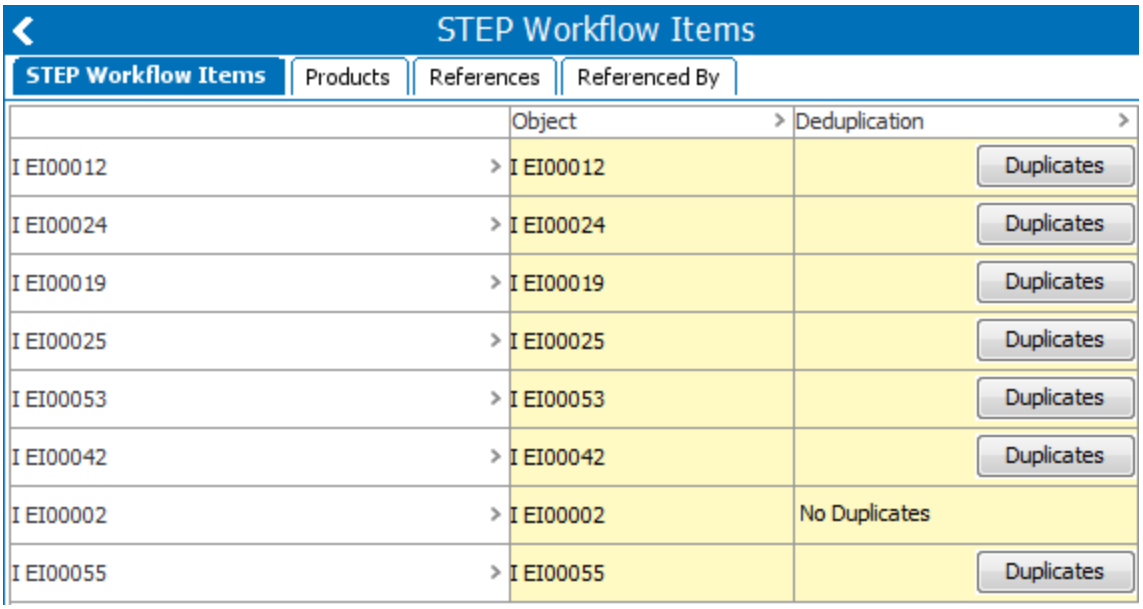
4. After saving the workflow, navigate to the relevant matching algorithm. In the 'Match Action' area of the 'Matching Algorithm' tab, click 'Edit'. For the 'Clerical Review Step Workflow' parameter, click the ellipsis button **(…)** and choose the workflow you configured from the selector. Click **Save** to save changes to the algorithm.

5. The next time the relevant match codes are generated and the matching algorithm is run, objects that fall between the 'Clerical Review Threshold' and the 'Auto Threshold' will be initiated into the workflow.

Working with Items in the Workflow

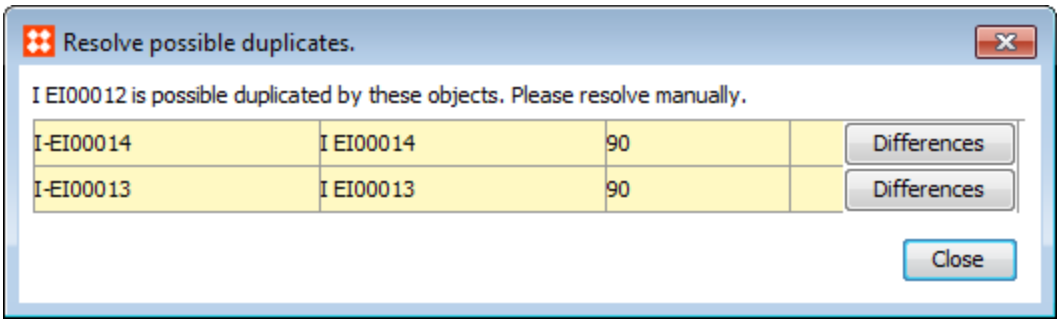
To view any items placed into the workflow configured for clerical review, navigate to the **STEP Workflow** tab and expand the flipper for the appropriate workflow.

1. Select the desired state to see which objects require a clerical review.

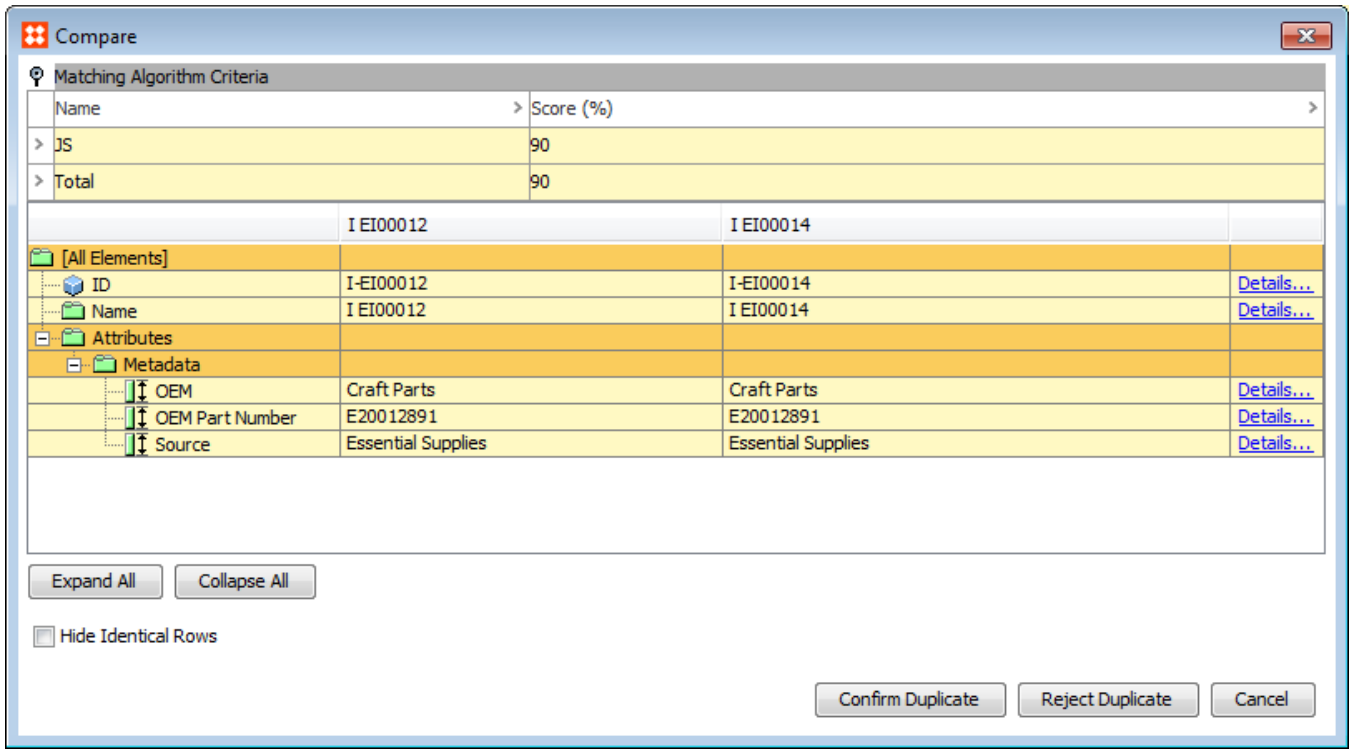


2. To review an object, click the **Duplicates** button. A window will appear that lists all potential duplicates for the selected object and the equality metric for each pair.

Note: This can also be accessed via the 'Tasks' tab of the relevant object in **Tree**.



3. To compare the selected object with one of its potential duplicates, click the **Differences** button. A window will appear comparing the attributes of each object. From here the user can confirm or reject that they are duplicates via the **Confirm Duplicate** and **Reject Duplicate** buttons.



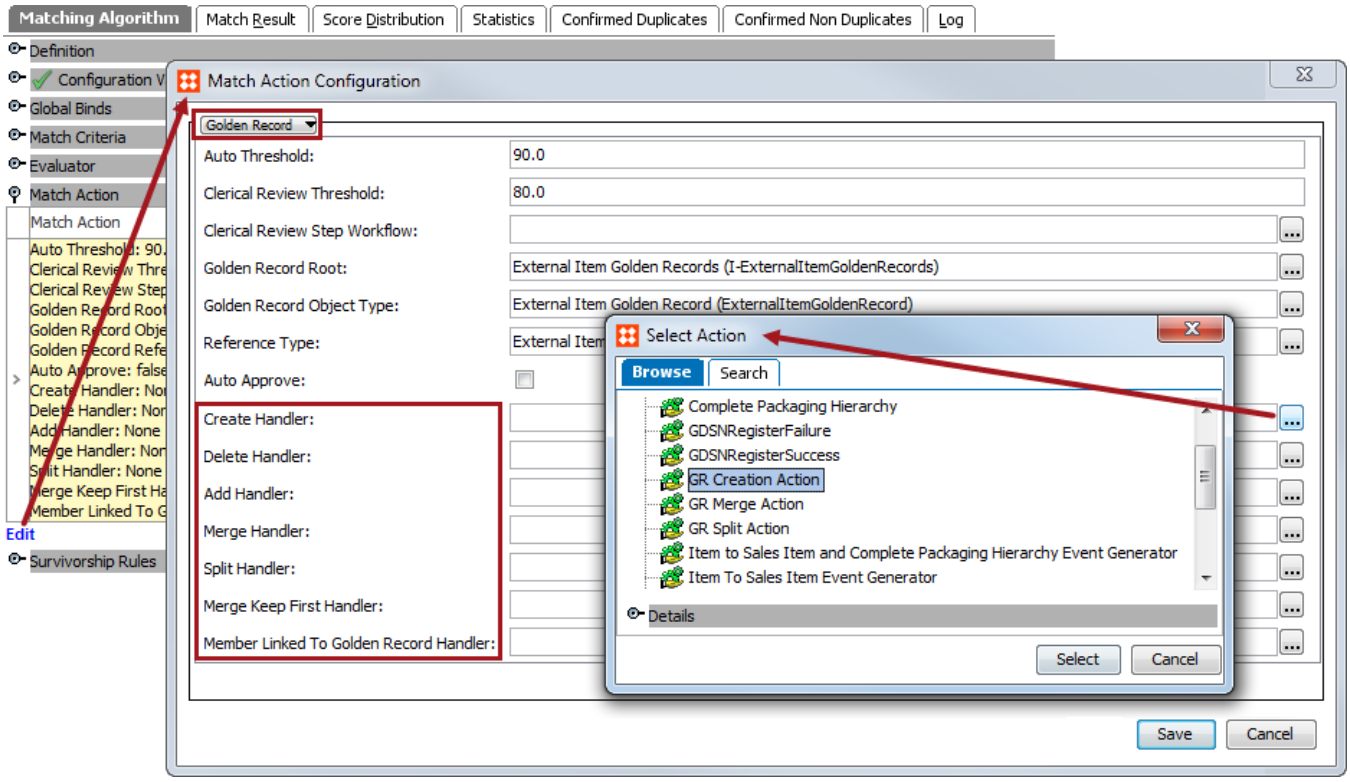
Clerical reviews can also be performed via workflows in Web UI. For more information, see the **Configuring a Deduplication Clerical Review** section of the **Matching and Linking** documentation.

Updating Golden Records

Working with a golden record setup often requires specific actions when a golden record is changed (created, deleted, merged, split, etc.). In these cases, the matching algorithm can be configured to call a business rule via a handler in order to allow for more granular processing of events. For example, when two existing golden records are merged, additional actions may need to occur in addition to the survivorship rules.

Golden Record Handlers

Based on the requirement of the handler as defined below, an existing business action or condition can be selected. For more information on creating a new business rule, see the **Business Rules** section of the **Business Rules** documentation.



The following handlers are available for Golden Record Match Actions:

- **Create Handler**: The selected business action runs on the golden record after it has been created and has initial source object links, but before survivorship rules run.
- **Delete Handler**: The selected business action runs after the golden record is deleted. For example, when merging two golden records, one is deleted. The delete handler runs after the merge handler, which means that the golden record has no linked source records. Alternatively, in this case, if the delete handler field is blank, then the incoming references of the surviving golden record are re-targeted and re-approved (if they were approved before); the golden record is deleted and, if auto-approve is enabled, the deletion is approved.
- **Add Handler**: The selected business action runs on the golden record after a new source is added, but before any survivorship rules run.
- **Merge Handler**: The selected business action runs when two golden records are merged (because their sources match). The source(s) are moved to the golden record that will be kept and the delete handler is called for the golden record that will be deleted.
- **Split Handler**: The selected business action runs when a golden record is split (because one or more of its sources no longer match). The split handler runs after the new golden record is created and its source record links are updated, but before survivorship rules run. The original and new golden records each reflect the correct source records. The create handler is not called when golden records split.

- **Merge Keep First Handler**: The selected business condition runs when two golden records are being merged and allows identification of the golden record that should be kept. Use the Current Object and Secondary Object binds in the condition and return one of the following options:
  - null = default behavior; keep the golden record with most members, if there is an equal number, keep the oldest golden record
  - true = golden record bound to “Secondary Object” is deleted
  - false = golden record bound to “Current Object” is deleted
- **Member Linked to Golden Record Handler**: The selected business action runs on the source object when a source object link changes from one golden record to a new golden record. The handler runs after the sources have been added, but before survivorship rules run.

For information about writing business rules that require access to two objects (for example, two golden records in merge and split cases), see the JavaScript **Secondary Object Bind** documentation.

Internal Data Source Objects

In golden record setups, there will typically be a need for having objects related to golden records, on which it is possible to maintain data. Such 'enrichment records' or 'Internal Data Source Objects' should be created in accordance with the following rules:

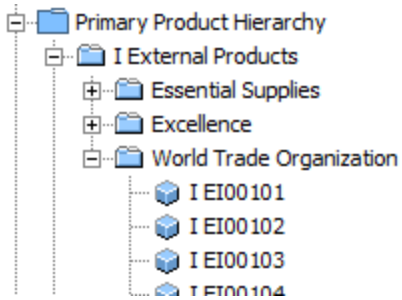
- Use a unique object type that is different from the object types of golden record and other source objects.
- Do not generate match codes for internal data source objects.
- In the Matching Component Model Configuration, Source Object Type aspect, add the Object Type of the internal data source object.
- Golden records should use the same reference types for internal source objects and for other source objects.

Use the following setup to update the golden record when an internal data source object changes:

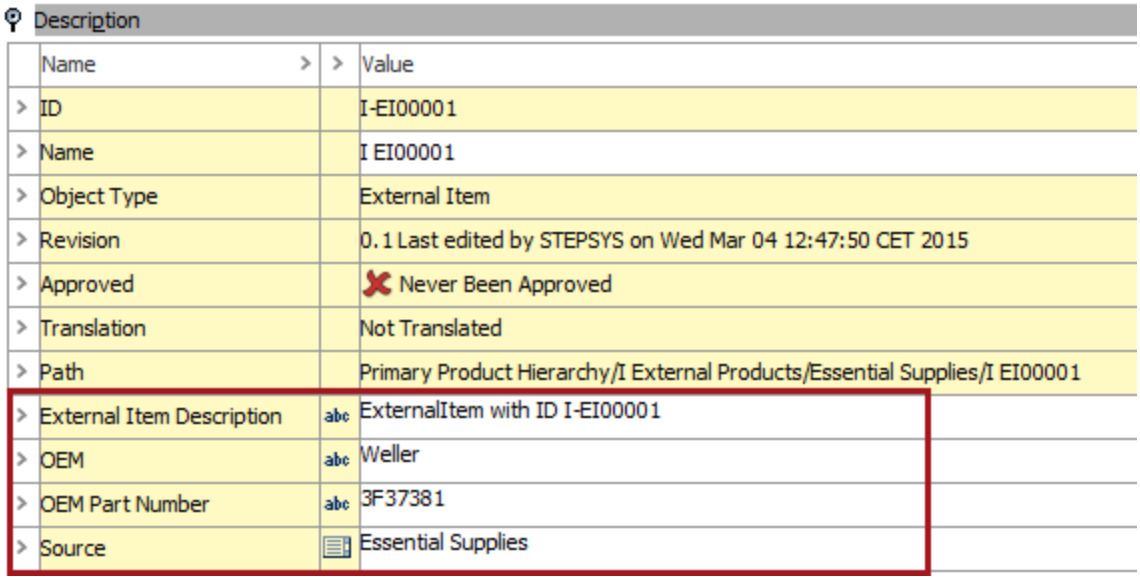
1. Configure the event processor to listen on events for internal data source objects.
2. Create a business action to find the golden record for the internal data source object, identify one of the other source objects for the golden record, and then generate an event for that object for the event processor.
3. Create an event filter condition that is always false since the original event for the internal data source object will not go onto the queue.

Configuration Example - Basic

This example use case concerns deduplication of products from different suppliers. The products are of the object type 'External Item' and live below an 'External Products' product category node, below which they are again organized by supplier.



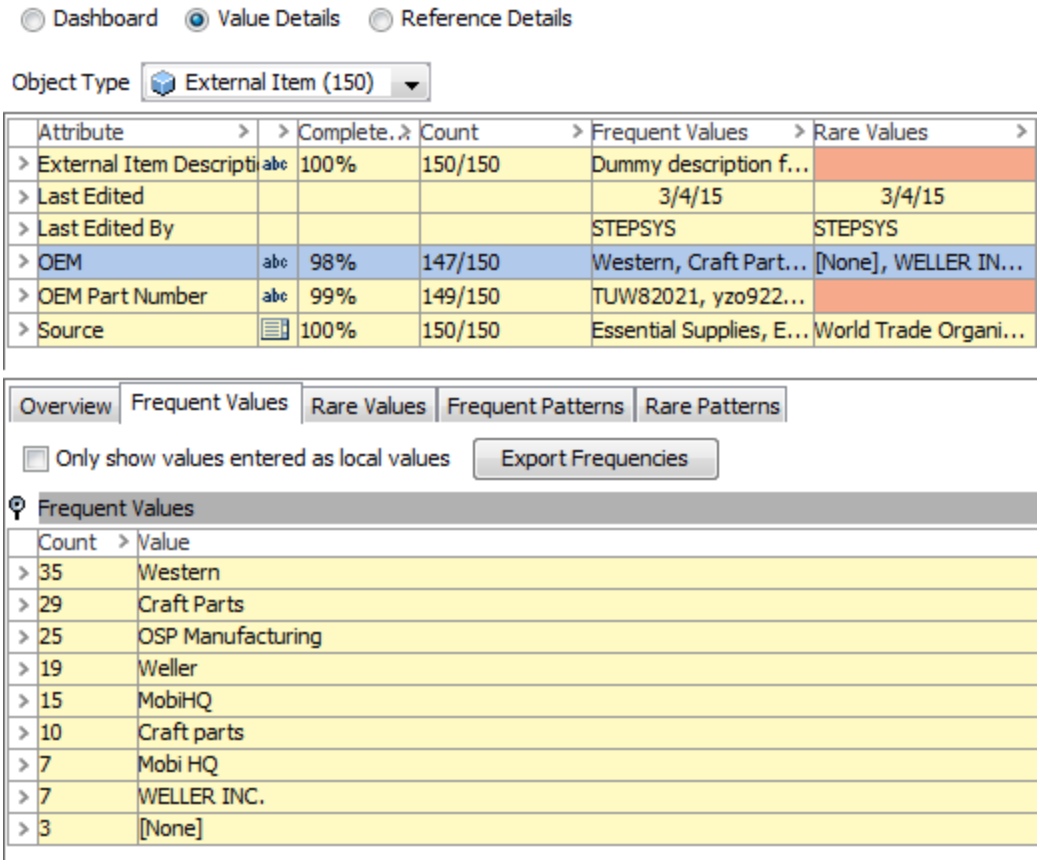
Each External Item has four significant attributes: 'OEM' (Original Equipment Manufacturer), 'OEM Part Number', 'Source' (supplier), and 'External Item Description'. The main objective is to identify duplicates based on the OEM and OEM Part Number attributes, i.e., the items are duplicates if they have the same OEM and OEM Part Number.



Data Profile Analysis

Designing a deduplication strategy requires intimate understanding of the data, and to that end STEP Data Profiles can be of great help. Data profiles show to what extent relevant attributes are populated, and can highlight the most frequent and rare values and patterns.

If a profile is generated from the 'External Products' node, it is possible to see that there are missing values for both OEM and OEM Part Number, and this should be accounted for in the deduplication strategy. Furthermore, as illustrated below, the profile shows that the OEM values include obvious duplicates like 'Craft Parts'/'C'raft part'" and'"Welle'"'"WELLER INC'" indicating that some form of normalization is required.



For OEM Part Number, there are more than 100 distinct values and thus the profile does not, with the default settings, provide exact statistics. Still, it is possible to see that both uppercase and lowercase letters are used and that punctuation is used in some values and not in others. Again, this indicates that normalization will be required.

OEM Part Number	abc	99%	149/150	TUW82021, yzo922...	
Source		100%	150/150	Essential Supplies, E...	World Trade Organi...

Overview | Frequent Values | Rare Values | Frequent Patterns | Rare Patterns

Only show values entered as local values | Export Frequencies

Frequent Values

Count	Value
2	TUW82021
2	yzo92281
2	E200 7591
2	TUW50131
2	95H92971
2	D4-87751
2	95H2581
2	D4-9551
2	YZO61421
2	d4-77601
2	TUW21681

Looking at the frequent patterns info, there are no clear distinct patterns in the values.

Overview | Frequent Values | Rare Values | Frequent Patterns | Rare Patterns

Only show patterns for local values | Export Frequencies

Frequent Patterns

Count	Pattern
26	AAA99999
14	AAA-99999
13	A999A-99999
12	AAAAA99999
12	AA-99999
11	99A99999
10	A99999
10	A9-99999
10	A999 99999
5	A99999999
5	9A99999
5	A999999
3	AAA9999
2	A9999
2	9A9999
2	AAA-9999
1	A999A-9999
1	A999A99999

With two 'matching' attributes, it would be possible to generate two match codes per object, but for this case, this is likely not the best strategy. Thus, the number of different OEM values is quite low, especially if they are normalized, and comparing all items from the same OEM would result in too many comparisons.

As there is a significant spread in OEM Part Number values, generating match codes based solely on these could work. Additionally, you also want a match on OEM and cannot tie a specific OEM Part Number value pattern to an OEM (a match on OEM Part Number is not necessarily a true match). However, this would require that the matching algorithm logic inspected the OEMs later to determine if there is a match or not.

A possible solution is to generate composite match codes that include information from both attributes. If the values are normalized during the match code generation it will, with this approach, be possible to simplify the setup so that identical match codes are automatically considered a match. This can be achieved by working with a Window Size of 1 (only compare objects with the same match code) and have matching algorithm logic that does not check anything, but simply for each comparison, indicates that there is a match.

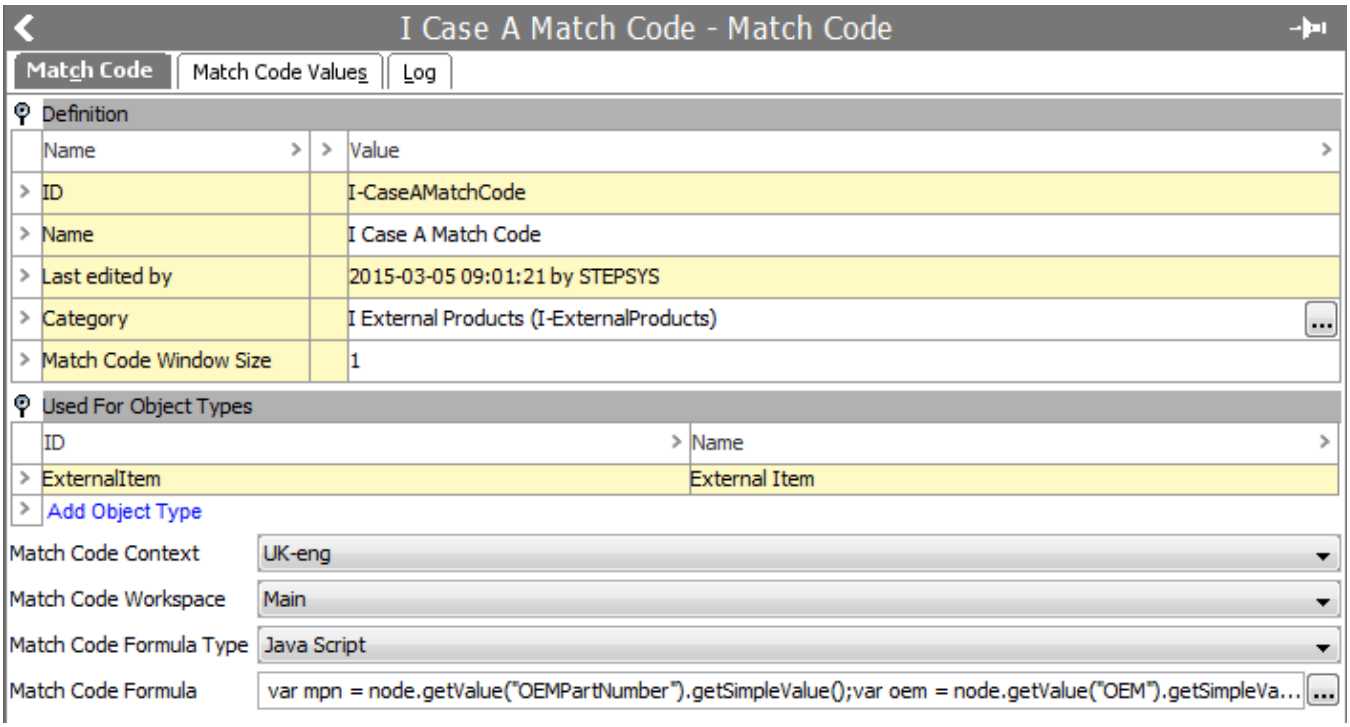
For more information on data profiles, see the **Data Profiles** section of the **Data Profiling** documentation.

Match Code Configuration

The JavaScript below can be used for match code generation ('Current Object' is assumed bound to 'node'). Here, a match code is only generated if there are values for both attributes (two items with missing values are not a match) and basic normalization is applied. For OEM Part Number, punctuation and spaces are removed and the values put in lowercase. For OEM, the same two operations are applied and furthermore, only the first four characters of the values are used, and are separated by a colon (:).

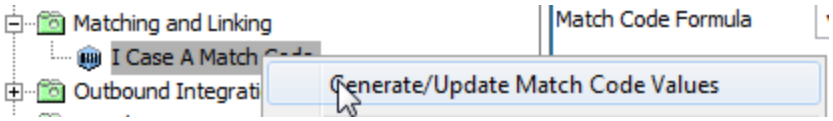
```
var mpn = node.getValue("OEMPartNumber").getSimpleValue();
var oem = node.getValue("OEM").getSimpleValue();
if(oem && mpn) {
   mpn += ""; // Converts to JS string
   mpn = mpn.replace(/[^\w]|_/g, ""); // Leaves only letters and digits
   mpn = mpn.toLowerCase(); // Converts to lowercase
   oem += ""; // Converts to JS string
   oem = oem.replace(/[^\w]|_/g, "");  // Leaves only letters and digits
   oem = oem.substring(0, 4).toLowerCase(); // First 4 characters in lowercase
   return mpn + ":" + oem
}
else {
   return "";
}
```

For an item with OEM Part Number 'D4-90581' and OEM 'Mobi HQ', a match code 'd490581:mobi' would be generated. Likewise, an item with OEM Part Number 'E200 173' and OEM 'Craft Parts' would get a 'e200173:craf' match code, and so forth.

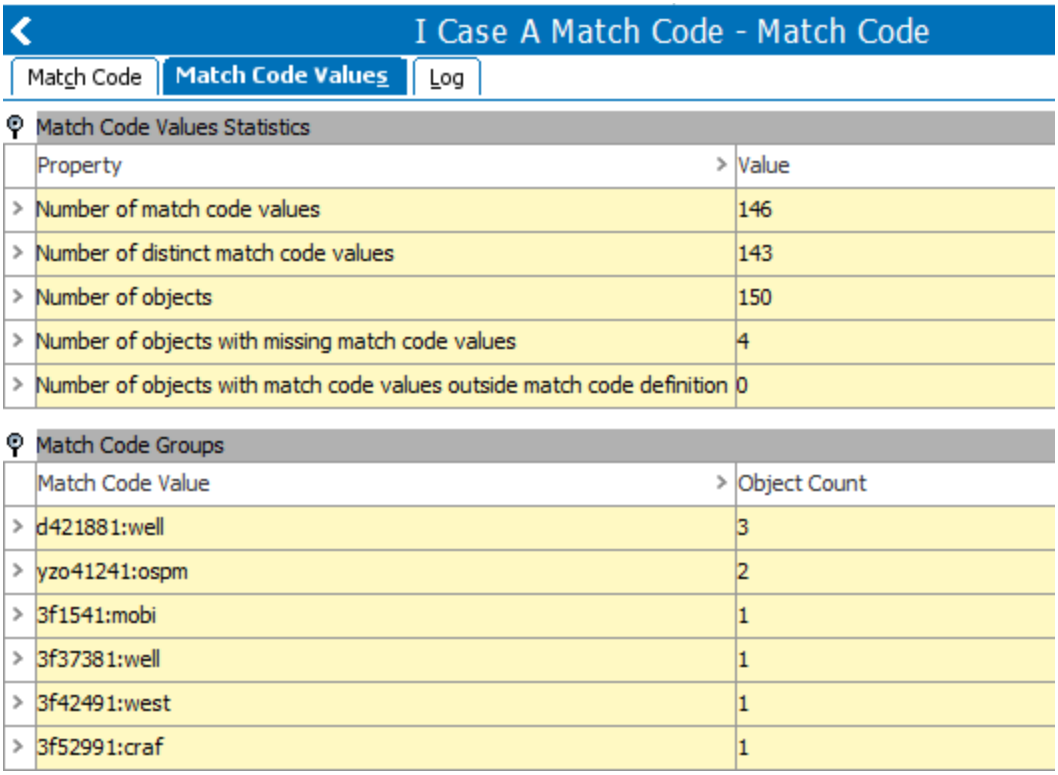


For more information of configuring match codes, see the **Configuring Match Codes** section of the **Matching and Linking** documentation.

When the match code definition has been configured, match codes can be generated from the match code object context menu as shown below.



Following this, statistics and most common match codes can be inspected via the 'Match Code Values' tab.



I Case A Match Code - Match Code

Match Code | Match Code Values | Log

Match Code Values Statistics

Property	Value
Number of match code values	146
Number of distinct match code values	143
Number of objects	150
Number of objects with missing match code values	4
Number of objects with match code values outside match code definition	0

Match Code Groups

Match Code Value	Object Count
d421881:well	3
yzo41241:ospm	2
3f1541:mobi	1
3f37381:well	1
3f42491:west	1
3f52991:craf	1

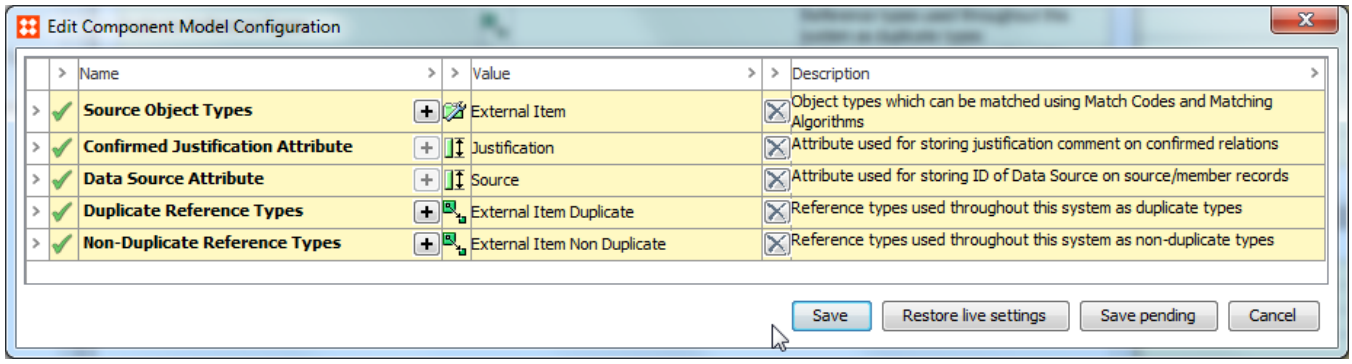
From here it is possible to see that there are duplicates in the set as three items have the Match Code 'd421881:well' and two items have 'yzo41241:ospm'.

Matching Algorithm Configuration

Matches can be compared, rejected / verified, and potentially merged by configuring and applying a matching algorithm.

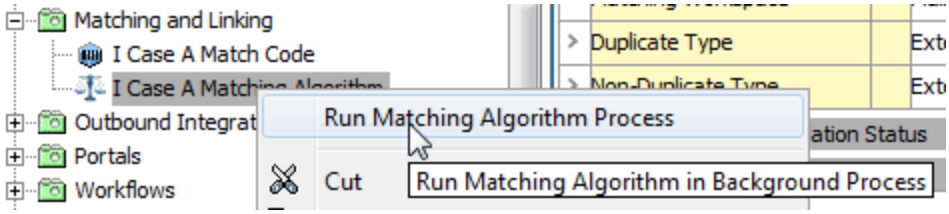
To do this, the matching component model must be configured first. Here the Source Object Type 'External Item' must be selected along with a Data Source Attribute (in our case 'Source') valid for External Items. Additionally, two different reference types must be configured and selected. These are used to indicate that objects are confirmed as duplicates / non-duplicates and must be valid from External Item to External Item. Finally, an attribute used to hold the justification for confirmations must be selected. This attribute must be valid for the reference types.

For more information on configuring the component model, see the **Component Model Configuration** section of the **Matching and Linking** documentation.



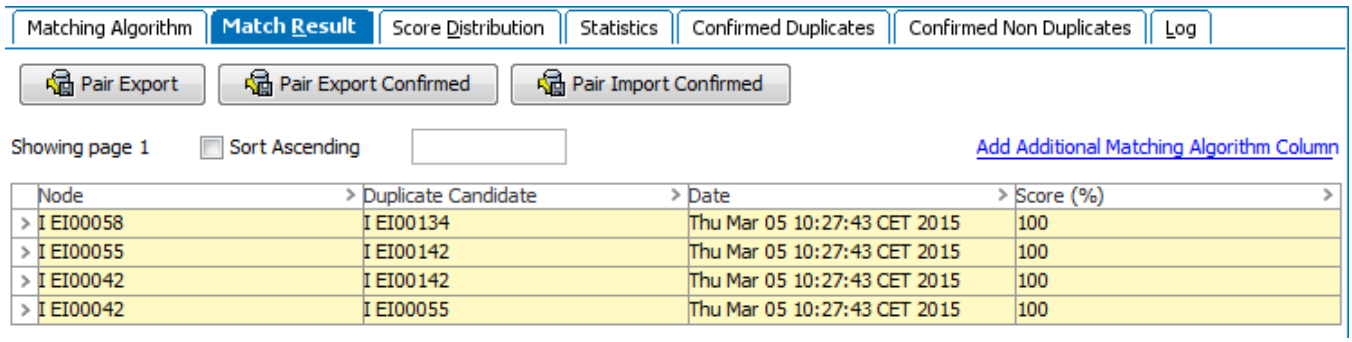
For more information on configuring matching algorithms, see the **Configuring Matching Algorithms Overview** section of the **Matching and Linking** documentation.

Once the matching algorithm has been configured, it can be applied via the object context menu as illustrated in the image below.



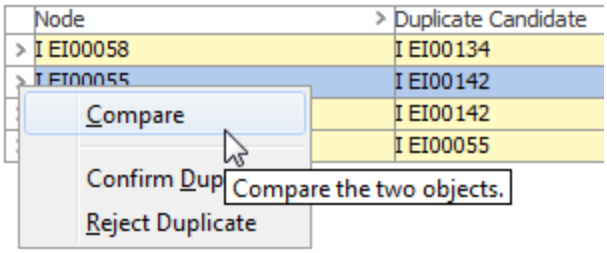
Handling Identified Duplicates

The matches that the matching algorithm finds can be inspected from the matching algorithm 'Match Result' tab.



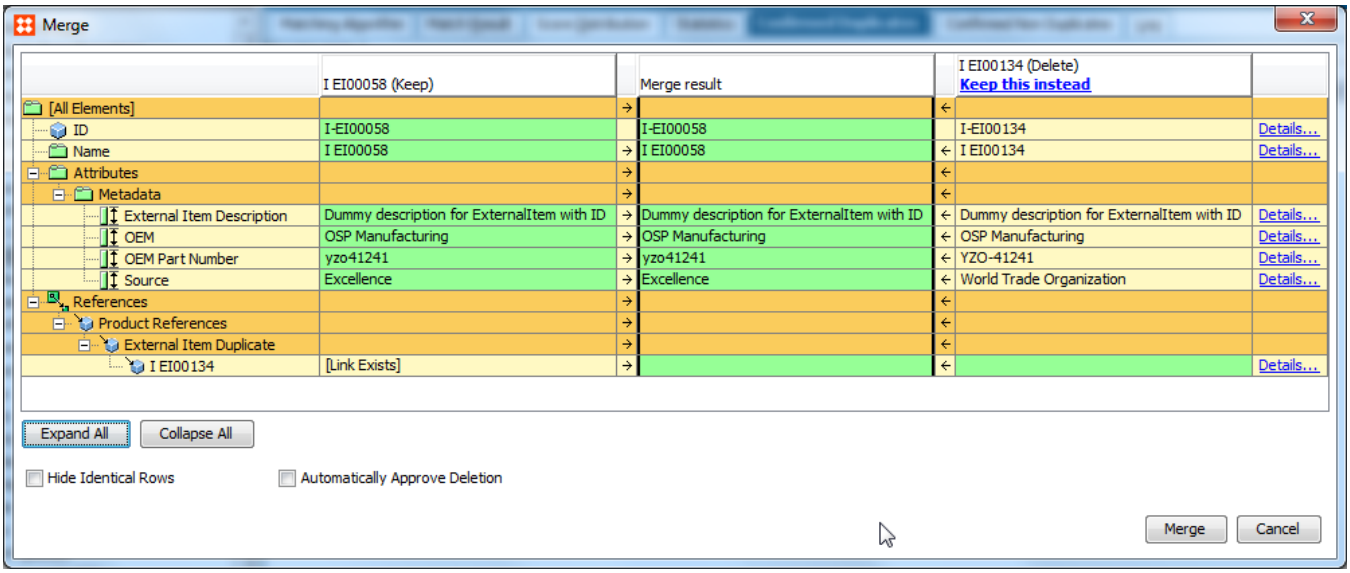
It is important to understand that with the 'Identify Duplicates' match action nothing has happened to the objects at this point. No references or new objects (golden records) have been created. On the 'Match Result' tab you simply see the pairs that the matching logic has identified as duplicates.

From this tab it is possible to compare pairs and mark them as either confirmed duplicates or confirmed non-duplicates.



It is important to understand that if a pair has been confirmed as duplicates / non duplicates, the pair will not be considered if the Matching Algorithm is re-applied regardless of whether the data on the objects has changed. The confirmed duplicate / non-duplicate relationship can be updated either via the 'Remove From List' option or by deleting the references.

From the 'Confirmed Duplicates' tab, apart from removing a pair, it is also possible to merge a pair into a single record. If this option is selected, then a dialog like the one shown below will open. Here you can decide which object to keep and manually merge data from the object you choose to delete and the one you wish to keep.



For more information on handling identified duplicates, see the **Handling Identified Duplicates** section of the **Matching and Linking** documentation.

Golden Record Configuration

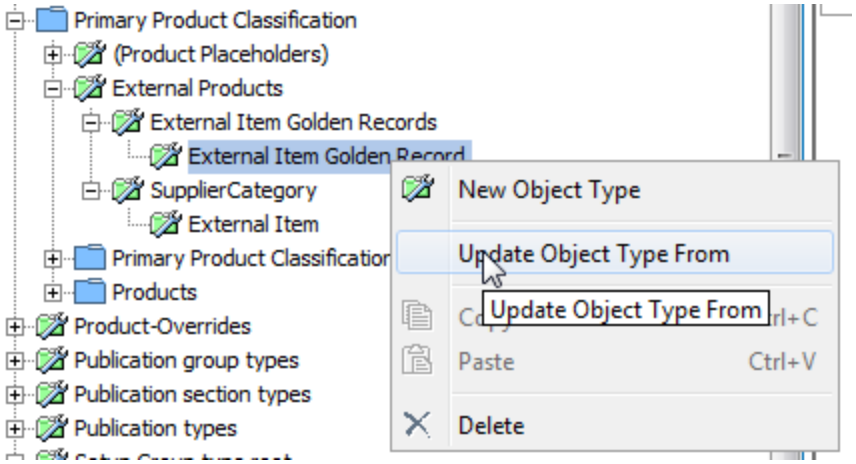
If instead of just identifying (and potentially merging) duplicates you want to have the matching functionality produce golden records, some extra configuration is required.

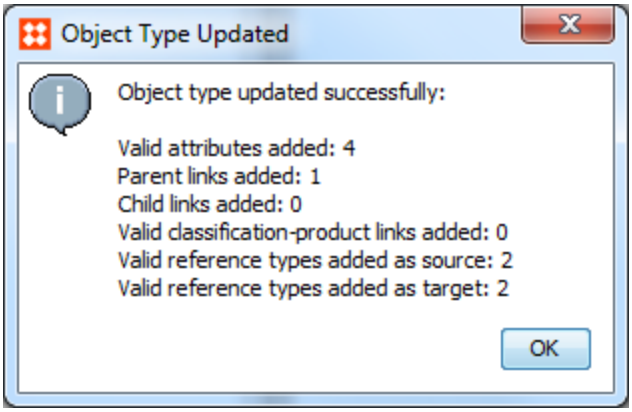
Golden records automatically created by the matching functionality should be of a different object type than the source objects, e.g., 'ExternalItemGoldenRecord'. The golden record object type must have an auto ID pattern configured.

Description

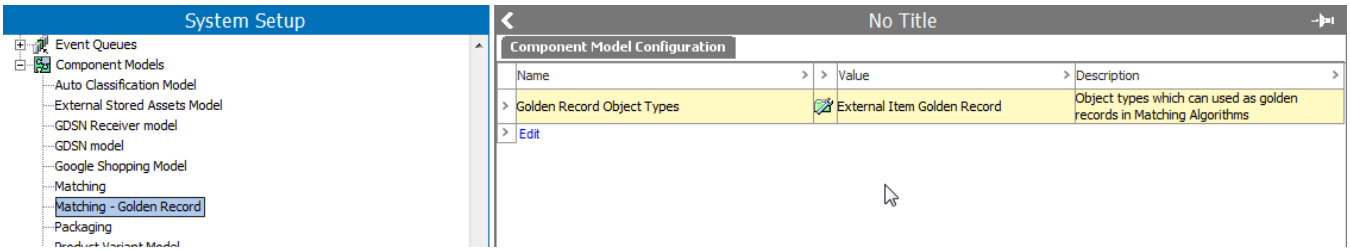
Name	Value
ID	ExternalItemGoldenRecord
Name	External Item Golden Record
Last edited by	2015-03-06 08:37:05 by STEPSYS
Name Pattern	
ID Pattern	ExternalItemGoldenRecord-[id]

Validity-wise, if you intend to copy all data (attribute values and references) from source objects, the golden record object type should have the same valid attributes and be a valid source for the same reference / link types. An easy way to secure this is via the 'Update Object Type From' context menu option as illustrated in the image below.

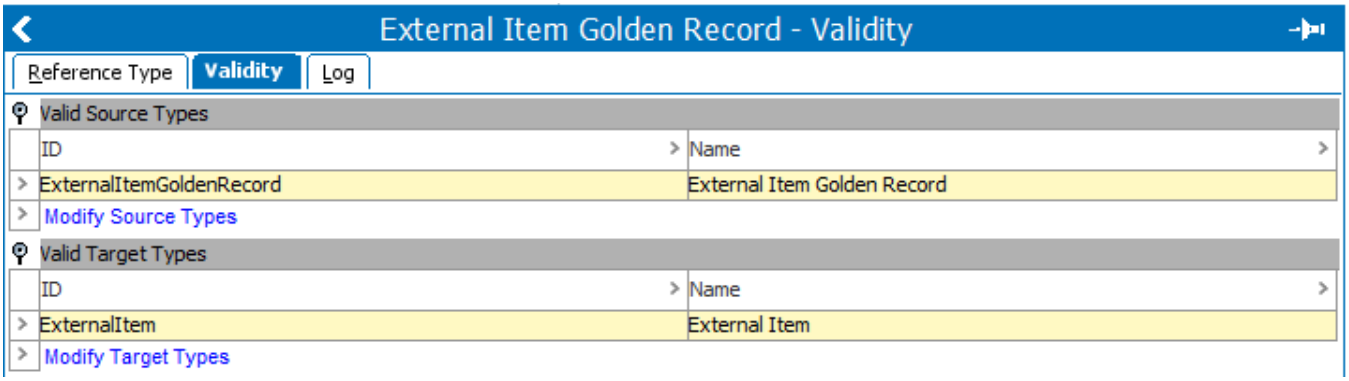




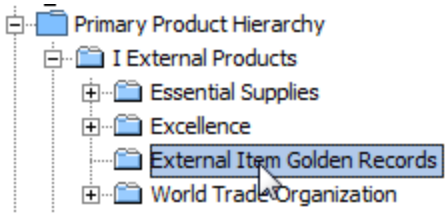
Once an object type for the golden records has been created, it must be selected in the 'Matching – Golden Record' component model as shown below.



In order for golden records to referenced back to their source objects a golden record reference type must be created. The reference type must allow for multiple targets and should be valid from the golden record object type to the source object type.



In addition to this, a root node for the golden records must be created. Initially, all golden records will be created as immediate children of this node.



The golden record object type, the reference type, and the golden records root node must be selected in the Golden Record Match Action on the matching algorithm. Here you will have to specify an "Auto Threshold" and a "Clerical Review Threshold".

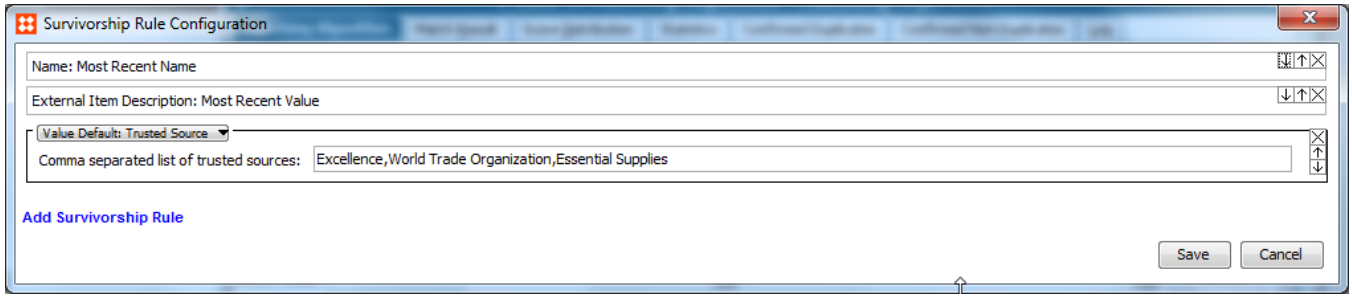
For more information about configuring golden records, see the **Configuring Golden Records** section of the **Matching and Linking** documentation.

Survivorship Rules Configuration

To copy data from source objects to golden records, survivorship rules must be configured. Two different approaches exist for this: 'Most Recent' and 'Trusted Source'. A mixture of these can be used and different rules can be configured for individual pieces of data.

The Trusted Source option takes a comma-separated list of sources as an argument. Notice here that it is preferable if the source attribute is LOV based so that you know exactly which values to expect.

In the image below, the survivorship rules have been configured, allowing data to be copied from the External Item Source Objects to the External Items Golden Records.



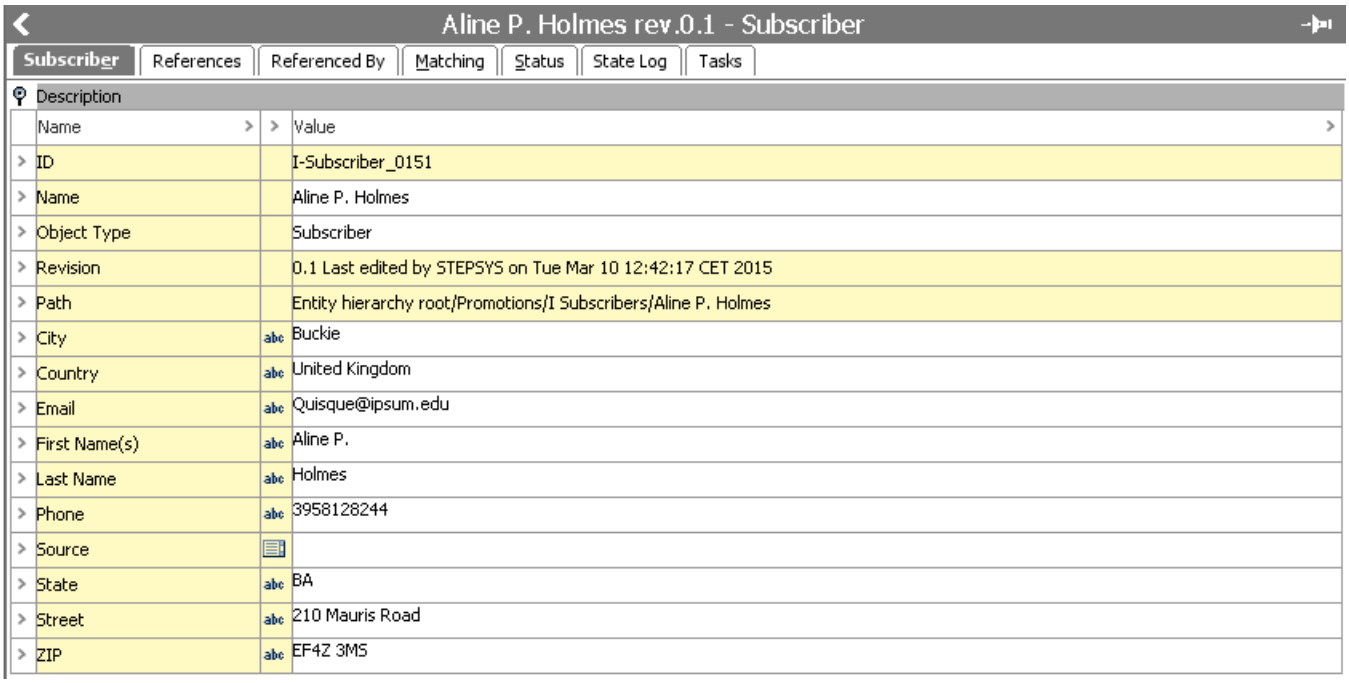
For more information on configuring survivorship rules, see the **Survivorship Rules** section of the **Matching and Linking** documentation.

Configuration Example - Advanced

This use case example focuses on more advanced matching logic that could be required when working with data that doesn't have any obvious identifiers.

The source objects for this case, for which duplicates are to be identified, are 200 'Subscriber' objects similar to the one shown below. Each object has the following significant attributes:

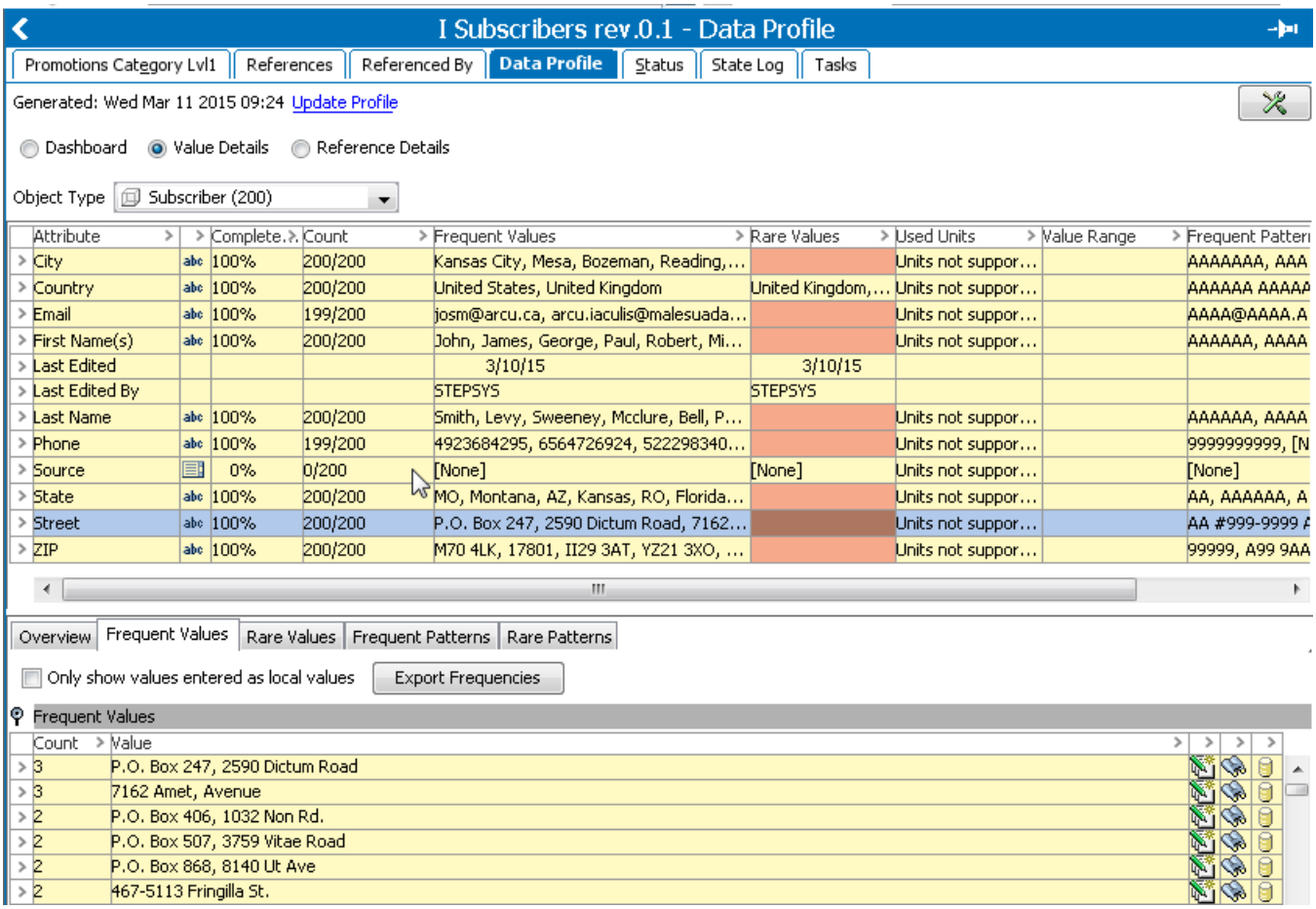
- City
- Country
- Email
- First Name(s)
- Last Name
- Phone
- State (Region)
- Street
- ZIP



Data Profile Analysis

When inspecting the data via the Data Profile functionality, the following can be observed:

- Data is almost complete with a few email addresses and phone numbers missing
- Country is always either 'United Kingdom' or 'United States'
- Street values are not standardized (e.g., different abbreviations are used)
- Phone numbers are standardized and match a uniform pattern (10 digits)



For more information on data profiles, see the **Data Profiles** section of the **Data Profiling** documentation.

There are a multitude of different cases to account for when deduplicating data like this. To give an idea, for this example, the following will be dealt with:

- Nicknames might be used ('Bob' and 'Robert' might be the same person)
- Middle names might be missing or abbreviated in 'First Name(s)' values
- Case (lower / upper) may differ in entered values
- Some names are more common than others
- Subscribers could have changed phone number and/or email address
- Subscribers could have moved
- Street values might have been entered in very different ways

Single field street values are notoriously hard to deal with, so for the strategy detailed here, these will only be used as a last resort. Instead, matching will be attempted based primarily on names, phone number, and email address, the logic being that if you have a match on these three pieces of information, there is a high probability that it is in fact the same person. Still, we also need to account for the cases where people have changed either email or phone number.

Library Functions

For the matching process, a number of JavaScript functions have been declared in a business library with ID 'MatchingFunctions'. These functions can be drawn upon for both pure JavaScript matching algorithms and JavaScript in decision tables. The functions are described below.

Important: It should be stressed that the below functions are simply examples and likely cannot be used in their current form for a real world case. Test thoroughly with your own data before implementing in your live STEP system.

---

normalizeValue

The normalizeValue function simply puts a text in lowercase and removes everything but letters and digits. It can be specified whether the function should only process and return the first word in the text.

```
function normalizeValue(value, handleFirstWordOnly) {
   if(value) {
      var normVal = value + "";
      if(handleFirstWordOnly) {
         normVal = normVal.split(" ")[0];
      }
      normVal = normVal.toLowerCase();
      normVal = normVal.replace(/[^\w]|_/g, "");
      return normVal;
   }
   else {
      return "";
   }
}
```

normalizeStreet

The normalizeStreet function applies basic normalization to 'Street' values and uses a transformation lookup table with ID 'AddressAbbreviations' to replace common abbreviations like 'rd', 'ave' and 'ap' with their full word counterpart.

The logic reads:

- Convert input to JavaScript string
- Convert to lowercase
- Remove all instances of (.), (,), and (#) (more characters should probably be removed, but be careful removing dashes if used in street number ranges)
- Split the string by space characters and loop through the array of words applying the lookup table
- Piece together the string again and return it

Lookup Table

Replace with default value when no matches are found (Value Substitution only):

Ignore Case

From	To
aly	alley
anx	annex
apt	apartment
arc	arcade
ave	avenue
bch	beach
bg	burg
bldg	building
blf	bluff
blvd	boulevard
bnd	bend
br	branch

```
function normalizeStreet(input, lookupTableHome) {
   var output = "";
   if(input) {
      input = input + "";
      input = input.toLowerCase();
      input = input.replace(/[\.\,#]|_/g, "");
      var inArr = input.split(" ");
      var outArr = [];
      for(var i = 0; i < inArr.length; i++) {
outArr.push(lookupTableHome.getLookupTableValue("AddressAbbreviations", inArr[i]));
      }
      for(var j = 0; j < outArr.length; j++) {
         output += outArr[j];
         if(j != outArr.length - 1) {
            output += " ";
         }
      }
   }
   return output;
}
```

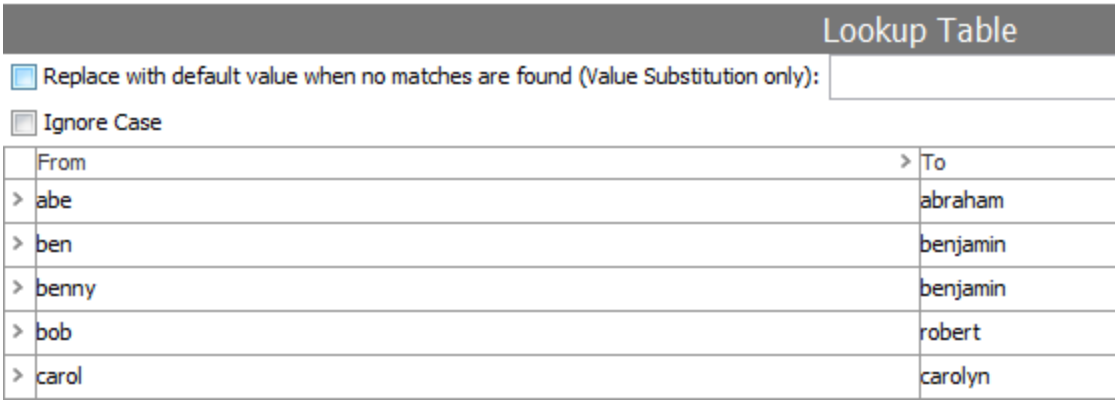
nameComparison

The purpose of the nameComparison function is to produce a value between 0 and 1 indicating how good a match a name is. Apart from 'manager' and 'lookupTableHome' that are passed as arguments (because you cannot create bindings from library functions), the function takes six arguments:

- firstName1 – Normalized value of the 'First Name(s)' attribute for the first of the two objects being compared (normalized via normalizeValue returning only first word)
- lastName1 – Normalized value of the 'Last Name' attribute for the first of the two objects being compared (normalized via normalizeValue)
- firstName2 – Normalized value of the 'First Name(s)' attribute for the second of the two objects being compared (normalized via normalizeValue returning only first word)
- lastName2 – Normalized value of the 'Last Name' attribute for the second of the two objects being compared (normalized via normalizeValue)
- commonNameFactor – A number between 0 and 1 indicating how hard common names like 'John Smith' should be punished. The lower the number, the harder the punishment. Typical range between 0.8 and 9.9.
- nicknameMatchFactor – A number between 0 and 1 indicating how hard nickname matches should be punished. The lower the number, the harder the punishment. Typical range between 0.8 and 1.
- lastNameWeightFactor – A number between 0 and 1 indicating how important a rare last name is compared to a rare first name

Basically, the function first produces a string for each object being compared. It consists of the normalized first name, a colon, and the normalized last name. If the two strings are identical, the function will call the function getFullNameWeight passing commonNameFactor and lastNameWeight as arguments. This function will return a value between 0 and 1 based on how common the name is and this value will also be the return value for nameComparison. Here 'John Smith' will produce a low value while a more uncommon name will produce a higher value. The functionality of getFullNameWeight is described further down.

If the two generated strings are not identical, nameComparison will check whether the last names are identical. If this is the case, a transformation lookup table with ID 'Nicknames' will be used to check if there is a match when common nicknames, like 'Bill', are replaced with full names like 'William'. If there is a match after the nickname replacement, the getFullNameWeight function is used again to produce a common name weight and this weight is multiplied with the nicknameMatchFactor and returned. It should be noted that in this simplified setup, cases like 'Ben', mapping to both 'Benjamin' and 'Benedict' are not handled in the nickname matching.


Lookup Table

Replace with default value when no matches are found (Value Substitution only):

Ignore Case

From	To
abe	abraham
ben	benjamin
benny	benjamin
bob	robert
carol	carolyn

If none of the above is true, a value of 0 indicating no match is returned. The complete function can be seen below.

```
function nameComparison(normFirstName1, normLastName1, normFirstName2, normLastName2,
manager, lookupTableHome, commonNameFactor, nicknameMatchFactor, lastNameWeightFactor) {
   var nameMatchValue = 0;
   var normName1 = null;
   if(normFirstName1 && normLastName1) {
      normName1 = normFirstName1 + ":" + normLastName1;
   }
   var normName2 = null;
   if(normFirstName2 && normLastName2) {
      normName2 = normFirstName2 + ":" + normLastName2;
   }
   if(normName1 && normName2) {
      if(normName1 == normName2) {
         nameMatchValue = getFullNameWeight(normFirstName1, normLastName1, manager,
lookupTableHome, commonNameFactor, lastNameWeightFactor);
      }
      else if(normLastName1 && normLastName2 && normLastName1 == normLastName2) {
         var lookup1 = lookupTableHome.getLookupTableValue("Nicknames", normFirstName1) +
"";
         var lookup2 = lookupTableHome.getLookupTableValue("Nicknames", normFirstName2) +
"";
         if(lookup1 == lookup2) {
            var fullNameWeight = getFullNameWeight(lookup1, normLastName1, manager,
lookupTableHome, commonNameFactor, lastNameWeightFactor);
nameMatchValue = fullNameWeight * nicknameMatchFactor;
         }
      }
   }
   return nameMatchValue;
}
```

getNameWeight and getFullNameWeight

In order to produce a metric for how common a name is, the getFullNameWeight function mentioned above and its helper function getNameWeight uses two transformation lookup tables: 'FirstNameFrequencies' and 'LastNameFrequencies' that contain frequency information for the most common first names and last names.

Transformation Lookup Table

Description

Name		Value
ID		FirstNameFrequencies
Name		First Name Frequencies
Object Type		Transformation Lookup Table
Revision		4.0 Last edited by STEPSYS on Fri Mar 13 12:02:12 CET 2015
Approved		Never Been Approved
Translation		Not Translated
Path		Classification 1 root/Configurations/Transformation Lookup Tables/First Name Frequencies
Name Frequency Max Value	123	3.271
Name Frequency Min Value	123	0.101

Lookup Table

Replace with default value when no matches are found (Value Substitution only): -1

Ignore Case

From	To
aaron	0.24
adam	0.259
alan	0.204

Transformation Lookup Table

Description

Name		Value
ID		LastNameFrequencies
Name		Last Name Frequencies
Object Type		Transformation Lookup Table
Revision		0.1 Last edited by STEPSYS on Tue Mar 10 21:52:59 CET 2015
Approved		Never Been Approved
Translation		Not Translated
Path		Classification 1 root/Configurations/Transformation Lookup Tables/Last Name Frequencies
Name Frequency Max Value	123	1.006
Name Frequency Min Value	123	0.1

Lookup Table

Replace with default value when no matches are found (Value Substitution only): -1

Ignore Case

From	To
adams	0.174
allen	0.199

Note: The frequencies are obtained from U.S. Census and are thus, strictly speaking, not representative for UK names. However, for this example they will suffice.

The getNameWeight function is called from getFullNameWeight and works either on a first name or a last name using the appropriate lookup table to produce a metric. In this simplified setup, the function simply produces a number between 1 and commonNameFactor (initially supplied to nameComparison as an argument) based on the frequencies in the lookup table. If a name is not on the list, it will get a value of 1 indicating that it is an uncommon name.

getFullNameWeight uses lastNameWeightFactor (also supplied to nameComparison as an argument) to produce a weighted average of the first name and last name weight. Both functions are shown below.

```
function getNameWeight(name, manager, lookupTableHome, isFirstName, minWeight) {
   var newMax = 1;
   var newMin = minWeight;
   var lookupTableID;
   if(isFirstName) {
      lookupTableID = "FirstNameFrequencies";
   }
   else {
      lookupTableID = "LastNameFrequencies";
   }
   var lookupValue = parseFloat(lookupTableHome.getLookupTableValue(lookupTableID, name));
   var returnValue;
   if(lookupValue == -1) {
      returnValue = newMax;
   }
   else {
      var origMax = parseFloat(manager.getAssetHome().getAssetByID(lookupTableID).getValue
("NameFrequencyMaxValue").getSimpleValue());
      var newRange = newMax - newMin;
      returnValue = ((-1 * (newRange / origMax)) * lookupValue) + 1;
   }
   return returnValue;
}


function getFullNameWeight(firstName, lastName, manager, lookupTableHome, minWeight,
lastNameWeightFactor) {
   var firstNameWeight = getNameWeight(firstName, manager, lookupTableHome, true,
minWeight);
   var lastNameWeight = getNameWeight(lastName, manager, lookupTableHome, false,
minWeight);
   return (firstNameWeight * (1 - lastNameWeightFactor)) + (lastNameWeight *
lastNameWeightFactor);
```

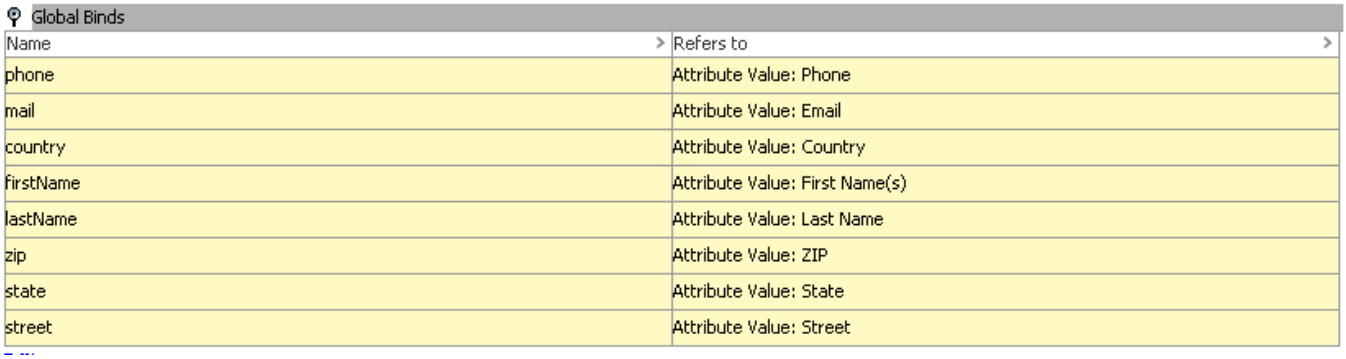
Matching Algorithm Configuration

In this example, the matching algorithm logic will be implemented via a decision table drawing upon the library functions described above.

For more information on configuring matching algorithms, see the **Configuring Matching Algorithms Overview** section of the **Matching and Linking** documentation.

Global Binds

For performance reasons, all attribute values used in the decision table comparison will be obtained via global binds. The configuration can be seen below.



Global Binds

Name	Refers to
phone	Attribute Value: Phone
mail	Attribute Value: Email
country	Attribute Value: Country
firstName	Attribute Value: First Name(s)
lastName	Attribute Value: Last Name
zip	Attribute Value: ZIP
state	Attribute Value: State
street	Attribute Value: Street

Transformer Expressions

When using decision tables, it is recommended to separate the different parts of the logic, making it easier to maintain and fine tune. In this example, transformer expressions are used for normalization, and thus, this part of the logic is separated from the comparison part. The transformers are described below:

- **normPhone Transformer Expression**

  This transformer uses the normalizeValue library function to normalize phone number values obtained via the 'phone' global bind ('Match Expression Context' is bound to 'mec').

```
return mf.normalizeValue(mec.evaluate("phone"), false);
```

- **normEmail Transformer Expression**

  For email, normalizeValue cannot be used, as punctuation should not be removed. Instead, values are put in lowercase.

```
var input = mec.evaluate("mail");
if(input) {
   return input.toLowerCase();
}
else {
   return "";
```

```
}
```

- **normFirstName Transformer Expression**

  Similar to normPhone, normFirstName uses the normalizeValue library function but normalizes and returns only the first word in the first name values.

```
return mf.normalizeValue(mec.evaluate("firstName"), true);
```

- **normLastName Transformer Expression**

  Similar to normPhone.

```
return mf.normalizeValue(mec.evaluate("lastName"), false);
```

- **normCountry Transformer Expression**

  Similar to normPhone.

```
return mf.normalizeValue(mec.evaluate("country"), false);
```

- **normZip Transformer Expression**

  Analog to normPhone.

```
return mf.normalizeValue(mec.evaluate("zip"), false);
```

- **normStreet Transformer Expression**

  For normStreet the normalizeStreet library function is used. 'Lookup Table Home' is bound to 'LookupTableHome' and passed as an argument along with the street value.

```
return mf.normalizeStreet(mec.evaluate("street"), lookupTableHome);
```

>	normPhone	Transformer	JavaScript Function: Bindings, return mf.normalizeValue(mec.evaluate("phone"));
>	normEmail	Transformer	JavaScript Function: Bindings, var input = mec.evaluate("mail");if(input) {return input.toLowerCase();}else {return "";}
>	normFirstName	Transformer	JavaScript Function: Bindings, return mf.normalizeValue(mec.evaluate("firstName"), true);
>	normLastName	Transformer	JavaScript Function: Bindings, return mf.normalizeValue(mec.evaluate("lastName"), false);
>	normCountry	Transformer	JavaScript Function: Bindings, return mf.normalizeValue(mec.evaluate("country"), false);
>	normZip	Transformer	JavaScript Function: Bindings, return mf.normalizeValue(mec.evaluate("zip"), false);
>	normStreet	Transformer	JavaScript Function: Bindings, return mf.normalizeStreet(mec.evaluate("street"), lookupTableHome);

Constants

To make it easier to fine tune the matching logic, all constants used in the algorithm are represented as constant expressions. The constants are described below.

- **commonNameFactor Constant Expression**

  Constant used for punishing common names. Most common names would get a value equal to or close to this number. Rare names will get a value of 1. Initially set to 0.9.

- **nicknameMatchFactor Constant Expression**

  Constant used for punishing nickname replacements. Initially set to 0.85.

- **nonMatchingPhoneOrEmailFactor Constant Expression**

  Constant used for punishing non-matching phone or emails. Initially set to 0.95.

- **nonMatchingPhoneAndEmailFactor Constant Expression**

  Constant used for punishing cases where neither phone or email match. Initially set to 0.8.

- **lastNameWeightFactor**

  Importance of last name compared to first name. Initially set to 0.6. In the getFullNameWeight library function, the constant will be used as follows: ([First Name Weight] * (1 – lastNameWeightFactor)) + ([Last Name Weight] * lastNameWeightFactor)

Comparator Expressions

- **phoneMatch Comparator Expression**

  The phoneMatch Expression works on normalized phone numbers and simply returns 1 (true) if there is a phone value for both objects being compared and they are identical. Otherwise 0 (false) is returned.

```
var phone1 = mec.evaluate("normPhone", "first");
var phone2 = mec.evaluate("normPhone", "second");
return (phone1 && phone2 && phone1 == phone2) ? 1 : 0;
```

- **emailMatch Comparator Expression**

The emailMatch Expression is identical to the phoneMatch Expression described above, but works on normalized email values instead.

```
var email1 = mec.evaluate("normEmail", "first");
var email2 = mec.evaluate("normEmail", "second");
return (email1 && email2 && email1 == email2) ? 1 : 0;
```

- **nameMatchValue Comparator Expression**

  The nameMatchValue Expression invokes the nameComparison library function and returns the result. Notice how the constants described above are referenced via the match expression context.

```
return mf.nameComparison(
   mec.evaluate("normFirstName", "first"),
   mec.evaluate("normLastName", "first"),
   mec.evaluate("normFirstName", "second"),
   mec.evaluate("normLastName", "second"),
   manager,
   lookupTableHome,
   parseFloat(mec.evaluate("commonNameFactor")),
   parseFloat(mec.evaluate("nicknameMatchFactor")),
   parseFloat(mec.evaluate("lastNameWeightFactor"))
```

```
);
```

- **countryZipMatch Comparator Expression**

  countryZipMatch works in the same way as phoneMatch and emailMatch, but concatenates on the country and zip values before comparison.

```
var country1 = mec.evaluate("normCountry", "first");
var country2 = mec.evaluate("normCountry", "second");
var zip1 = mec.evaluate("normZip", "first");
var zip2 = mec.evaluate("normZip", "second");
if(country1 && country2 && zip1 && zip2) {
   return (country1 + zip1) == (country2 + zip2) ? 1 : 0;
}
else {
   return 0;
}
```

- **streetEditDistance Comparator Expression**

  streetEditDistance uses the built-in levenshteinDistance function to get the edit distance between normalized street values. 'Matching Functions' have been bound to 'coreMatchingFunctions'.

```
var street1 = mec.evaluate("normStreet", "first");
var street2 = mec.evaluate("normStreet", "second");
return coreMatchingFunctions.levenshteinDistance(street1, street2);
```

Rules Setup

Based on the expressions described above, the rules shown below can be configured. Notice that this is just an example and that equally good or better rules could be configured. Also, notice how only STEP functions can be used for calculations in the Result column. Thus expression values are referenced via the STEP function mcevaluate("ExpressionID").

Rules

phoneMatch	emailMatch	nameMatchVa...	countryZipMatch	streetEditDistance	Result	Comment
= 1	= 1				99.9	
= 1					100 * mcevaluate("nameMatchValue") * mcevaluate("nonMatchingPhoneOrEmailFactor")	
	= 1				100 * mcevaluate("nameMatchValue") * mcevaluate("nonMatchingPhoneOrEmailFactor")	
		> 0	= 1	< 2	100 * mcevaluate("nameMatchValue") * mcevaluate("nonMatchingPhoneAndEmailFactor")	

Add Rule

The rules state the following:

- If there is a match on both phone and email, there is a very high probability that it is the same person, and '99.9' is returned regardless of how well the name and address values match.
- If there is a match on phone only, the nameMatchValue is multiplied with 100 and the nonMatchingPhoneOrEmailFactor value, and returned.

- If there is a match on email only, the nameMatchValue is multiplied with 100 and the nonMatchingPhoneOrEmailFactor value, and returned.
- If neither phone nor email match, but there is a match on name (nameMatchValue > 0) AND a match on country and zip AND the edit distance between street values is less than 2, nameMatchValue is multiplied with 100 and the nonMatchingPhoneAndEmailFactor value, and returned.

Match Code Configuration

Given the logic outlined above, you will need to make sure that subscriber objects get compared if they either have the same email, same phone number, or same name and approximate location. To this end, if data is complete for a subscriber object, three match codes will be generated:

- Prefix 'PHONE-' concatenated with the phone number
- Prefix 'MAIL-' concatenated with the email
- Prefix 'NAMEADDR-' concatenated with normalized first name, last name, country, and zip

Notice here that with a big data set, the last match code probably would not work as it would cause too many comparisons. For example, John Smiths with the same ZIP Code.

A JavaScript version of the match code formula is shown below. Based on the match codes, the matching can be run with a Window Size of 1. In the code below, it is assumed that 'Current Object' has been bound to 'node' and that a dependency to the 'Matching Functions' library has been declared, giving access to the library via the JavaScript variable 'mf'.

```
var normFirstName = mf.normalizeValue(node.getValue("S-FirstNames").getSimpleValue(),
true);
var normLastName = mf.normalizeValue(node.getValue("S-LastName").getSimpleValue(), false);
var normCountry = mf.normalizeValue(node.getValue("S-Country").getSimpleValue(), false);
var normZip = mf.normalizeValue(node.getValue("S-ZIP").getSimpleValue(), false);


var nameAddr = "";
if(normFirstName && normLastName && normCountry && normZip) {
   nameAddr = normFirstName + ":" + normLastName + ":" + normCountry + ":" + normZip;
}
var mail = node.getValue("S-Email").getSimpleValue();
var phone = node.getValue("S-Phone").getSimpleValue();


var mcArr = [];
if(nameAddr) mcArr.push("NAMEADDR-" + nameAddr);
if(mail) mcArr.push("MAIL-" + mail);
if(phone) mcArr.push("PHONE-" + phone);
```

```
if(mcArr.length > 0) return mcArr;
else return "";
```

For more information of configuring match codes, see the **Configuring Match Codes** section of the **Matching and Linking** documentation.

Generating Match Codes and Running a Matching Algorithm

In order to stay up-to-date with changes to data in STEP, you must maintain the matching components by periodically regenerating match codes and re-running matching algorithms.

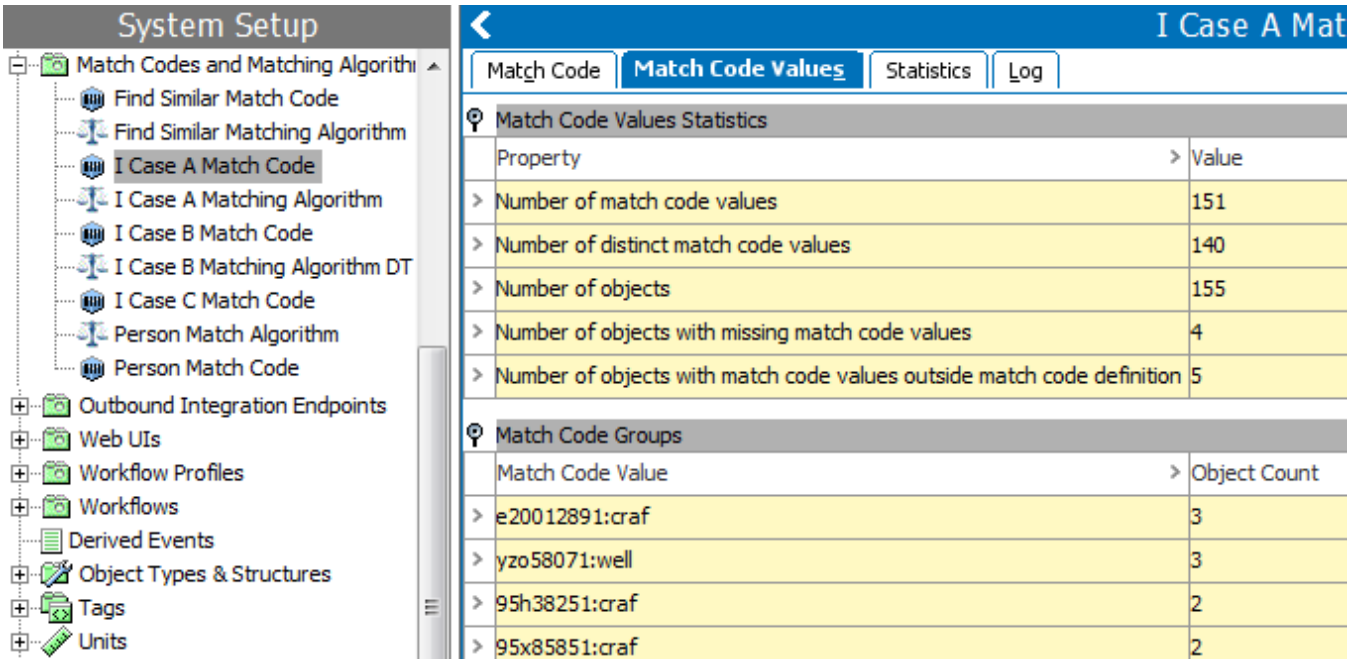
- Generating and updating match codes can be handled using an event processor, or can be performed manually (as described below).
- Running the matching algorithm requires an event processor and can be triggered automatically or initiated manually (as described below).

For information using an event processor to handle maintenance tasks automatically, see the **Configuring Matching Event Processor** section of the **Matching and Linking** documentation.

Match Code Values Statistics

Match code values are based on object type and optionally, a category (the parent node for the object in the hierarchy). Once a match code value exists, changing the object type or moving the object may cause the object to fall outside the match code value definition, making it invalid.

The Match Code editor includes a Match Code Values Statistics flipper that displays the number of existing, missing, and invalid match code values. This data allows you to determine the maintenance tasks required to update your match code values. The invalid match code values should be removed prior to running the matching algorithm. Ideally, all objects will have a match code value and no values will be outside the match code definitions.



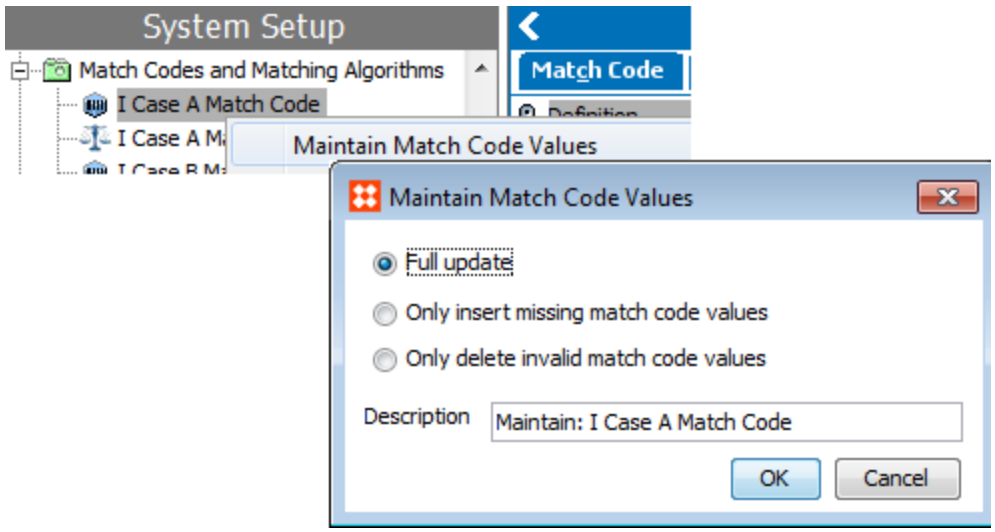
Note: The 'Maintain Match Code Values' functionality is controlled by System Setup > Action Sets > Setup Actions > Maintain Deduplication Configurations.

Maintain Match Code Values

When creating a new match code, the initial generation of match code values can be accomplished with the 'Maintain Match Code Values' option or by running a matching event processor with the Generate / Update Match Code Values option selected.

Important: The manual 'Maintain Match Code Values' process is required when the match code changes since this type of edit does not trigger events.

1. Verify a configured match code exists by following the steps defined in the **Configuring Match Codes** documentation.
2. Right-click on the match code node and select the 'Maintain Match Code Values' option.



3. In the 'Maintain Match Code Values' dialog, select the radio button for the desired action:
   - **Full update** updates existing match codes values, inserts missing match code values, and deletes invalid match code values. This default option is required for a new match code and creates a match code value for all objects that meet the match code criteria.
   - **Only insert missing match code values** is faster than a full update since it does not include updating or removing match code values. This option addresses the objects that are counted in the 'Number of objects with missing match code values' statistic. For example, after loading new objects, use this option to generate match codes for the new objects only, instead of regenerating match codes for all objects.
   - **Only delete invalid match code values** is faster than a full update since it does not include updating or adding match code values. This option addresses the objects that are counted in the 'Number of objects with match code values outside match code definition' statistic. For example, when the match code definition has been updated to exclude objects (because the object type has changed), use this option to only remove the match codes for the excluded objects, instead of regenerating match codes for all objects.

4. Click OK to generate / update match code values.

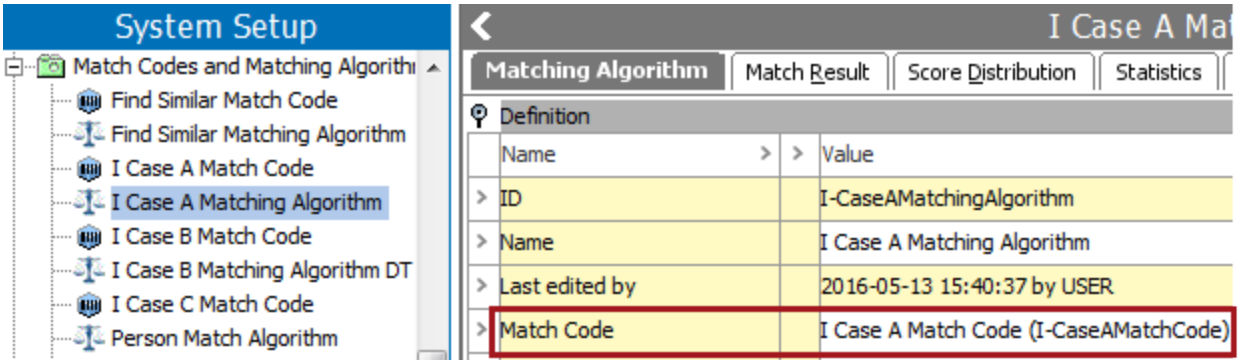
Run Matching Algorithm

Once match code values have been generated, you can run a matching algorithm, which requires an event processor. The 'Generate Events for Matching' option is a way of getting **all** objects in the matching algorithm definition into the queue to be matched. This option is useful when testing and fine-tuning your algorithm, as well as for running against all objects once an algorithm has been finalized.

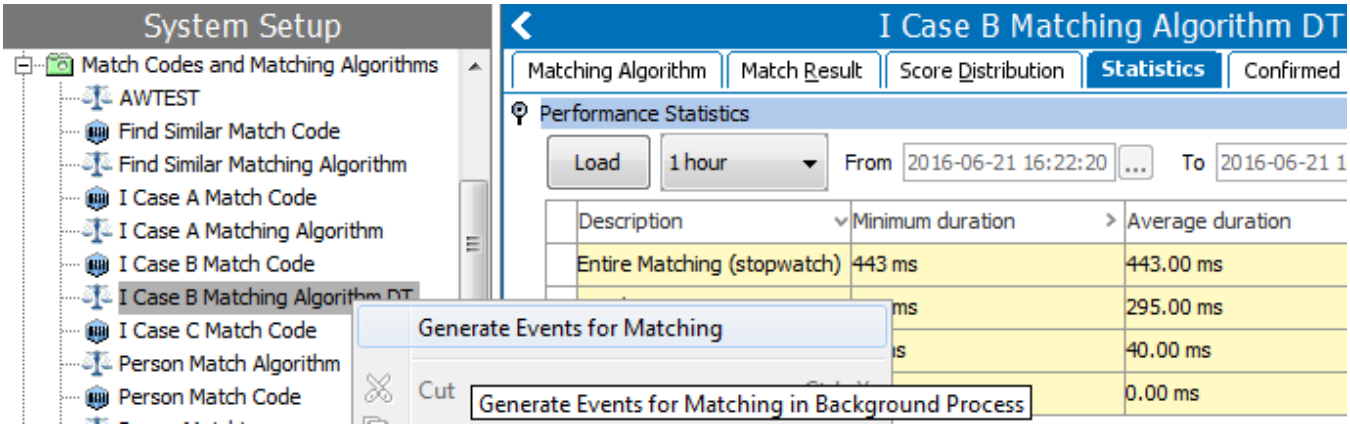
Important: The manual 'Generate Events for Matching' process is required when the matching algorithm definition changes since events are not triggered.

---

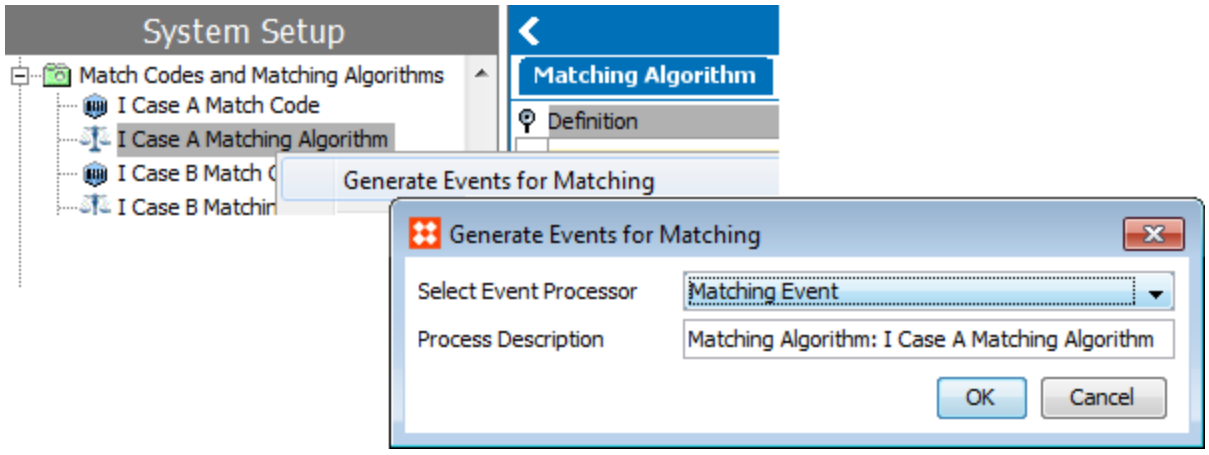
1. Create an event processor with the desired matching algorithm selected on the Configure Processing Plugin step. For detailed steps, see the **Configuring Matching Event Processor** section of the **Event Processor** documentation.
2. Select the desired matching algorithm node and determine the designated match code.



3. Review the designated match code and verify match code values are displayed on the Match Code Values tab (described at the beginning of this section). If needed, use the 'Maintain Match Code Values' functionality to update match code values and address any missing or invalid values.
4. Right-click the desired matching algorithm node and select 'Generate Events for Matching'.



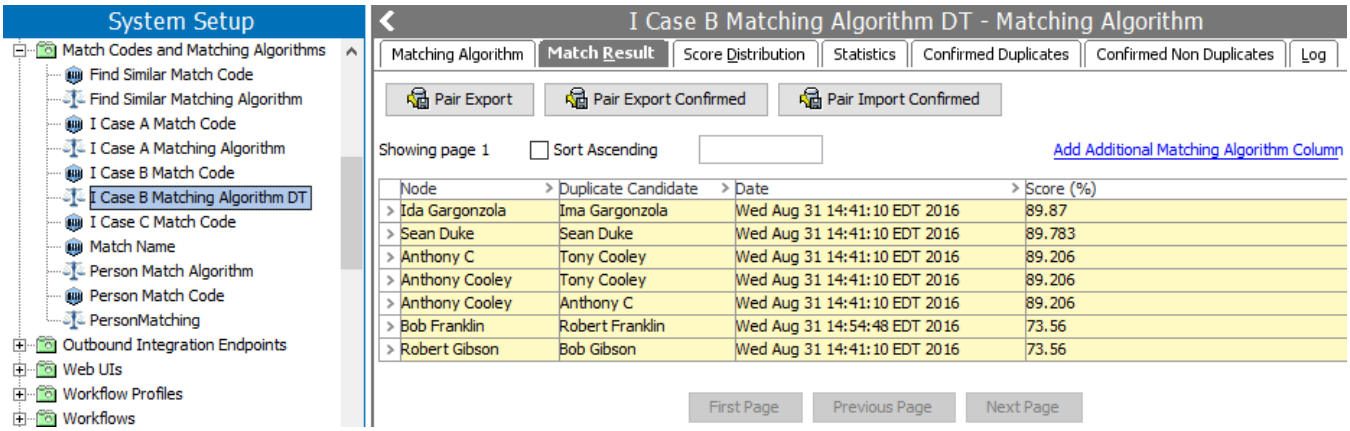
5. Select an existing event processor that is linked to the algorithm.



6. Click OK to start the background process which will rematch all objects using the matching algorithm.

Handling Potential Duplicates

Once the matching algorithm has been run, the results can be viewed on the 'Match Result' tab of the matching algorithm.

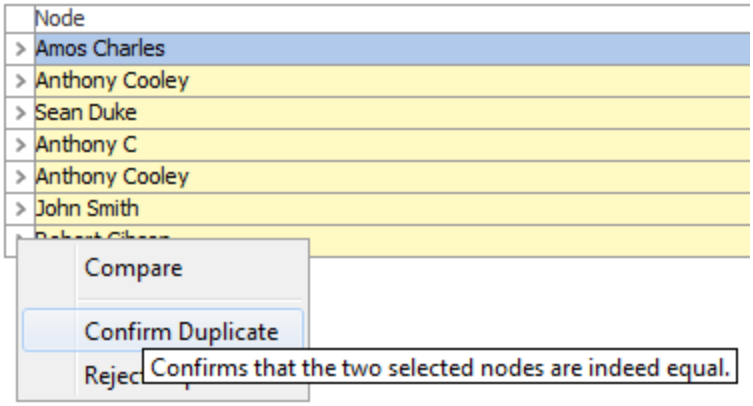


Note that potential duplicates identified via a golden record configuration can also be handled in a workflow. For more information, see the **Clerical Review** section of the **Matching and Linking** documentation.

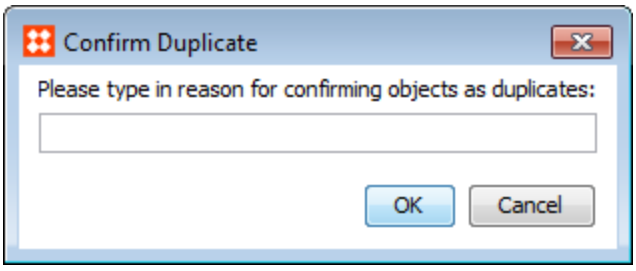
Confirm or Reject Duplicates

From the 'Match Result' tab it is possible to compare pairs and mark them as either confirmed duplicates or confirmed non-duplicates.

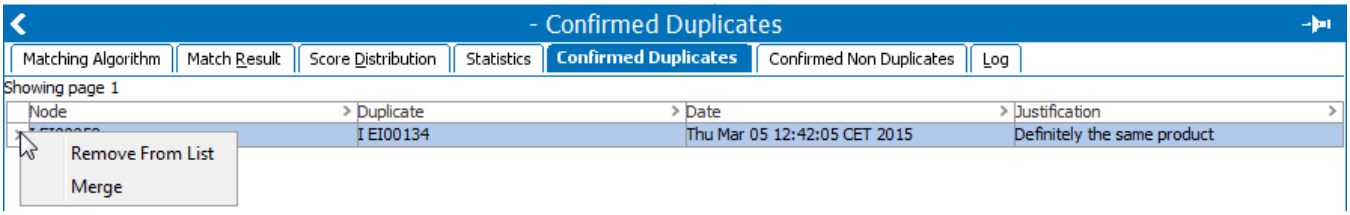
1. In **System Setup**, select the relevant matching algorithm, and then click the 'Match Result' tab.
2. Click the row that contains the duplicates you wish to confirm or reject.



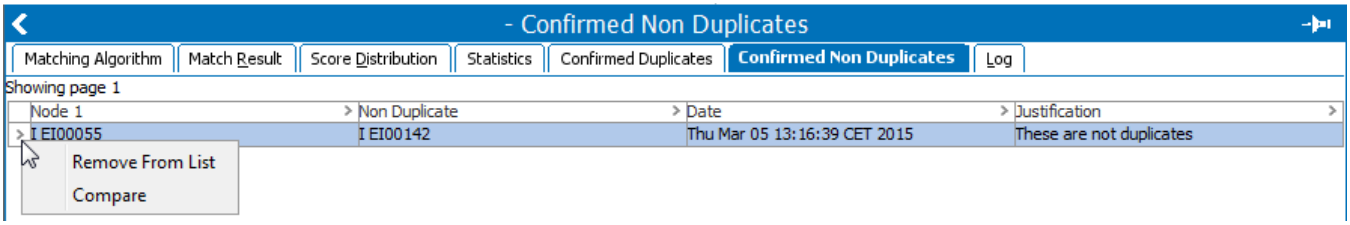
3. Provide a reason for the confirmation / rejection. Click **OK**.



If two objects are confirmed as being duplicates, a reference of the 'Duplicate Type' specified in the component model and in the matching algorithm will be created, the pair will be removed from the 'Match Result' tab, and instead, will show up on the 'Confirmed Duplicates' tab.



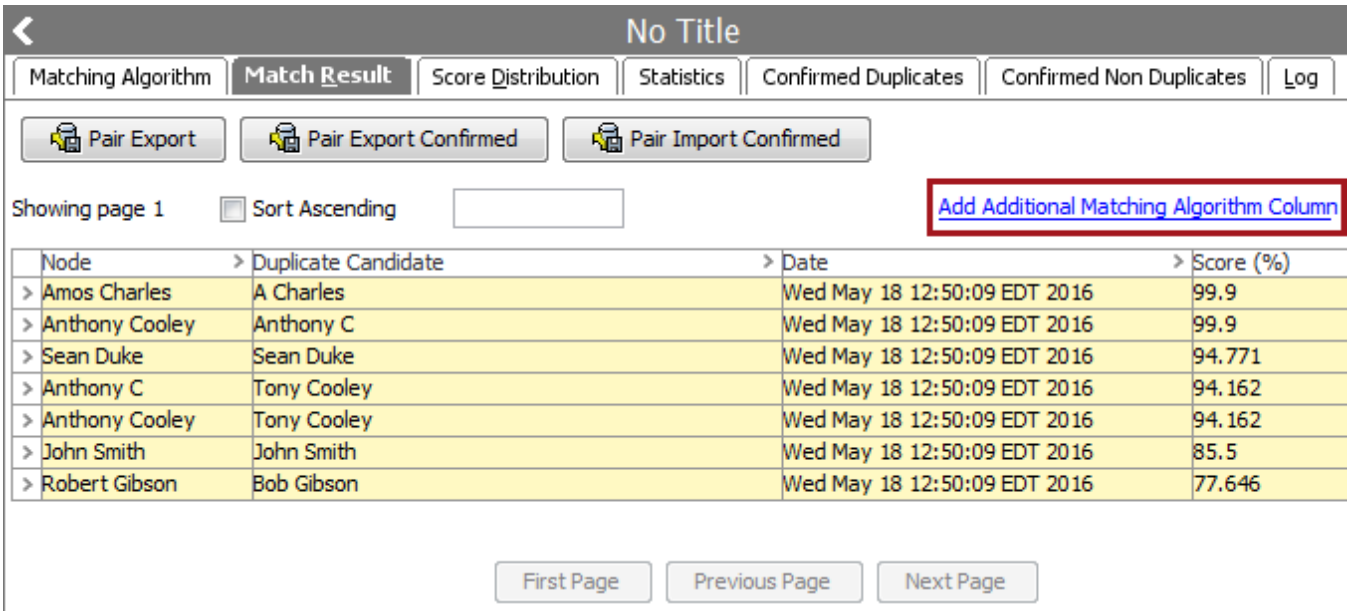
Likewise, if a pair is rejected as being duplicates, a reference of the 'Non Duplicate Type' will be created and the pair will be shown on the 'Confirmed Non Duplicates' tab.



It is important to understand that if a pair has been confirmed as duplicate / non duplicate, the pair will not be considered if the matching algorithm is reapplied, regardless of whether the data on the objects has changed. The confirmed duplicate / non-duplicate relationship can be updated either via the 'Remove From List' options shown above or by deleting the references.

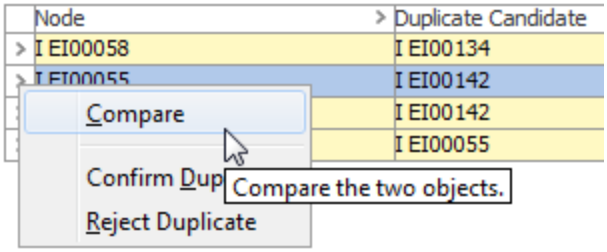
Add Additional Matching Algorithm

If you need more information about the objects before you decide whether they are duplicates or not, add an additional matching algorithm or compare the objects (described below). An additional algorithm can be added via a link on the 'Match Result' tab.

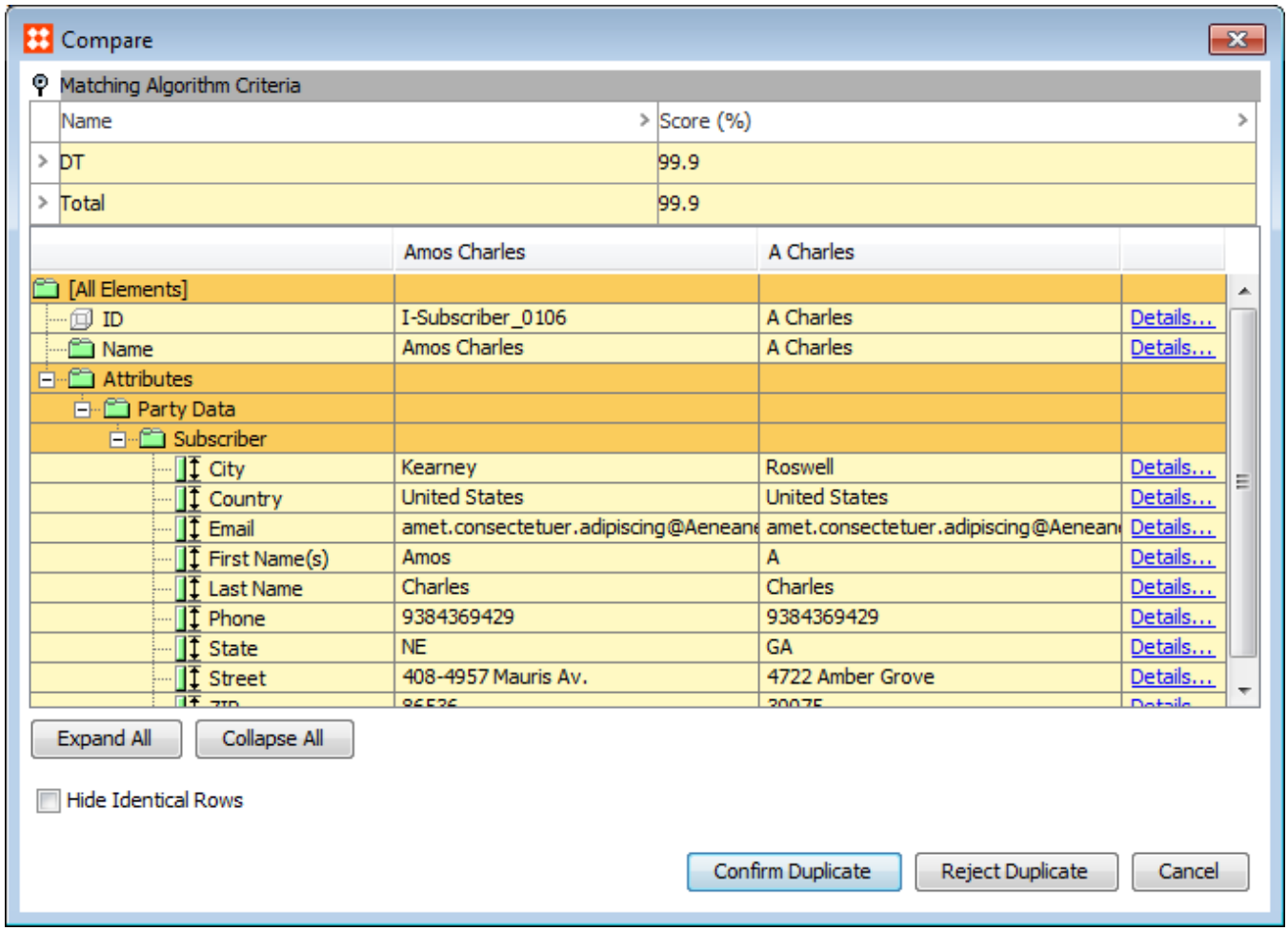


Compare Matched Objects

To compare a pair of objects, right-click on the applicable row in the 'Match Result' or 'Confirmed Non Duplicates' tab and select the 'Compare' option.



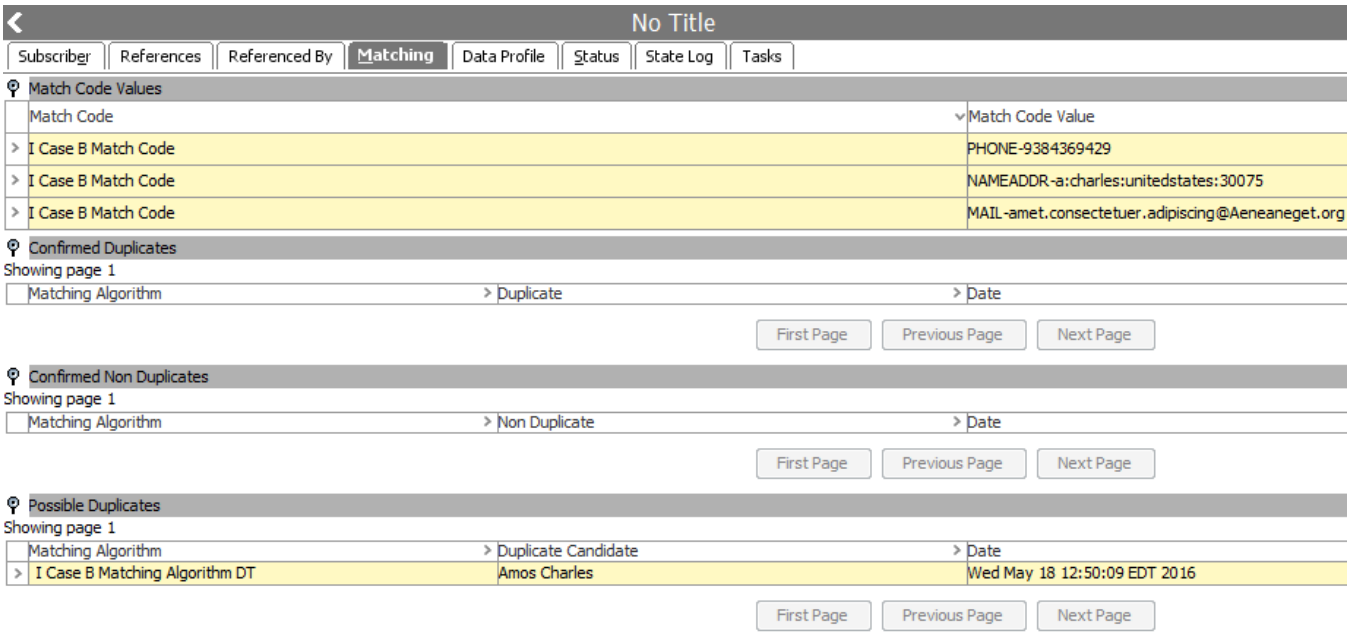
The 'Compare' screen can be used to review the similarities and differences between the paired objects. When accessed via the 'Match Result' you can confirm or reject duplicates via the **Confirm Duplicate** and **Reject Duplicate** buttons.



View Matched Objects in Tree

Duplicate information can also be viewed on objects in the Tree.

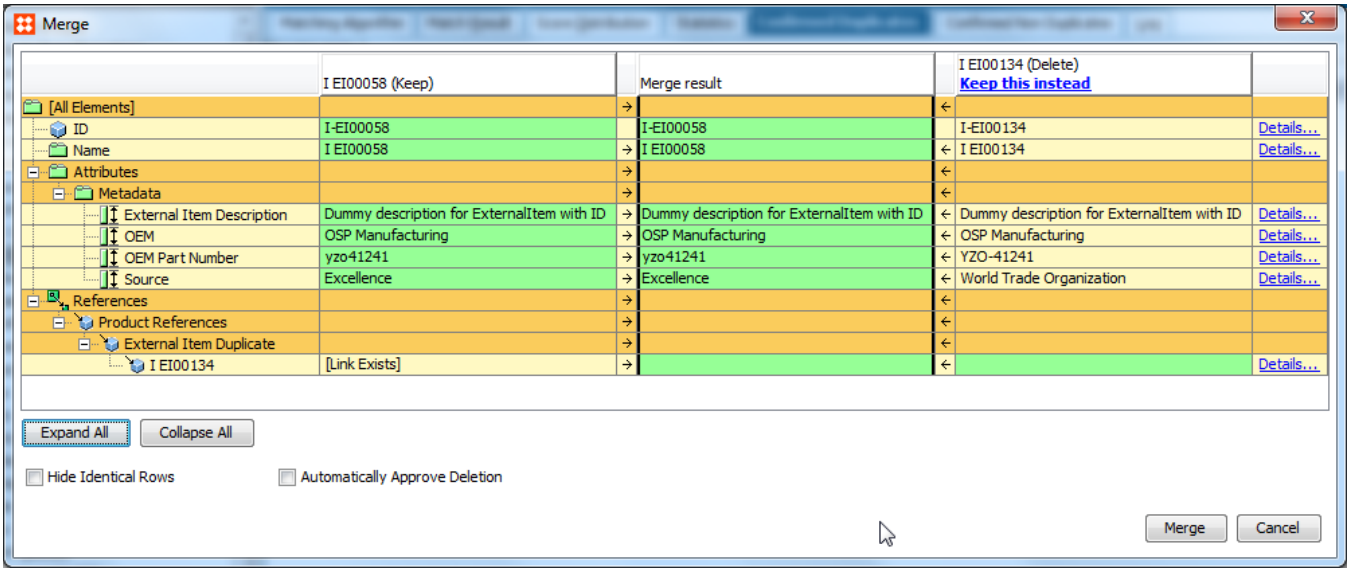
In **Tree**, select the relevant object, and then click the 'Matching' tab. Alternatively, you can access this information by clicking the link of the object if it is listed in the **Match Result** tab.



Merging Confirmed Duplicates

From the 'Confirmed Duplicates' tab, apart from removing a pair, it is also possible to merge a pair into a single record. If this option is selected, then a dialog like the one shown below will open. Here you can decide which object to keep, and manually merge data from the object you choose to delete and the one you wish to keep.

Important: Because duplicate source records are deleted during a merge this should not be used as part of a Golden Record solution.



If the object that remains does not contain any data in any context, the data is taken from the deleted object and merged into the remaining object.

Data here is defined as:

- Attributes
- Object name
- Reference types
- Object to classification link types
- Table types
- Object to attribute links

There is no accumulation for reference and link types. If the reference or link type is already populated in any context nothing is merged from the object that is deleted.

The five columns show the data type, the object to be kept, the result of the merge, the object to be deleted, and a details link used to inspect differences between the data on the objects.

- To keep the object that is currently set to be deleted, and instead delete the other object, click **Keep this instead** in the header of the 'Delete' column.
- Select 'Hide Identical Rows' to toggle between hiding and showing rows where the duplicate pairs contain the same data.
- Select 'Automatically Approve Deletion' to automatically approve that the deletion of objects in the 'Delete' column during the merge process.

The green cell background color indicates where data is taken from.

During the merge process, all references to the deleted object are modified to point to the object that remains in the database. This means that the source objects will be modified. If you select 'Automatically Approve Deletion', only the deletion of the objects is approved. Changes to objects because of references that are pointed to another target are not approved.

For more information about how to merge confirmed matches via Web UI, see the **Merging Confirmed Matches** section of the **Web UI** documentation.