Oracle® Healthcare Master Person Index

Message Processing Reference

Release 2.0.13

E25319-05

August 2016

ORACLE®

Oracle Healthcare Master Person Index Message Processing Reference, Release 2.0.13

E25319-05

Contents

3 The Database Structure

4 Working with the Java API

7 Web Services

Preface

Oracle Healthcare Master Person Index (OPHMPI) provides a flexible framework that allows you to design and create custom single-view applications, or master person indexes, which cleanse, match, and cross-reference healthcare objects across an enterprise. This reference provides background information on the message processes, the database structure, Java API classes and methods, and web services used by OHMPI.

Audience

This document is intended for users of OHMPI that want to learn about the logic behind message processing, the database structure, and use the Java API classes and methods to create programs.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

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

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

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

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

Finding Information and Patches on My Oracle Support

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

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

Creating a My Oracle Support Account

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

To register for My Oracle Support:

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

Signing In to My Oracle Support

To sign in to My Oracle Support:

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

Finding Information on My Oracle Support

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

Searching by Article ID

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

To search by article ID:

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

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

Searching by Product and Topic

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

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

Finding Patches on My Oracle Support

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

To locate and download a patch:

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

Finding Oracle Documentation

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

Finding Oracle Health Sciences Documentation

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

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

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

Finding Other Oracle Documentation

To get user documentation for other Oracle products:

1. Go to the following Web page:

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

   Alternatively, you can go to http://www.oracle.com, point to the Support tab, and then click **Documentation**.
2. Scroll to the product you need and click the link.
3. Click the link for the documentation you need.

Conventions

The following text conventions are used in this document:

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

1

Introduction to Oracle Healthcare Master Person Index

This chapter introduces you to the Oracle Healthcare Master Person Index (OHMPI) and lists some of its features.

This chapter includes the following section:

Introducing Oracle Healthcare Master Person Index

Oracle Healthcare Master Person Index provides a flexible framework that allows you to create matching and indexing applications called **enterprise-wide master person index applications**. It is an application building tool to help you design, configure, and create an OHMPI application that will uniquely identify and cross-reference the healthcare objects stored in your system databases. Business objects can be any type of entity for which you store information, such as patients, doctors, hospitals, medications, and so on.

When you create an OHMPI application, custom database scripts, and a custom Java API are automatically generated based on the information you specify in the wizard and the configuration files. Both the database scripts and API are derived from the object structure you define. For example, if you create an OHMPI application with an Patient object, the database scripts will define a table named SBYN_PATIENT and one named SBYN_PATIENTSBR. The Java API will include a class named PatientObject that includes "get" methods for each field you defined for the Patient object.

Oracle Healthcare Master Person Index provides features and functions that allow you to create and configure an OHMPI application for any type of data. The primary function of Oracle Healthcare Master Person Index is to automate the creation of a highly configurable OHMPI application. A wizard guides you through the initial setup steps, and the Configuration Editor allows you to further customize the configuration of the OHMPI application. The components you need to implement an OHMPI application are automatically generated.

OHMPI provides the following features:

- **Rapid Development** - Rapid and intuitive development of an OHMPI application using a wizard to create the OHMPI configuration and using XML documents to configure the attributes of the index. Templates are provided for quick development of patient and medical provider object structures.
- **Automated Component Generation** - Oracle Healthcare Master Person Index automatically creates the configuration files that define the primary attributes of the OHMPI application, including the configuration of the Master Index Data

Manager (MIDM). OHMPI also generates scripts that create the appropriate database schemas and an XSD based on the object structure you create and configure.

- **Configurable Survivor Calculator** - Oracle Healthcare Master Person Index provides predefined strategies for determining which field values to populate in the single best record (SBR). You can define different survivor rules for each field, and you can create a custom survivor strategy to implement in the OHMPI application.
- **Flexible Architecture** - Oracle Healthcare Master Person Index provides a flexible platform that allows you to create an OHMPI application for any business object. You can customize the object structure so the OHMPI application can match and store any type of data, allowing you to design an application that specifically meets your data processing needs.
- **Configurable Matching Algorithm** - Oracle Healthcare Master Person Index provides standard support for the OHMPI Match Engine. In addition, you can plug in a custom matching algorithm to the OHMPI application.
- **Custom Java API** - Oracle Healthcare Master Person Index generates a Java API that is customized to the object structure you define.
- **Standard Reports** - Oracle Healthcare Master Person Index provides a set of standard reports with each OHMPI application that can be run from a command line or from the MIDM. The reports help you monitor the state of the data stored in the OHMPI application and help you identify configuration changes that might be required.

OHMPI also provides pre-configured templates that allow you to quickly create OHMPI application for a patient solution or a provider index solution. See the following guides for more information:

- *Oracle Healthcare Master Person Index Australia Patient Solution User's Guide*
- *Oracle Healthcare Master Person Index United Kingdom Patient Solution User's Guide*
- *Oracle Healthcare Master Person Index United States Patient Solution User's Guide*
- *Oracle Healthcare Master Person Index Provider Index User's Guide*

2

Understanding Operational Processes

Master person index applications created by Oracle Healthcare Master Person Index (OHMPI) use a custom Java API library to transform and route data into and out of the OHMPI database. In order to customize the way the Java methods transform the data, it is helpful to understand the logic of the primary processing functions and how messages are typically processed through the Oracle Healthcare Master Person Index system.

This chapter describes and illustrates the processing flow of messages to and from the OHMPI application, providing background information to help design and create custom processing rules for your implementation.

This chapter includes the following section:

- "Learning About Message Processing"

  To provide a complete overview, message processing is divided into the following sections:

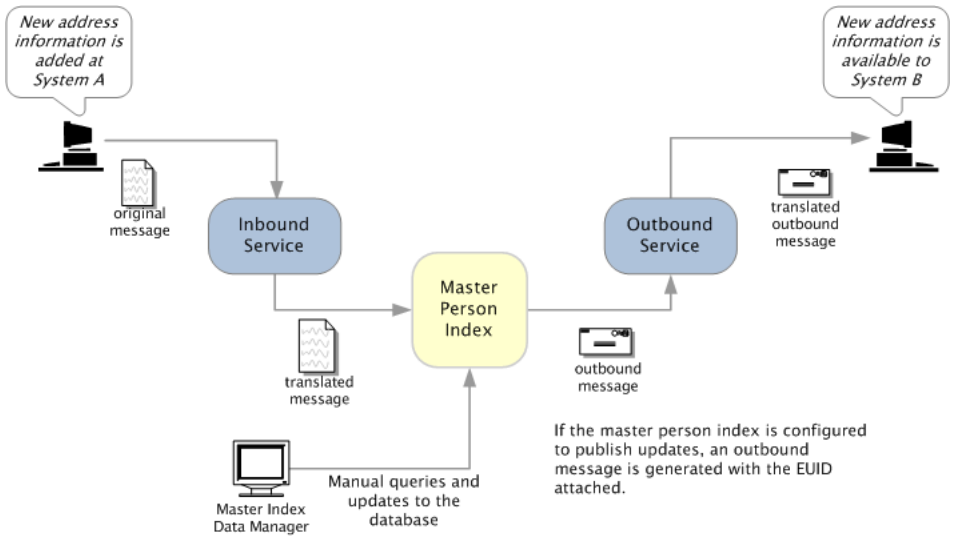
  - "Inbound Message Processing"
  - "Outbound Message Processing"
  - "Inbound Message Processing Logic"
  - "Primary Function Processing Logic"

Learning About Message Processing

This section provides a summary of how inbound and outbound messages can be processed in an OHMPI application. An OHMPI application cross-references records stored in various computer systems of an organization and identifies records that might represent or do represent the same object. The OHMPI application uses web services to connect to and share data with these external systems.

Figure 2–1 illustrates the flow of information through an OHMPI application that includes a JMS Topic to which updates to the index are published.

Figure 2–1 Oracle Healthcare Master Person Index Processing Flow

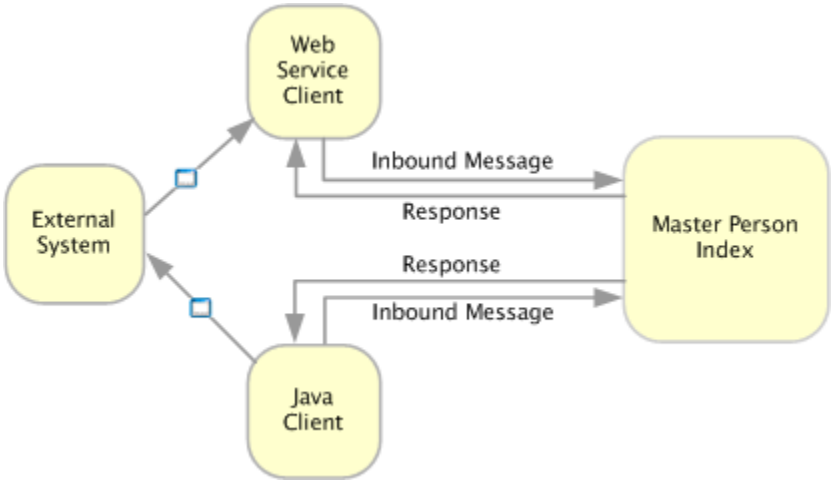


Inbound Message Processing

An inbound message refers to the transmission of data from external systems to the OHMPI database. These messages can be sent into the database via web services. The steps below describe how inbound messages are processed (see Figure 2–2 for an illustration of inbound messages to Master Person Index).

1. The message is transformed into the appropriate format for the OHMPI, and validations are performed against the data elements of the message to ensure accurate delivery. The message is typically validated using information stored in the OHMPI configuration files and the external business process logic.
2. After the OHMPI application processes the message, an enterprise-wide universal identifier (EUID) is returned (for a new or updated record). Alternatively, the entire updated message can be published using the generated outbound message (see "Outbound Message Processing").
3. If the message was successfully transmitted to the database, the appropriate changes to the database are processed.

Figure 2–2 Inbound Message Processing Data Flow



About Inbound Messages

The format for an inbound message is defined by the object structure of the Oracle Healthcare Master Person Index, located in the `object.xml` configuration file. The inbound messages can either conform to the required format for the OHMPI application, or they can be mapped to the required format in an external business process.

In addition to the objects and fields defined in `object.xml`, you can find the web service information in the WSDL file.

Outbound Message Processing

An outbound message refers to the transmission of data from the OHMPI database to any external system. Messages can be transmitted from the OHMPI application in two ways. The first way is by transmitting the output of `executeMatch` (an EUID). This is described in "Inbound Message Processing" and is only used for messages received from external systems.

The second way is by publishing updates from the OHMPI application to a JMS Topic, which allows you to publish complete, updated single best records (SBRs) to any system subscribing to that topic. When updates are made to the database from either external systems or the Master Index Data Manager (MIDM), the OHMPI application generates outbound messages in the format of the outbound XSD.

> **Note:** An OHMPI application only publishes the outbound message to JMS Topics and not to JMS Queues.

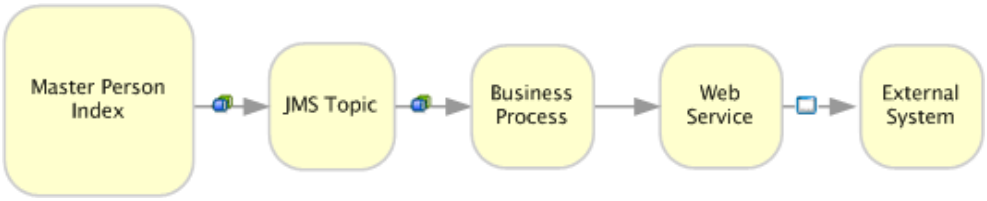
The following describes how the second type of outbound message is processed. A JMS Topic must be defined in the application server for this type of processing to occur.

1. When a message is received from an external system or data is entered through the MIDM, the OHMPI application processes the information and after internal operations are completed, OHMPI generates an XML message, which is sent to the JMS Topic that is configured to publish messages from the OHMPI application.

2. Messages published by the JMS Topic are processed through a business process (or other client application), which uses the OHMPI outbound XSD. The external business process can take the OHMPI XSD and transform the message into the appropriate format.
3. The message is routed using the appropriate binding component and sent to the appropriate external systems.

Figure 2–3 illustrates the flow of data for a message outbound from an OHMPI application.

Figure 2–3 Outbound Message Processing Data Flow



About Outbound Messages

This outbound message is used to publish changes in the OHMPI database to external systems via a JMS Topic. The output of the `executeMatch` process described earlier is an EUID of the new or updated record. You can use this EUID to obtain additional information and configure a business process to output the data, or you can process all updates in the OHMPI application through a JMS Topic using the outbound message. The outbound message is in XML format. When you customize the OHMPI object structure and generate the OHMPI application, an outbound schema file named `outbound.xsd` is created based on the object structure.

Outbound XSD Structure

The outbound XSD is located in the file structure for the OHMPI application under the *files-generated* folder. The XSD includes transaction information along with the updated record from the OHMPI database, and includes the following primary elements: OutMsg, SBR, SystemObject, the parent object, and any child objects. The OutMsg element defines the Event and ID. The Event field is populated with the type of transaction that created the outbound message, and the ID field is populated with the unique identification code of that transaction. The SBR element is created from `object.xml`. Table 2–1 describes the components of the SBR portion of the outbound OTD.

Table 2–1 Outbound XSD SBR Attributes

Node	Description
EUID	The EUID of the record that was inserted or modified.
Status	The status of the record.
CreateFunction	The date the record was first created.
CreateUser	The logon ID of the user who created the record.
UpdateSystem	The processing code of the external system from which the updates to an existing record originated.
ChildType	The name of the parent object.
CreateSystem	The processing code of the external system from which the record originated.

Table 2–1 (Cont.) Outbound XSD SBR Attributes

Node	Description
UpdateDateTime	The date and time the record was last updated.
CreateDateTime	The date and time the record was created.
UpdateFunction	The type of function that caused the record to be modified.
RevisionNumber	The revision number of the record.
UpdateUser	The logon ID of the user who last updated the record.
SystemObject	The object's local identifier in a specified system. This field has three sub-fields: ■ **LID**: The local ID assigned to the person in the system of origin. ■ **System**: The processing code of the system of origin. ■ **Status**: The status of the local ID in the enterprise record.
Object_Name	The fields in this node are defined by the object structure (as defined in `object.xml`). It is named by the parent object and contains all fields and child objects defined in the structure. This section varies depending on the customizations made to the object structure.

Outbound Message Trigger Events

When outbound messaging is enabled, the following transactions automatically generate an outbound message that is sent to the JMS Topic (if a JMS Topic has been incorporated into a Master Person Index or IHE Profile project).

- Activating a system record
- Activating an enterprise record
- Adding a system record
- Creating an enterprise record
- Deactivating a system record
- Deactivating an enterprise record
- Merging an enterprise record
- Merging a system record
- Transferring a system record
- Undoing assumed match
- Unmerging an enterprise record
- Unmerging a system record
- Updating an enterprise record
- Updating a system record

Sample Outbound Message

The following text is a sample outbound message for an OHMPI application based on an Oracle Healthcare Master Person Index. Your outbound messages will appear differently depending on how you configure the client project connectivity components.

```
<?xml version="1.0" encoding="UTF-8"?>
<OutMsg Event="UPD" ID="00000000000000004010">
  <SBR EUID="0000006501"  Status="active" CreateFunction="Add"
ChildType="Customer"
   CreateSystem="System" UpdateFunction="Update" RevisionNumber="3"
CreateUser="mdm"
   UpdateSystem="System" UpdateDate="" CreateDate="" UpdateUser="mdm">
    <SystemObject SystemCode="ORACLE" LID="2372385720"
Status="active"></SystemObject>
    <SystemObject SystemCode="ORACLE" LID="2837482562"
Status="active"></SystemObject>
    <SystemObject SystemCode="SAP" LID="8327423434"
Status="active"></SystemObject>
    <Customer PersonCatCode="MEMBER" FirstName="ELIZABETH" FirstName_
Std="ELIZABETH"
     FirstName_Phon="E421" MiddleName="ANN" LastName="WARNER" LastName_
Std="WARNER"
     LastName_Phon="WARNAR" Suffix="" Title="DR" SSN="555999222"
     DOB="04/14/1964 00:00:00"Death="" Gender="F" MStatus="M" Race="C" Ethnic="31"
     Religion="AG" Language="EN" SpouseName="CRAIG" MotherName="JEN"
MotherMN="SMITH"
     FatherName="MARK" Maiden="MILLER" PobCity="MILFORD" PobState="ONTARIO"
     PobCountry="CAN" VIPFlag="NONE" VetStatus="NONE" Status=""
     DriverLicense="CT12941284" DriverLicenseSt="CT" Dod="" DeathCertificate=""
     Nationality="CAN" Citizenship="CAN">
      <Alias FirstName="LIZ" MiddleName="" LastName="MILLER"></Alias>
      <Address AddressType="H" AddressLine1="1330 BLOSSOM ST."
       AddressLine1_HouseNo="1330"AddressLine1_StDir="" AddressLine1_
StName="BLOSSOM"
       AddressLine1_StPhon="BLASAN" AddressLine1_StType="St" AddressLine2=""
       City="SHEFFIELD" StateCode="CT" PostalCode="09876" PostalCodeExt=""
       County="CAPE BURR" CountryCode="USA"></Address>
      <Phone PhoneType="CC" Phone="9894774477" PhoneExt=""></Phone>
    </Customer>
  </SBR>
</OutMsg>
```

Inbound Message Processing Logic

When records are transmitted to the OHMPI application, one of the "execute match" methods is usually called and a series of processes are performed to ensure that accurate and current data is maintained in the database. The execute match methods include `executeMatch`, `executeMatchUpdate`, `executeMatchDupRecalc`, and `executeMatchUpdateDupRecalc`. The MIDM uses `executeMatchGui`. For more information about how these methods differ, refer to the Javadocs provided with Oracle Healthcare Master Person Index.

The steps performed by the standard `executeMatch` method are outlined below, and the diagrams on the following pages illustrate the message processing flow. The processing steps performed in your environment might vary from this depending on how you customize the business process and environment.

The steps outlined below refer to the following parameters in the master.xml file. They are described in *Oracle Healthcare Master Person Index Configuration Reference*.

- OneExactMatch parameter
- SameSystemMatch parameter
- MatchThreshold parameter

- DuplicateThreshold parameter
- Update mode

Note: There are several decision points in the match process that can be defined by custom logic using custom plug-ins. For more information, see *Oracle Healthcare Master Person Index Configuration Reference*. These decision points are not listed in the below steps, which describe the default processing logic. Chapter 5, "Inbound Message Processing with Custom Logic" provides the same steps as below with the decision points included.

---

1. When a message is received by the OHMPI application, a search is performed for any existing records with the same local ID and system as those contained in the message. This search only includes records with a status of **A**, meaning only active records are included. If a matching record is found, an existing EUID is returned.
2. If an existing record is found with the same system and local ID as the incoming message, it is assumed that the two records represent the same object. Using the EUID of the existing record, the OHMPI application performs an update of the record's information in the database.
   - If the update does not make any changes to the object's information, no further processing is required and the existing EUID is returned.
   - If there are changes to the object's information, the updated record is inserted into the database, and the changes are recorded in the sbyn_ transaction table.
   - If there are changes to key fields (that is, fields used for matching or for the blocking query) and the update mode is set to **Pessimistic**, potential duplicates are reevaluated for the updated record.
3. If no records are found that match the record's system and local identifier, a second search is performed using the blocking query match search. A search is performed on each of the defined query blocks to retrieve a candidate pool of potential matches.

   Each record returned from the search is weighted using the fields defined for matching in the inbound message.
4. After the search is performed, the number of resulting records is calculated.
   - If a record or records are returned from the search with a matching probability weight above the match threshold, the OHMPI application performs exact match processing (see Step 5).
   - If no matching records are found, the inbound message is treated as a new record. A new EUID is generated and a new record is inserted into the database.
5. If records were found within the high match probability range, exact match processing is performed as follows:
   - If only one record is returned from this search with a matching probability that is equal to or greater than the match threshold, additional checking is performed to verify whether the records originated from the same system (see Step 6).

- If more than one record is returned with a matching probability that is equal to or greater than the match threshold and exact matching is set to false, then the record with the highest matching probability is checked against the incoming message to see if they originated from the same system (see Step 6).
- If more than one record is returned with a matching probability that is equal to or greater than the match threshold and exact matching is true, a new EUID is generated and a new record is inserted into the database.
- If no record is returned from the database search, or if none of the matching records have a weight in the exact match range, a new EUID is generated and a new record is inserted into the database.

Note: Exact matching is determined by the *OneExactMatch* parameter, and the match threshold is defined by the *MatchThreshold* parameter. For more information about these parameters, see *Oracle Healthcare Master Person Index Configuration Reference*.

---

6. When records are checked for same system entries, the OHMPI application tries to retrieve an existing local ID using the system of the new record and the EUID of the record that has the highest match weight.
   - If a local ID is found and same system matching is set to **true**, a new record is inserted, and the two records are considered to be potential duplicates. These records are marked as same system potential duplicates.
   - If a local ID is not found, the OHMPI application performs an update, following the process described in Step 2 earlier.
   - If a local ID is found and same system matching is set to **false**, it is assumed that the two records represent the same object. Using the EUID of the existing record, the OHMPI application performs an update, following the process described in Step 2 earlier.
7. If a new record is inserted, all records that were returned from the blocking query are weighed against the new record using the matching algorithm. If a record is updated and the update mode is pessimistic, the same occurs for the updated record. If the matching probability weight of a record is greater than or equal to the potential duplicate threshold, the record is flagged as a potential duplicate. For more information about thresholds and the update mode, see *Oracle Healthcare Master Person Index Configuration Reference*.

   The following flow charts provide a visual representation of the processes performed in the default configuration. Figure 2–4, "Inbound Message Processing Using executeMatch" and Figure 2–5, "Inbound Message Processing Using executeMatch (continued)" represent the primary flow of information. Figure 2–6, "Record Update Expansion" expands on update procedures illustrated in Figure 2-4 and Figure 2-5.

Figure 2–4 Inbound Message Processing Using executeMatch

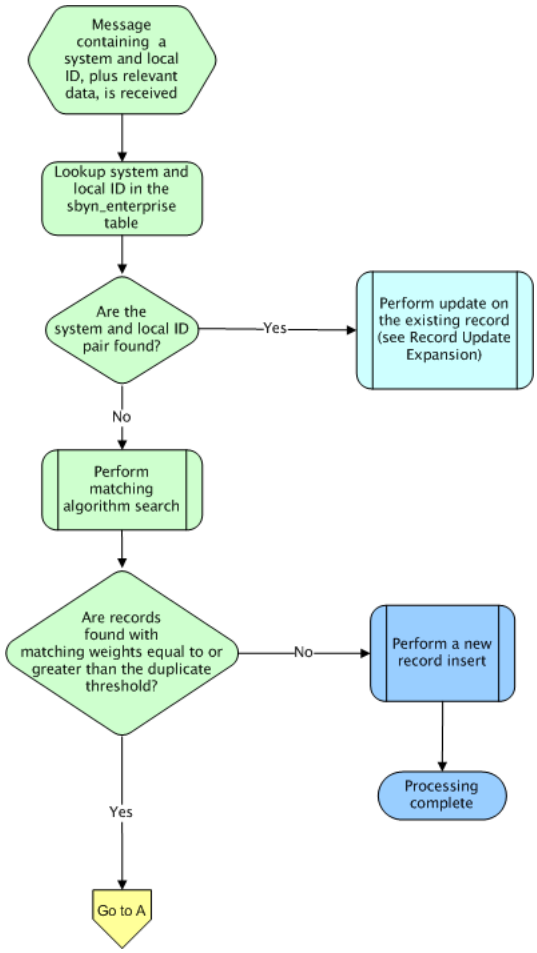


Figure 2–5 Inbound Message Processing Using executeMatch (continued)

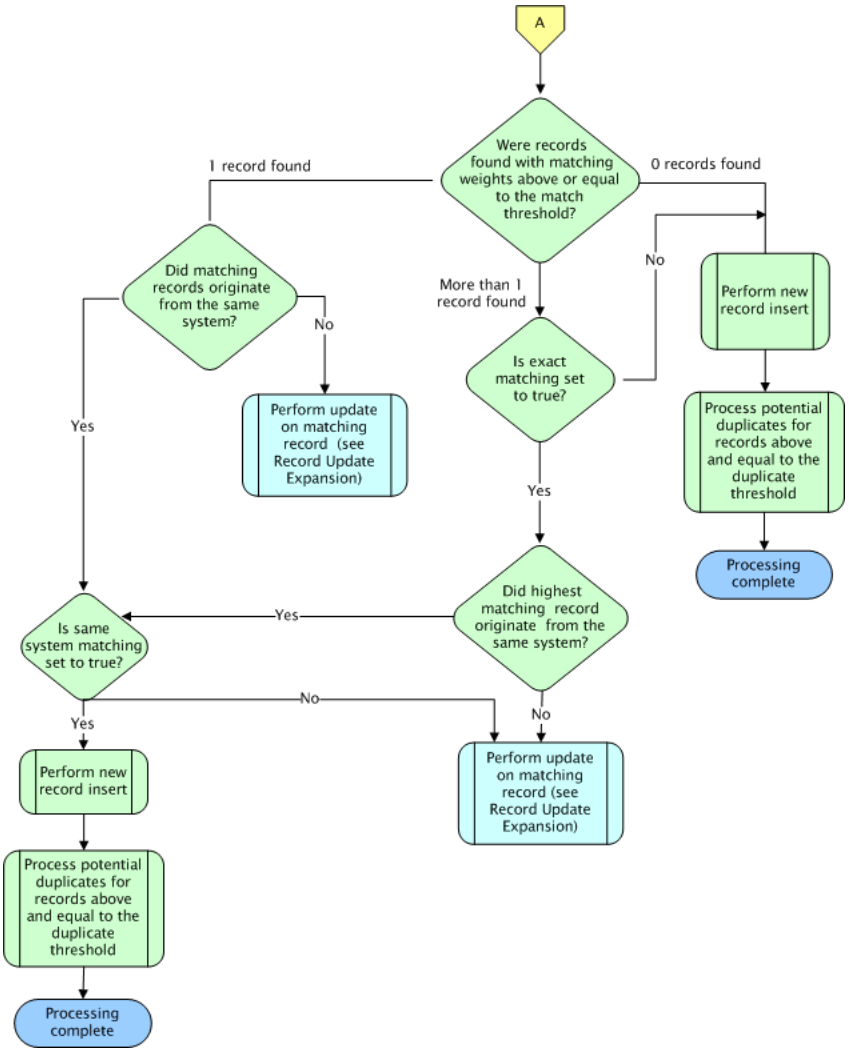
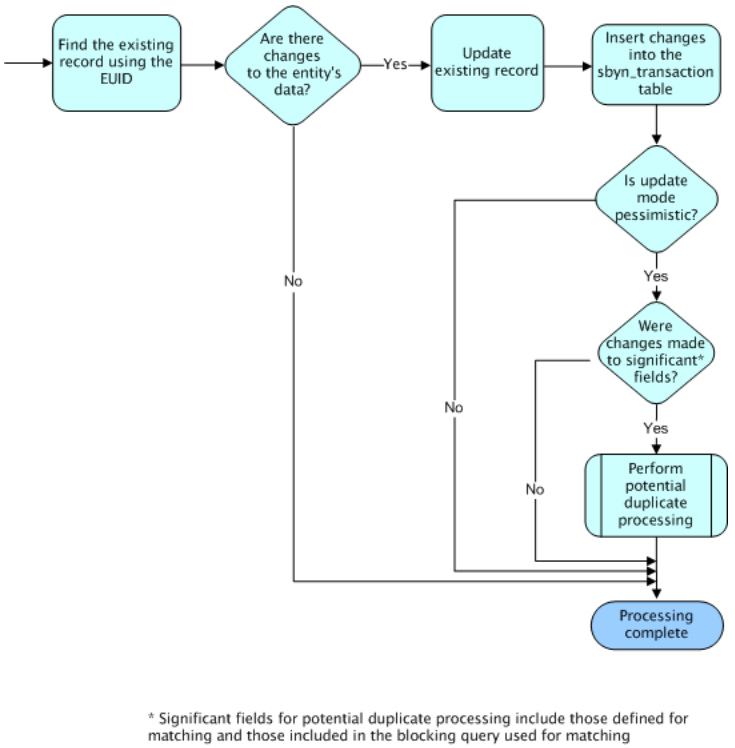


Figure 2–6 Record Update Expansion



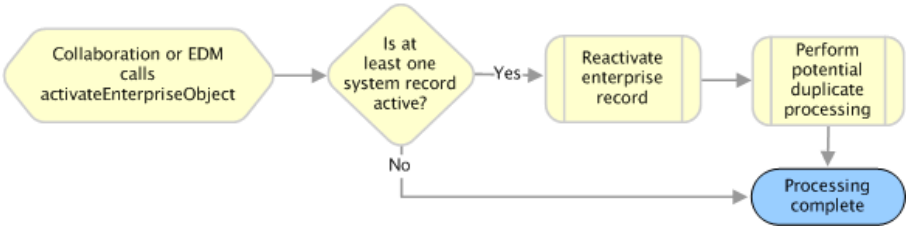
Primary Function Processing Logic

The primary functions of an OHMPI application can be performed from the MIDM or can be called from business processes, Java clients, web services, and so on. Whether potential duplicates are evaluated after a call to any of these functions is dependent on the update mode settings. Potential duplicates are only processed against the single best record (SBR) and not the system records. These functions are all defined in the `MasterController` and `MasterControllerEJB` classes, and are fully described in the Oracle Healthcare Master Person Index Javadocs. In the following diagrams, significant fields for potential duplicate processing include fields defined for matching and fields included in the blocking query used for matching. In all of the methods described below, an entry is made in the transaction history table (sbyn_transaction).

activateEnterpriseObject

This method reactivates an enterprise record. The MIDM calls this method when you reactivate a previously deactivated EUID. Since all potential duplicates were deleted when the EUID was originally deactivated, potential duplicates are always recalculated, regardless of the update mode. Figure 2–7 illustrates the processing steps.

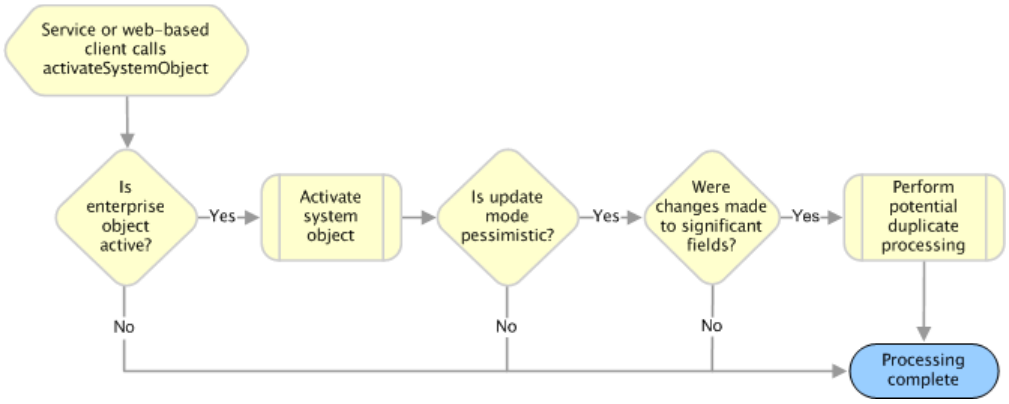
Figure 2–7 activateEnterpriseObject Processing



activateSystemObject

This method reactivates a system record. The MIDM calls this method when you reactivate a previously deactivated system record. If the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR. If key fields were updated, potential duplicates are recalculated for the enterprise record. Figure 2–8 illustrates the processing steps.

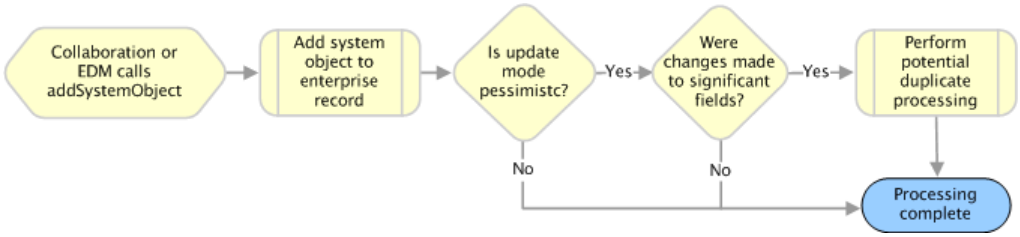
Figure 2–8 activateSystemObject Processing



addSystemObject

This method adds a system record to an enterprise record. The MIDM calls this method when you add a system record to an existing enterprise record from the Record Details window. If the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR. If key fields were updated and the update mode is set to **Pessimistic**, potential duplicates are recalculated for the enterprise record. Figure 2–9 illustrates the processing steps.

Figure 2–9 addSystemObject Processing



createEnterpriseObject

There are two `createEnterpriseObject` methods, both of which add a new enterprise record to the database and bypass any potential duplicate processing. One method takes only one system record as a parameter and the other takes an array of system records. These methods cannot be called from the MIDM and are designed for use in business processes, web services, or Java clients.

deactivateEnterpriseObject

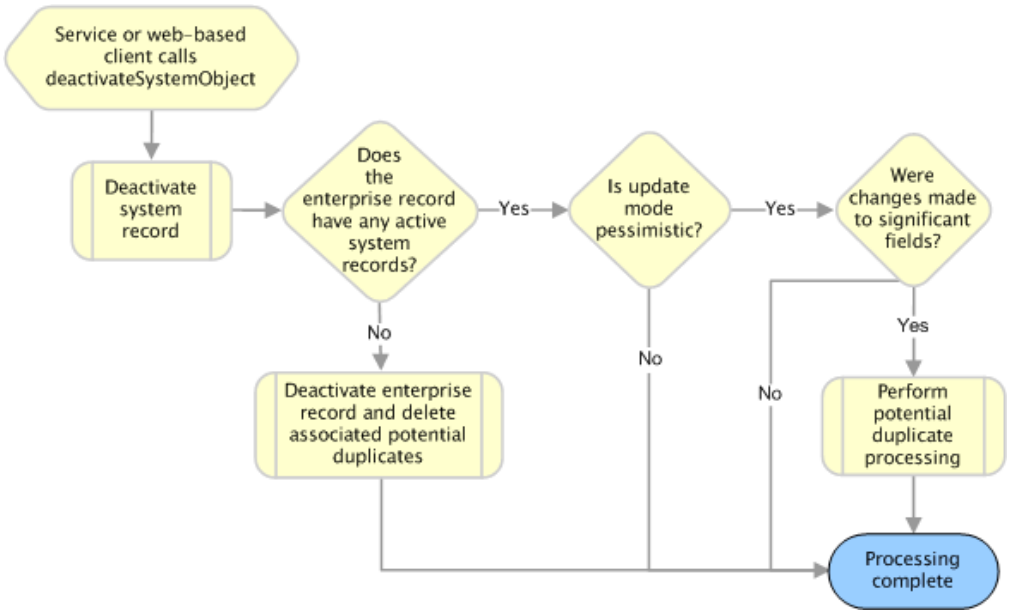
This method deactivates an enterprise record specified by its EUID. The MIDM calls this method when you deactivate an EUID from the Record Details page. When an enterprise record is deactivated, all potential duplicate listings for that record are deleted.

deactivateSystemObject

This method deactivates a system record in an enterprise record. The MIDM calls this method when you deactivate a system record. If the enterprise record containing this system record has no active system records remaining, the enterprise record is deactivated and all potential duplicate listings are deleted.

If the enterprise record has active system records after the transaction and the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR. If key fields were updated, potential duplicates are recalculated for the enterprise record. Figure 2–10 illustrates the processing steps.

Figure 2–10 deactivateSystemObject Processing



deduplicateSystemObject

This method deduplicates source system records in cases where the local ID of the incoming message is blank, null, or not required. This method is especially useful for the case in which the external source systems do not maintain or assign a local ID.

The following are the variations available with the deduplicateSystemObject API:

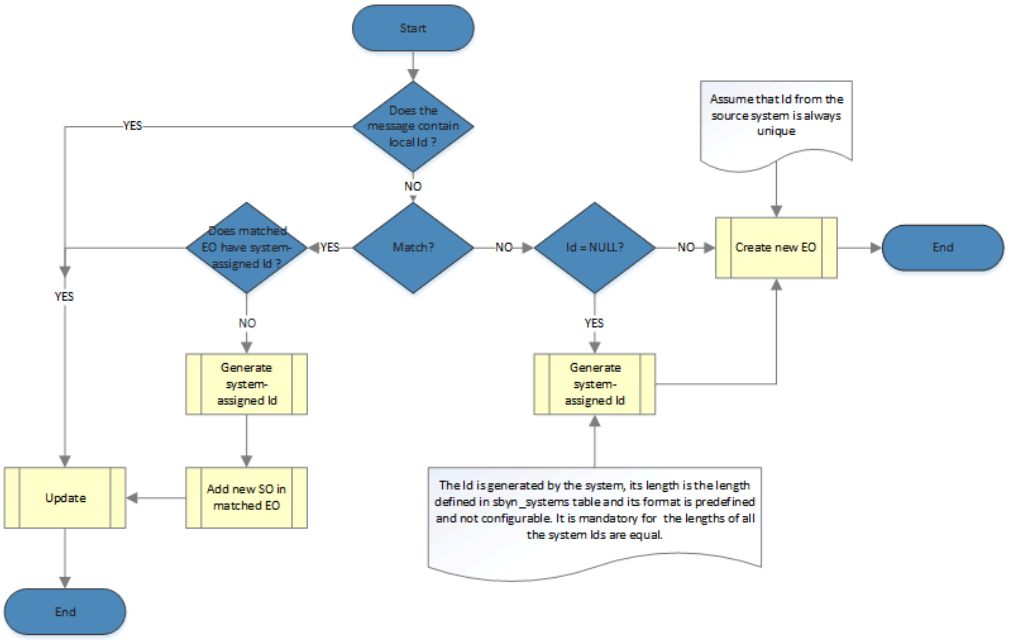
- **matchThreshold**: By default, the deduplication process uses the match threshold configured in master.xml. This parameter takes precedence over the one configured in master.xml.
- **transactionOption**: The following are the valid options:
  - **TRANSACTION_LOG_DELTA_DISABLED**: The delta information (BLOB) in each transaction log is not persisted. The record change history is no longer maintained in the MPI database, and operation which relies on the transaction delta information in the transaction log table does not work.
  - **TRANSACTION_LOG_DELTA_ENABLED**: The delta information (BLOB) in each transaction log is persisted. This is the default value for the transactionOption.

The deduplicateSystemObject API uses the local ID (if available) of the incoming record to check if it already exists in the database. If it already exists in the database, it updates the existing record.

If the incoming record does not exists in the database, the system performs a probabilistic match. If a match is found, EnterpriseObject/SBR is updated with the incoming message. If the system-assigned local ID system record exists, the system updates the system-assigned local ID system record. Otherwise, the system creates a new system record with the system-assigned local ID and updates the EnterpriseObject. If a match is not found, a new EnterpriseObject is created using the incoming message.

The format of local ID assigned by the internal system is *U[0-9]{9}*.

Figure 2–11 Handling Local Identifier



To use this API, you must use a different local ID validator. You can configure this in validation.xml as follow:

```
<ValidationConfig module-name="Validation"
parser-class="com.sun.mdm.index.configurator.impl.validation.ValidationConfigurati
on">
   <rules>
       <rule name="validate-local-id" object-name="SystemObject"
class="com.sun.mdm.index.objects.metadata.validation.SystemAssignedLocalIdValidato
r"/>
   </rules>
</ValidationConfig>
```

If you use a custom local ID validator, make sure it accepts the format for the system assigned local ID.

To deduplicate the source system records from the same system, you must enable the *SameSystemMatch* flag in master.xml.

```
<parameter>
     <parameter-name>SameSystemMatch</parameter-name>
     <parameter-type>java.lang.Boolean</parameter-type>
     <parameter-value>false</parameter-value>
</parameter>
```

Note: For projects created prior to OHMPI 2.0.13 patch, perform the following:

- Add an additional record in the SBYN_SEQ table. Execute the following SQL command to populating the SBYN_SEQ table:

```
INSERT INTO SBYN_SEQ_TABLE VALUES ('SYSTEM_ASSIGNED_LOCALID',
0);
```

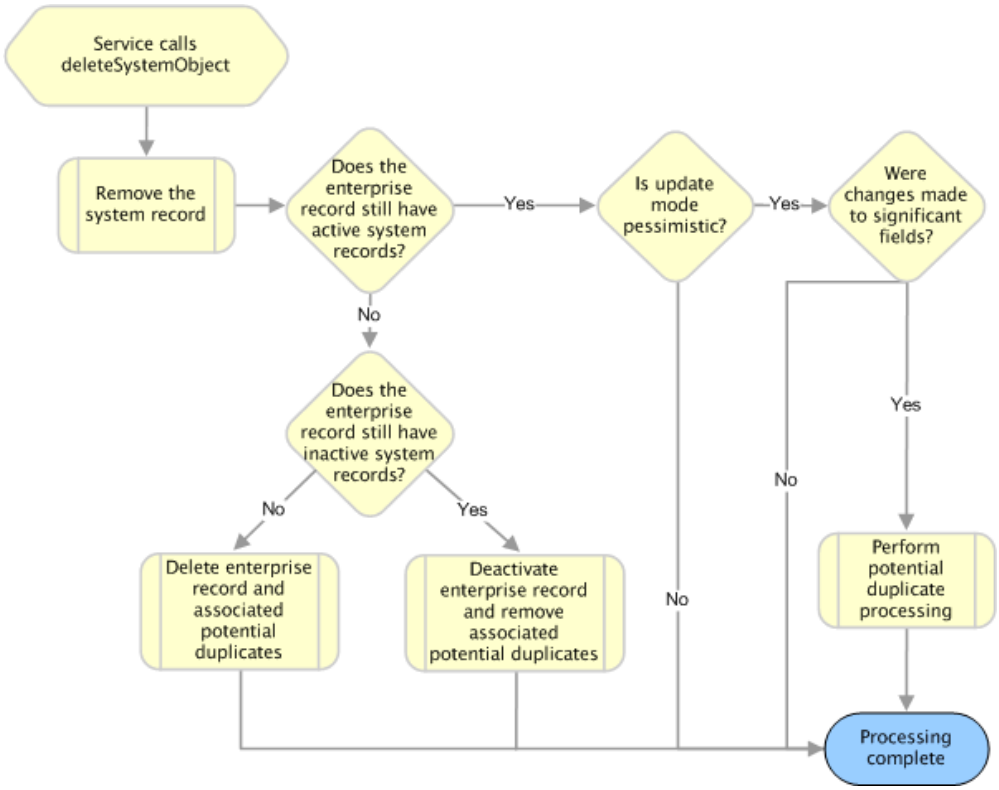
- For users with no Administrator privilege, add the following in security.xml for the corresponding role:

```
<name>deduplicateSystemObject</name>
```

deleteSystemObject

Unlike `deactivateSystemObject`, this method permanently removes a system record from an enterprise record. This method cannot be called from the MIDM. If the enterprise record containing the deleted system record has no active system records remaining, the enterprise record is deactivated (even if the enterprise record does have deactivated system records). If the enterprise record has no remaining system records after the system object is deleted, the enterprise record is also deleted. In both cases, any potential duplicate listings for that enterprise record are removed. If the enterprise record has active system records after the transaction and the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR. If key fields were updated, potential duplicates are recalculated for the enterprise record. Figure 2–12 illustrates the processing steps.

Figure 2–12 deleteSystemObject Processing

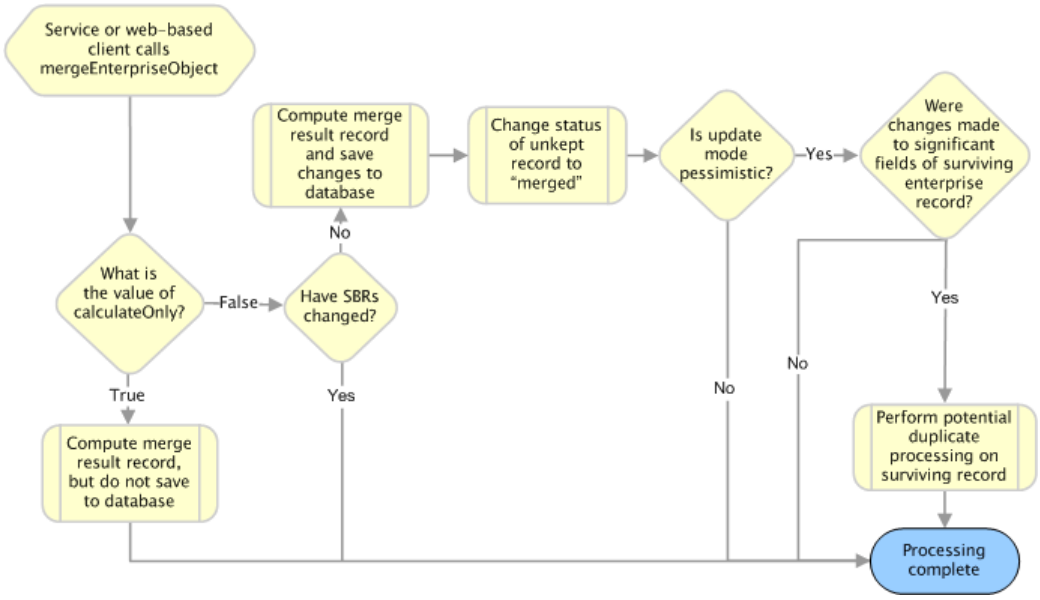


mergeEnterpriseObject

There are four `mergeEnterpriseObject` methods that merge two enterprise records (see the Javadocs provided with Oracle Healthcare Master Person Index for more information about each). The MIDM calls a merge method twice during a merge transaction. When you select records to merge and then click **Preview**, the method is called with the *calculateOnly* parameter set to **true** in order to display the merge result record for you to view. When you confirm the merge, the MIDM calls this method with the *calculateOnly* parameter set to **false** in order to commit the changes to the database and recalculate potential duplicates if needed. The method called by the MIDM checks the SBRs of the records involved in the merge against their corresponding SBRs in the database. If the SBRs differ, the merge is not performed since that means the records were changed by someone else during the merge process.

When this method is called with *calculateOnly* set to **false**, the application changes the status of the merged enterprise record to merged and deletes all potential duplicate listings for the merged enterprise record. If the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR of the surviving enterprise record. If key fields were updated, potential duplicates are recalculated for the enterprise record. Figure 2–13 illustrates the processing steps, and includes the check for SBR differences, which only occurs in two of the merge methods.

Figure 2–13 mergeEnterpriseObject Processing



mergeSystemObject

There are four methods that merge two system records that are either from the same enterprise record or from two different enterprise records (for more information about each method, see the Javadocs provided with Oracle Healthcare Master Person Index). The system records must originate from the same external system. The MIDM calls this method twice during a system record merge transaction. When you first click **Keep LID#** (where # is the heading number of the LID), the method is called with the *calculateOnly* parameter set to **true** in order to display the merge result record for you to view. When you confirm the merge, the MIDM calls this method with the *calculateOnly* parameter set to **false** in order to commit the changes to the database and recalculate potential duplicates if needed. Two of the merge methods compare the

SBRs of the records with their corresponding SBRs in the database to ensure that no updates were made to the records before finalizing the merge.

When this method is called with *calculateOnly* set to **false**, the application changes the status of the merged system record to *merged*. If the system records were merged within the same enterprise record and the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR. If key fields were updated, potential duplicates are recalculated for the enterprise record.

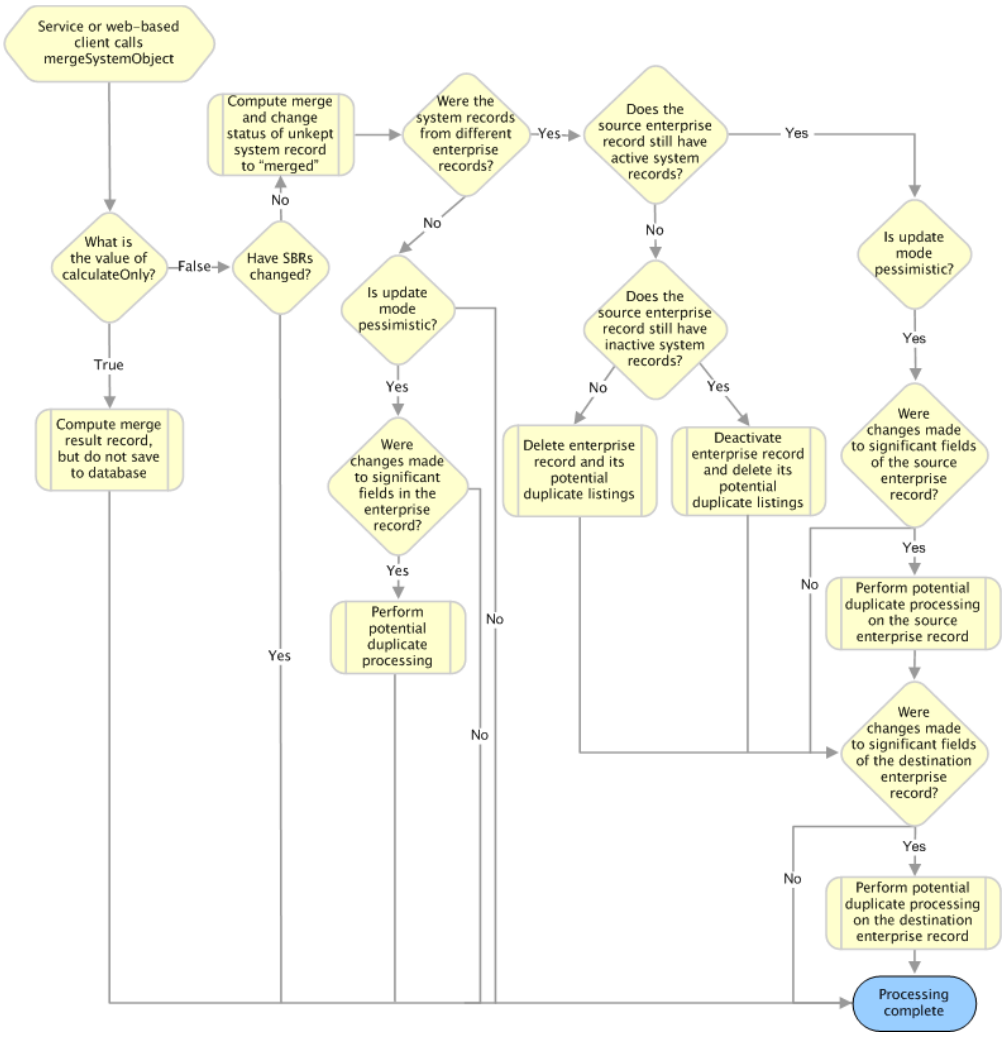
If the system records originated from two different enterprise records and the enterprise record that contained the unkept system record no longer has any active system records but does contain inactive system records, that enterprise record is deactivated and all associated potential duplicate listings are deleted.

Note: Note that if the system records are unmerged, the enterprise record is reactivated.

If the enterprise record that contained the unkept system record no longer has any system records, that enterprise record is deleted along with any potential duplicate listings.

If both enterprise records are still active and the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR for each enterprise record. If key fields were updated, potential duplicates are recalculated for each enterprise record. Figure 2–14 illustrates the processing steps, and includes the check for SBR differences, which only occurs in two of the merge methods.

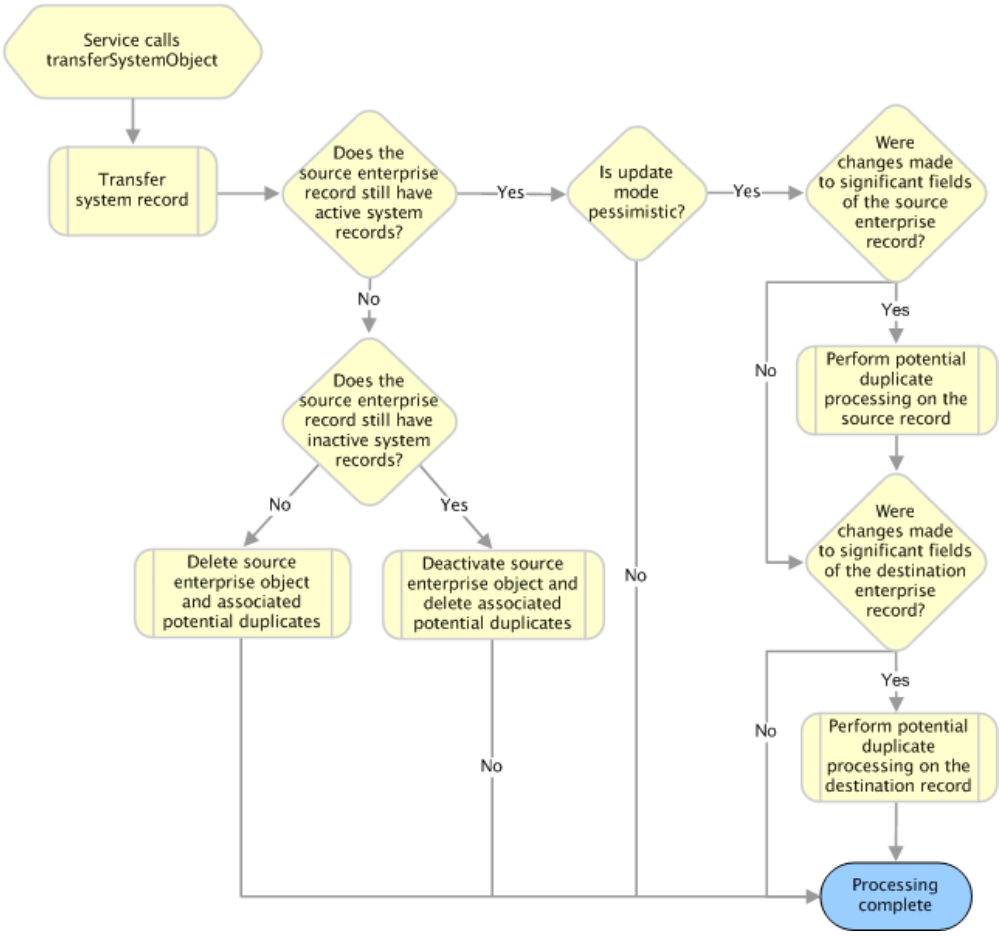
Figure 2–14 mergeSystemObject Processing



transferSystemObject

This transfers a system record from one enterprise record to another. This method is not called from the MIDM. If the enterprise record from which the system record was transferred no longer has any active system records (but still contains deactivated system records), that enterprise record is deactivated and any associated potential duplicate listings are removed. If the enterprise record from which the system record was transferred no longer has any system records, that enterprise record is deleted along with all associated potential duplicate listings. If both enterprise records are still active and the update mode is set to pessimistic, the application checks whether any key fields were updated in the SBR for each enterprise record. If key fields were updated, potential duplicates are recalculated for each enterprise record. Figure 2–15 illustrates the processing steps.

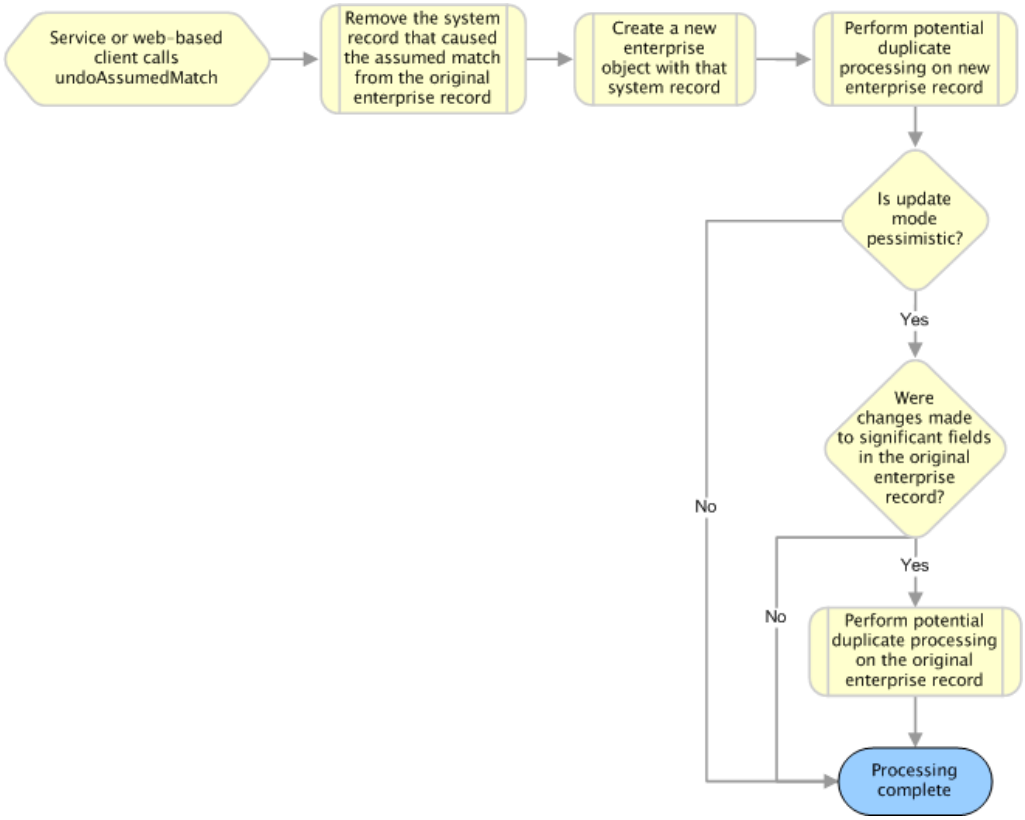
Figure 2–15 transferSystemObject Processing



undoAssumedMatch

This method reverses an assumed match made by the OHMPI application, using the information from the system record that created the assumed match to create a new enterprise record. The MIDM calls this method when you confirm the transaction after selecting Undo Assumed Match. Potential duplicates are calculated for the new record regardless of the update mode. If the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR of the original enterprise record. If key fields were updated, potential duplicates are recalculated for the enterprise record. Figure 2–16 illustrates the processing steps.

Figure 2–16 undoAssumedMatch Processing

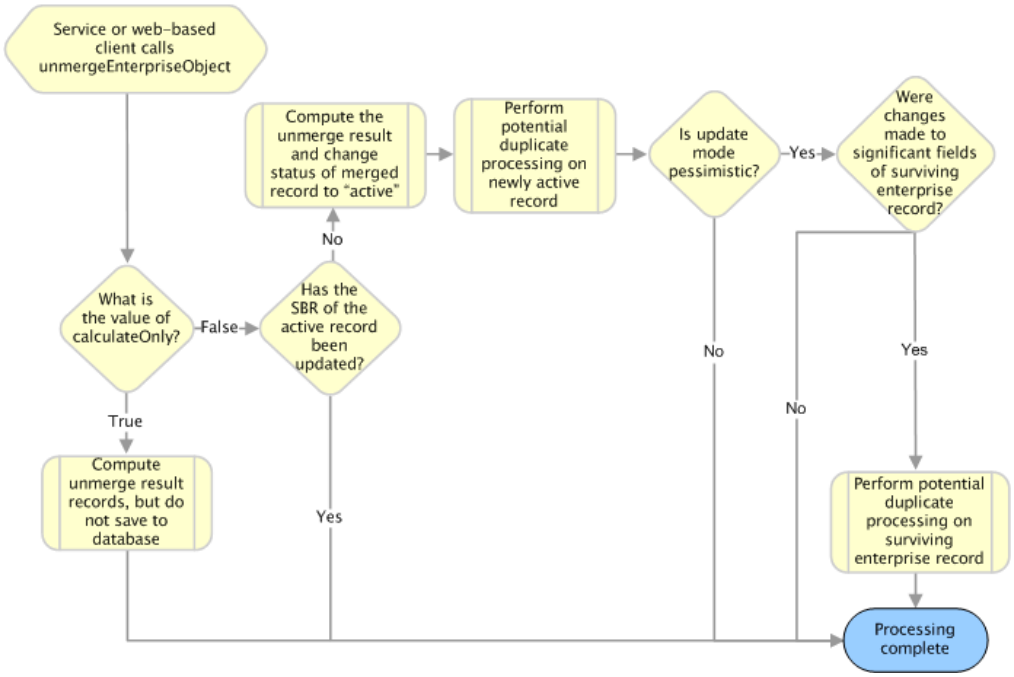


unmergeEnterpriseObject

There are two methods that unmerge two enterprise records that were previously merged. One method unmerges the record without checking to make sure the SBR of the active record was not changed by another process before finalizing the merge and one method performs the SBR check (see the Javadocs provided with Oracle Healthcare Master Person Index for more information). The MIDM calls this method twice during an unmerge transaction. When you first click **Unmerge**, the method is called with the *calculateOnly* parameter set to **true** in order to display the unmerge result records for you to view. When you confirm the unmerge, the MIDM calls this method with the *calculateOnly* parameter set to **false** in order to commit the changes to the database and recalculate potential duplicates.

When this method is called with *calculateOnly* set to **false**, the application changes the status of the merged enterprise record back to active and recalculates potential duplicate listings for the record. If the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR of the enterprise record that was still active after the merge. If key fields were updated, potential duplicates are recalculated for that enterprise record. Figure 2–17 illustrates the processing steps and includes the check for SBR updates.

Figure 2–17 unmergeEnterpriseObject Processing



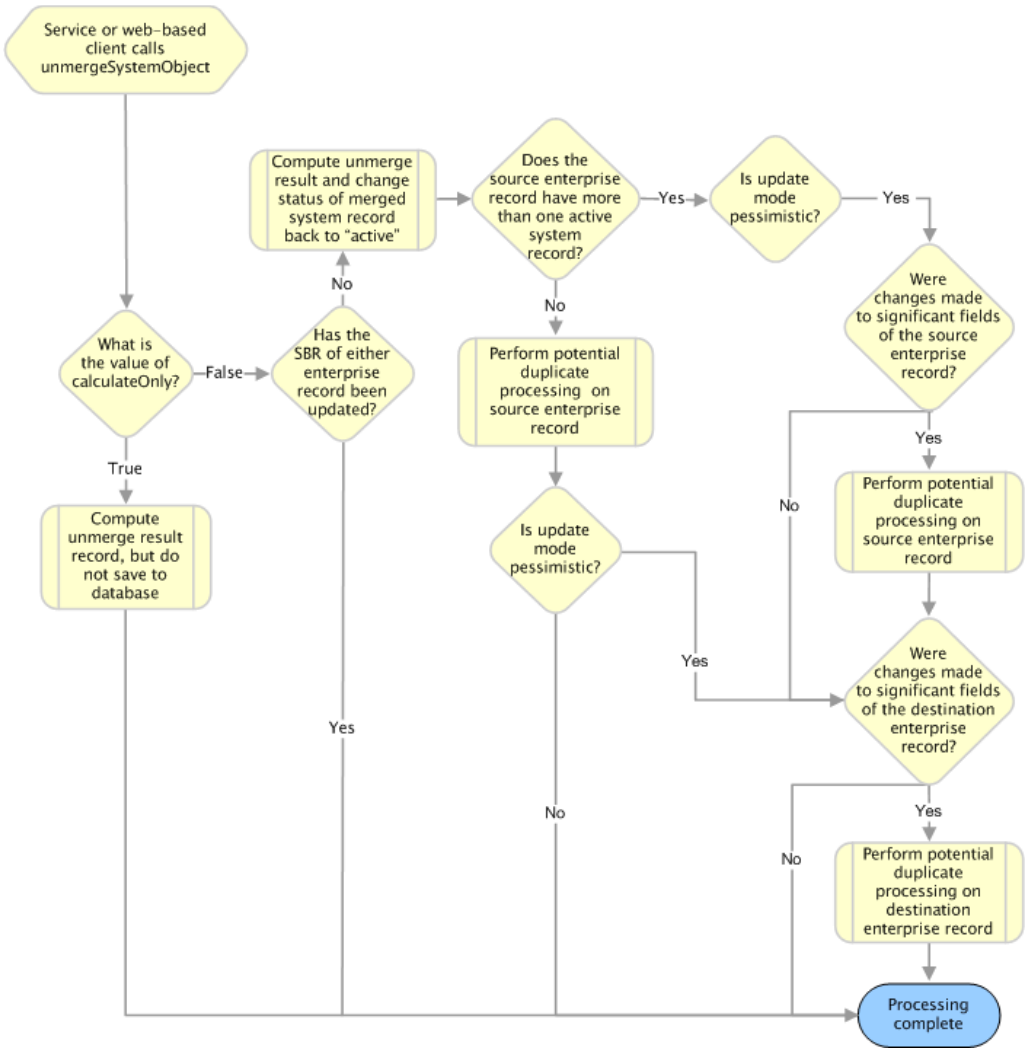
unmergeSystemObject

There are two methods that unmerge two system records that had previously been merged. One method unmerges the record without checking to make sure the SBR of the active record was not changed by another process before finalizing the merge and one method performs the SBR check (see the Javadocs provided with Oracle Healthcare Master Person Index for more information). The MIDM calls this method twice during a system record unmerge transaction. When you first click **Unmerge**, the method is called with the *calculateOnly* parameter set to **true** in order to display the unmerge result record for you to view. When you confirm the unmerge, the MIDM calls this method with the *calculateOnly* parameter set to **false** in order to commit the changes to the database and recalculate potential duplicates if needed.

When this method is called with *calculateOnly* set to **false**, the application changes the status of the merged system record back to active. If the source enterprise record (the record that contained the merge result system record after the merge) has more than one active system record after the unmerge and the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in that record. If key fields were updated, potential duplicates are recalculated for the source enterprise record.

If the source enterprise record has only one active system, potential duplicate processing is performed regardless of the update mode and of whether there were any changes to key fields. If the update mode is set to pessimistic, the application checks whether any key fields were updated in the SBR for the destination enterprise record. If key fields were updated, potential duplicates are recalculated for each enterprise record. Figure 2–18 illustrates the processing steps, assuming the system record unmerge involves two enterprise records and including the check for SBR updates.

Figure 2–18 unmergeSystemObject Processing

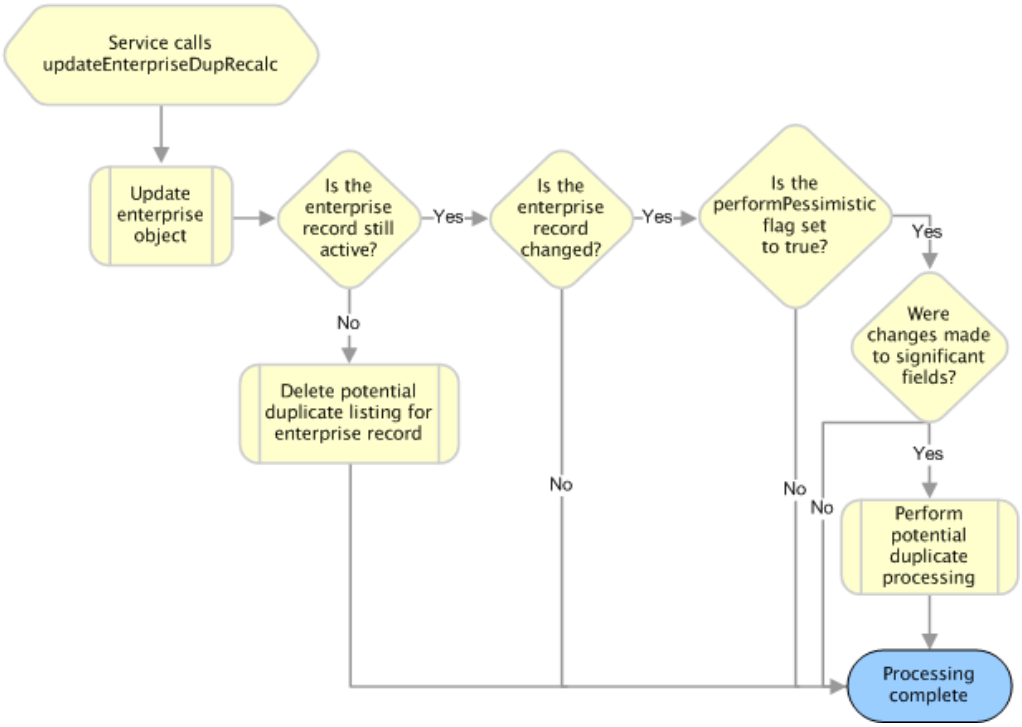


updateEnterpriseDupRecalc

This method updates the database to reflect new values for an enterprise record. It processes records in the same manner as `updateEnterpriseObject`, but provides an override flag for the update mode that allows you to defer potential duplicate processing. The MIDM does not call this method.

If the enterprise record is deactivated during the update, potential duplicates are deleted for that record. If the enterprise record was changed during the transaction but is still active and the *performPessimistic* parameter is set to **true**, the application checks whether any key fields were updated in the SBR of the enterprise record. If key fields were updated, potential duplicates are recalculated. Figure 2–19 illustrates the processing steps.

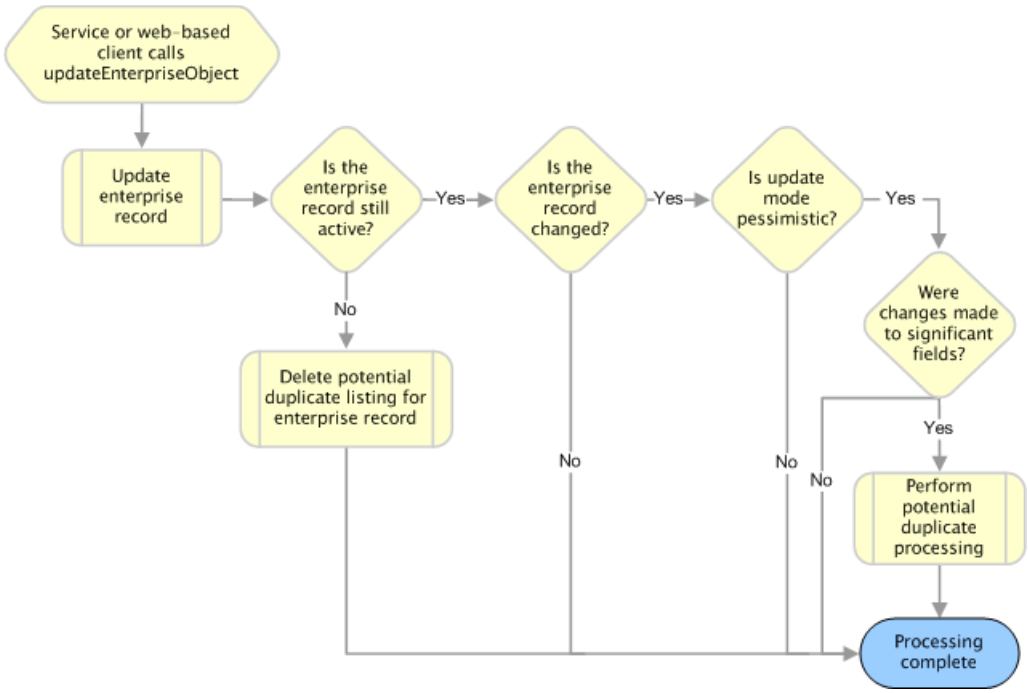
Figure 2–19 updateEnterpriseDupRecalc Processing



updateEnterpriseObject

This method updates the database to reflect new values for an enterprise record, and is called from the MIDM when you commit changes to an existing record. If the enterprise record is deactivated during the update, potential duplicates are deleted for that record. If the enterprise record is still active, was changed during the transaction, and the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR of the enterprise record. If key fields were updated, potential duplicates are recalculated. Figure 2–20 illustrates the processing steps.

Figure 2–20 updateEnterpriseObject Processing

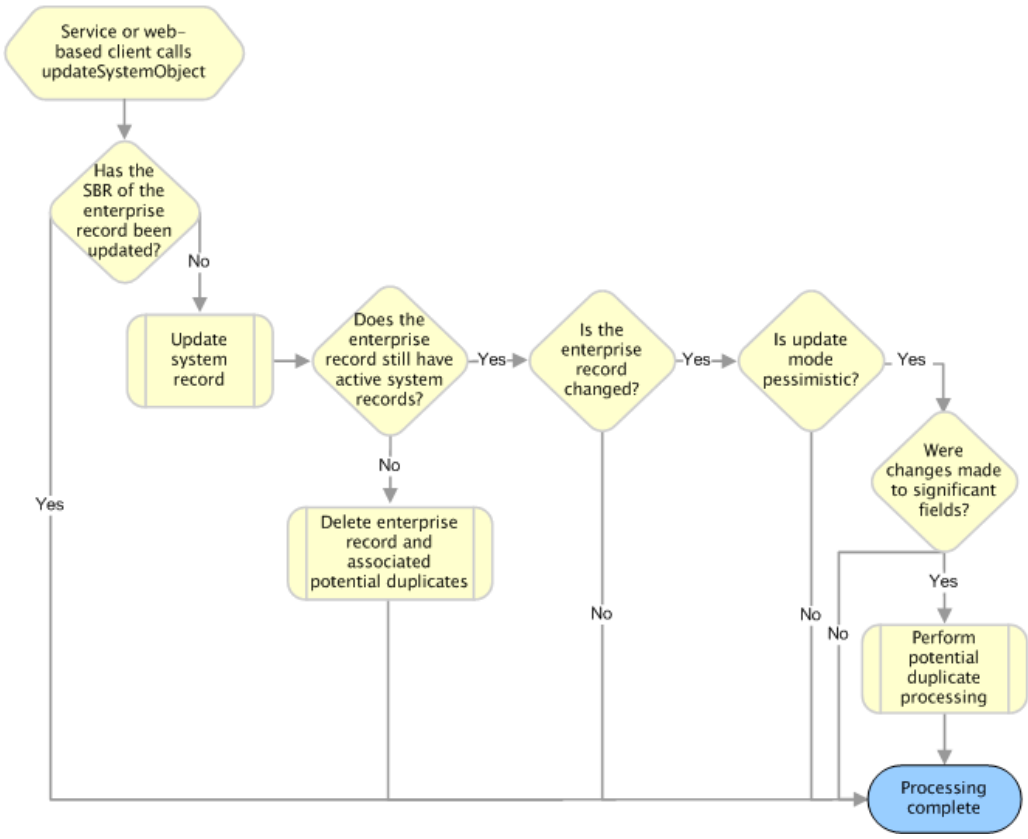


updateSystemObject

There are two methods that update the database to reflect new values for a system record. One method updates the record without checking that there were no concurrent changes to the record, and the other method compares the SBR of the associated enterprise object in the transaction with that in the database to be sure there were no concurrent changes (see the Javadocs provided with Oracle Healthcare Master Person Index for more information). The MIDM calls the method that checks for SBR changes when you commit changes to an existing system record.

If the enterprise record is deactivated during the update, potential duplicates are deleted for that record. If the enterprise record was changed during the transaction and is still active, and the update mode is set to **Pessimistic**, the application checks whether any key fields were updated in the SBR of the enterprise record. If key fields were updated, potential duplicates are recalculated. Figure 2–21 illustrates the processing steps and includes the check for SBR changes though it only occurs with one of the methods.

Figure 2–21 updateSystemObject Processing



3

The Database Structure

This chapter provides information about the Oracle Healthcare Master Person Index (OHMPI) database, including descriptions of each table and a sample entity relationship diagram. All information in this chapter pertains to the default version of the database. Your implementation might vary depending on the customization made to the OHMPI object structure and to the scripts used to create the OHMPI database.

This chapter includes the following sections:

Introducing the Structure of the Database Tables

The OHMPI database stores information about the entities being indexed, such as people or businesses. The database stores records from local systems in their original form and also stores a record for each object that is considered to be the single best record (SBR).

The structure of the database tables that store object information is dependent on the information specified in the `object.xml` file created by the wizard. Oracle Healthcare Master Person Index generates a script to create the tables and fields in the database based on the information in `object.xml`. If you update `object.xml`, regenerating the application updates the database scripts accordingly. This allows you to define the database as you define the object structure.

While most of the structures created in the database are based on information in `object.xml`, some of the tables, such as sbyn_seq_table and sbyn_common_detail, are standard for all implementations. The database includes tables that store information about the objects defined for the OHMPI application as well as tables that store common maintenance information, transactional information, and external system information. The database includes the tables listed in Table 3–1, " Oracle Healthcare Master Person Index Database Tables".

Table 3–1 Oracle Healthcare Master Person Index Database Tables

Table Name	Description
SBYN_OBJECT_NAME	Stores information for the parent objects associated with local system records. This database table is named by the parent object name. For example, a table storing company objects is named sbyn_company; a table storing person objects is named sbyn_person. Only one table stores parent object information for system records.

Table 3–1 (Cont.) Oracle Healthcare Master Person Index Database Tables

Table Name	Description
SBYN_OBJECT_NAMESBR	Stores information for the parent objects associated with single best records. This database table is named by the parent object name followed by *SBR*. For example, a table storing company objects is named sbyn_companysbr; a table storing person objects is named sbyn_personsbr. Only one table stores parent object information for SBRs.
SBYN_CHILD_OBJECT	Stores information for child objects associated with local system records. These database tables are named by their object name. For example, a table storing address objects is named sbyn_address; a table storing comment objects is named sbyn_comment. One database table is created for each child object defined in the object structure.
SBYN_CHILD_OBJECTSBR	Stores information for child objects associated with a single best record. These database tables are named by their object name followed by *SBR*. For example, a table storing address objects is named sbyn_addresssbr; a table storing comment objects is named sbyn_commentsbr. One SBR database table is created for each child object defined in the object structure.
SBYN_APPL	Lists the applications with which each item in stc_common_header is associated. Currently the only item in this table is **MPI**.
SBYN_ASSUMEDMATCH	Stores information about records that were automatically matched by the OHMPI application.
SBYN_AUDIT	Stores audit information about each time object information is accessed from the MIDM. Note: If audit logging is enabled, this table can grow very large and might require periodic archiving.
SBYN_COMMON_DETAIL	Contains all of the processing codes associated with the items listed in sbyn_common_header.
SBYN_COMMON_HEADER	Contains a list of the different types of processing codes used by the OHMPI application. These types are also associated with the drop-down lists you can specify for the MIDM.
SBYN_ENTERPRISE	Stores the local ID and system pairs, along with their associated EUID.
SBYN_MERGE	Stores information about all merge and unmerge transactions processed from either external systems or the MIDM.
SBYN_OVERWRITE	Stores information about fields that are locked for updates in an SBR.
SBYN_POTENTIALDUPLICATES	Stores a list of potential duplicate records and flags potential duplicate pairs that have been resolved.
SBYN_SEQ_TABLE	Stores the sequential codes that are used in other tables in the database, such as EUIDs, transaction numbers, and so on.
SBYN_SYSTEMOBJECT	Stores information about the system objects in the database, including the local ID and system, create date and user, status, and so on.
SBYN_SYSTEMS	Stores a list of systems in your organization, along with defining information.
SBYN_SYSTEMSBR	Stores transaction information about an SBR, such as the create or update date, status, and so on.
SBYN_TRANSACTION	Stores a history of changes to each record stored in the database.

Table 3–1 (Cont.) Oracle Healthcare Master Person Index Database Tables

Table Name	Description
SBYN_USER_CODE	Like the sbyn_common_detail table, this table stores processing codes and drop-down list values. This table contains additional validation information that allows you to validate information in a dependent field (for example, to validate cities against the entered postal code).

Understanding Database Table Details

The tables on this and the following pages describe each column in the default database tables. The columns are identical for Oracle, MySQL, and SQL Server databases, but the data types differ in some cases. Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences" lists the data type differences.

Table 3–2 Oracle, MySQL, and SQL Server Data Type Differences

Oracle Data Type	SQL Server Data Type	MySQL Data Type
BLOB	Varbinary(MAX)	blob
DATE	DateTime	datetime
INTEGER	Int	integer
LONG	Varchar(MAX)	mediumtext
NUMBER	Numeric	decimal
TIMESTAMP	DateTime	datetime
VARCHAR2	Varchar	varchar

SBYN_OBJECT_NAME

This table stores the parent object in each system record received by the OHMPI application. It is linked to the tables that store each child object in the system record by the *object_name* id column (where *object_name* is the name of the parent object). This table contains the columns listed below regardless of the design of the object structure, and also contains a column for each field you defined for the parent object in `object.xml`. Columns to store standardized or phonetic versions of certain fields are automatically added when you specify certain match types in the wizard.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–3 SBYN_OBJECT_NAME Table Description

Column Name	Data Type	Column Description
SYSTEMCODE	VARCHAR2(20)	The system code for the system record.
LID	VARCHAR2(25)	A local identification code assigned by the specified system.

Table 3–3 (Cont.) SBYN_OBJECT_NAME Table Description

Column Name	Data Type	Column Description
OBJECT_NAMEID	VARCHAR2(20)	A unique ID for the parent object in a system record. This is named according to the parent object. For example, if the parent object is "Company", the name of this column is "companyid"; if the parent object is "Person", the name of this column is "personid".

SBYN_OBJECT_NAMESBR

This table stores the parent object of the SBR for each enterprise object in the OHMPI database. It is linked to the tables that store each child object in the SBR by the *object_name* id column (where *object_name* is the name of the parent object). This table contains the columns listed below regardless of the design of the object structure, and also contains a column for each field defined for the parent object in `object.xml`. In addition, columns to store standardized or phonetic versions of certain fields are automatically added when you specify certain match types in the wizard.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–4 SBYN_OBJECT_NAMESBR Table Description

Column Name	Data Type	Column Description
EUID	VARCHAR2(20)	The enterprise unique identifier assigned by the OHMPI application.
OBJECT_NAMEID	VARCHAR2(20)	A unique ID for the parent object in a system record. This is named according to the parent object. For example, if the parent object is "Company", the name of this column is "companyid"; if the parent object is "Person", the name of this column is "personid".

SBYN_CHILD_OBJECT

The sbyn_*child_object* tables (where *child_object* is the name of a child object in the object structure) store information about the child objects associated with a system record in the OHMPI application. All tables storing child object information for system records contain the columns listed below. The remaining columns are defined by the fields you specify for each child object in the object structure definition file, including any standardized or phonetic fields.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–5 SBYN_CHILD_OBJECT Table Description

Column Name	Data Type	Column Description
*OBJECT_NAME*ID	VARCHAR2(20)	The unique ID for the parent object associated with the child object in the system record.
*CHILD_OBJECT*ID	VARCHAR2(20)	The unique ID for each record in the table. This column cannot be null.

SBYN_CHILD_OBJECTSBR

The sbyn_*child_object*s br tables (where *child_object* is the name of a child object in the object structure) store information about the child objects associated with an SBR in the OHMPI application. All tables storing child object information for SBRs contain the columns listed below. The remaining columns are defined by the fields you specify for each child object in `object.xml`, including any standardized or phonetic fields.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–6 SBYN_CHILD_OBJECT and SBYN_CHILD_OBJECTSBR Table Description

Column Name	Data Type	Column Description
*OBJECT_NAME*ID	VARCHAR2(20)	The unique ID for the parent object associated with the child object in the SBR.
*CHILD_OBJECT*ID	VARCHAR2(20)	The unique ID for each record in the table. This column cannot be null.

SBYN_APPL

This table stores information about the applications used in the Oracle Healthcare Master Person Index system. Currently, there is only one entry, **MPI**. The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–7 SBYN_APPL Table Description

Column Name	Data Type	Description
APPL_ID	NUMBER(10)	The unique sequence number code for the listed application.
CODE	VARCHAR2(8)	A unique code for the application.
DESCR	VARCHAR2(30)	A brief description of the application.
READ_ONLY	CHAR(1)	An indicator of whether the current entry can be modified. If the value of this column is **Y**, the entry cannot be modified.
CREATE_DATE	DATE	The date the application entry was created.

Table 3–7 (Cont.) SBYN_APPL Table Description

Column Name	Data Type	Description
CREATE_USERID	VARCHAR2(20)	The logon ID of the user who created the application entry.

SBYN_ASSUMEDMATCH

This table maintains a record of each assumed match transaction that occurs in the OHMPI application, allowing you to review these transactions and, if necessary, reverse an assumed match. This table can grow quite large over time; it is recommended that the table be archived periodically.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–8 SBYN_ASSUMEDMATCH Table Description

Column Name	Data Type	Description
ASSUMEDMATCHID	VARCHAR2(20)	The unique ID for the assumed match transaction.
EUID	VARCHAR2(20)	The EUID into which the incoming record was merged.
SYSTEMCODE	VARCHAR2(20)	The system code for the source system (that is, the system from which the incoming record originated).
LID	VARCHAR2(25)	The local ID of the record in the source system.
WEIGHT	VARCHAR2(20)	The matching weight between the incoming record and the EUID record into which it was merged.
TRANSACTION NUMBER	VARCHAR2(20)	The transaction number associated with the assumed match.

SBYN_AUDIT

This table maintains a log of each instance in which any of the OHMPI database tables are accessed through the MIDM. This includes each time a record appears on a search results page, a comparison page, the Record Details page, and so on. This log is only maintained if the MIDM is configured for it. This table can grow very large over time and might require periodic archiving.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–9 SBYN_AUDIT Table Description

Column Name	Data Type	Description
AUDIT_ID	VARCHAR2(20)	The unique identification code for the audit record. This column cannot be null.
PRIMARY_OBJECT_TYPE	VARCHAR2(20)	The name of the parent object as defined in object.xml.

Table 3–9 (Cont.) SBYN_AUDIT Table Description

Column Name	Data Type	Description
EUID	VARCHAR2(15)	The EUID whose information was accessed during an MIDM transaction.
EUID_AUX	VARCHAR2(15)	The second EUID whose information was accessed during an MIDM transaction. A second EUID appears when viewing information about merge and unmerge transactions, comparisons, and so on.
FUNCTION (Oracle/MySQL) OPERATION (SQL Server)	VARCHAR2(32)	The type of transaction that caused the audit record to be written. This column cannot be null.
DETAIL	VARCHAR2(120)	A brief description of the transaction that caused the audit record to be written.
CREATE_DATE	DATE	The date the transaction that created the audit record was performed. This column cannot be null.
CREATE_BY	VARCHAR2(20)	The user ID of the person who performed the transaction that caused the audit log. This column cannot be null.

SBYN_COMMON_DETAIL

This table stores the processing codes and description for all of the common maintenance data elements. This is the detail table for sbyn_common_header. Each data element in sbyn_common_detail is associated with a data type in sbyn_common_header by the common_header_id column. None of the columns in this table can be null.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–10 SBYN_COMMON_DETAIL Table Description

Column Name	Data Type	Description
COMMON_DETAIL_ID	NUMBER(10)	The unique identification code of the common table data element.
COMMON_HEADER_ID	NUMBER(10)	The unique identification code of the common table data type associated with the data element (as stored in the common_header_id column of the sbyn_common_header table).
CODE	VARCHAR2(20)	The processing code for the common table data element.
DESCR	VARCHAR2(50)	A description of the common table data element.

Table 3–10 (Cont.) SBYN_COMMON_DETAIL Table Description

Column Name	Data Type	Description
READ_ONLY	CHAR(1)	An indicator of whether the common table data element can be modified.
CREATE_DATE	DATE	The date the data element record was created.
CREATE_USERID	VARCHAR2(20)	The user ID of the person who created the data element record.

SBYN_COMMON_HEADER

This table stores a description of each type of common maintenance data and is the header table for sbyn_common_detail. Together, these tables store the processing codes and drop-down menu descriptions for each common table data type. For a person index, common table data types might include Religion, Language, Marital Status, and so on. For a business index, common table data types might include Address Type, Phone Type, and so on. None of the columns in this table can be null.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–11 SBYN_COMMON_HEADER Table Description

Column Name	Data Type	Description
COMMON_HEADER_ID	VARCHAR2(10)	The unique identification code of the common table data type.
APPL_ID	VARCHAR2(10)	The application ID from sbyn_ appl that corresponds to the application for which the common table data type is used.
CODE	VARCHAR2(8)	A unique processing code for the common table data type.
DESCR	VARCHAR2(50)	A description of the common table data type.
READ_ONLY	CHAR(1)	An indicator of whether an entry in the table is read-only (if this column is set to "Y", the entry is read-only).
MAX_INPUT_LEN	NUMBER(10)	The maximum number of characters allowed in the code column for the common table data type.
TYP_TABLE_CODE	VARCHAR2(3)	This column is not currently used.
CREATE_DATE	DATE	The date the common table data type record was created.
CREATE_USERID	VARCHAR2(20)	The user ID of the person who created the common table data type record.

SBYN_ENTERPRISE

This table stores a list of all the system and local ID pairs assigned to the enterprise records in the database, along with the associated EUID for each pair. This table is linked to sbyn_systemobject by the system code and LID columns, and is linked to sbyn_systemsbr by the euid column. This table maintains links between the SBR and its associated system objects. None of the columns in this table can be null.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–12 SBYN_ENTERPRISE Table Description

Column Name	Data Type	Description
SYSTEMCODE	VARCHAR2(20)	The processing code of the system associated with the local ID.
LID	VARCHAR2(25)	The local ID associated with the system and EUID.
EUID	VARCHAR2(20)	The EUID associated with the local ID and system.

SBYN_MERGE

This table maintains a record of each merge transaction that occurs in the OHMPI application, both through the MIDM and from external systems. It also records any unmerges that occur. The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–13 SBYN_MERGE Table Description

Column Name	Data Type	Description
MERGE_ID	VARCHAR2(20)	The unique, sequential identification code of merge record. This column cannot be null.
KEPT_EUID	VARCHAR2(20)	The EUID of the record that was retained after the merge transaction. This column cannot be null.
MERGED_EUID	VARCHAR2(20)	The EUID of the record that was not retained after the merge transaction.
MERGE_ TRANSACTIONNUM	VARCHAR2(20)	The transaction number associated with the merge transaction. This column cannot be null.
UNMERGE_ TRANSACTIONNUM	VARCHAR2(20)	The transaction number associated with the unmerge transaction.

SBYN_OVERWRITE

This table stores information about the fields that are locked or linked for updates in the SBRs. It stores the EUID of the SBR, the ePath to the field, and the current locked value of the field or the system code and LID of the linked system object. The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–14 SBYN_OVERWRITE Table Description

Column Name	Data Type	Description
EUID	VARCHAR2(20)	The EUID of an SBR containing fields for which the overwrite lock is set.
PATH	VARCHAR2(200)	The ePath to a field that is locked in an SBR from the MIDM.
TYPE	VARCHAR2(20)	The data type of a field that is locked in an SBR.
INTEGERDATA	NUMBER(38)	The data that is locked for overwrite in an integer field.
BOOLEANDATA	NUMBER(38)	The data that is locked for overwrite in a boolean field or the system code and LID of the linked system object.
STRINGDATA	VARCHAR2(200)	The data that is locked for overwrite in a string field or the system code and LID of the linked system object.
BYTEDATA	CHAR(2)	The data that is locked for overwrite in a byte field or the system code and LID of the linked system object.
LONGDATA	LONG	The data that is locked for overwrite in a long integer field or the system code and LID of the linked system object.
DATEDATA	DATE	The data that is locked for overwrite in a date field or the system code and LID of the linked system object.
FLOATDATA	NUMBER(38,4)	The data that is locked for overwrite in a floating decimal field or the system code and LID of the linked system object.
TIMESTAMPDATA	DATE	The data that is locked for overwrite in a timestamp field or the system code and LID of the linked system object.

SBYN_POTENTIALDUPLICATES

This table maintains a list of all records that are potential duplicates of one another. It also maintains a record of whether a potential duplicate pair has been resolved or permanently resolved. The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–15 SBYN_POTENTIALDUPLICATES Table Description

Column Name	Data Type	Description
POTENTIALDUPLICATEID	VARCHAR2(20)	The unique identification number of the potential duplicate transaction.
WEIGHT	VARCHAR2(20)	The matching weight of the potential duplicate pair.
TYPE	VARCHAR2(15)	This column is reserved for future use.

Table 3–15 (Cont.) SBYN_POTENTIALDUPLICATES Table Description

Column Name	Data Type	Description
DESCRIPTION	VARCHAR2(120)	A description of what caused the potential duplicate flag.
STATUS	VARCHAR2(15)	The status of the potential duplicate pair. The possible values are: ■ **U** - Unresolved ■ **R** - Resolved ■ **A** - Resolved permanently
HIGHMATCHFLAG	VARCHAR2(15)	This column is reserved for future use.
RESOLVEDUSER	VARCHAR2(30)	The user ID of the person who resolved the potential duplicate status.
RESOLVEDDATE	DATE	The date the potential duplicate status was resolved.
RESOLVEDCOMMENT	VARCHAR2(120)	Comments regarding the resolution of the duplicate status. This is not currently used.
EUID2	VARCHAR2(20)	The EUID of the second record in the potential duplicate pair.
TRANSACTIONNUMBER	VARCHAR2(20)	The transaction number associated with the transaction that produced the potential duplicate flag.
EUID1	VARCHAR2(20)	The EUID of the first record in the potential duplicate pair.

SBYN_SEQ_TABLE

This table controls and maintains a record of the sequential identification numbers used in various tables in the database, ensuring that each number is unique and assigned in order. Several of the ID numbers maintained in this table are determined by the object structure. The numbers are assigned sequentially, but are cached in chunks of 1000 numbers for optimization (so the application does not need to query the sbyn_seq_table table for each transaction). The chunk size for the EUID sequence is configurable. If the Repository server is reset before all allocated numbers are used, the unused numbers are discarded and never used, and numbering is restarted at the beginning of the next 1000-number chunk.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–16 SBYN_SEQ_TABLE Table Description

Column Name	Data Type	Description
SEQ_NAME	VARCHAR2(20)	The name of the object for which the sequential ID is stored.
SEQ_COUNT	NUMBER(38)	The current value of the sequence. The next record will be assigned the current value plus one.

The default sequence numbers are listed in the following table.

Table 3–17 Default Sequence Numbers

Sequence Name	Description
EUID	The sequence number that determines how EUIDs are assigned to new records. The chunk size for the EUID sequence number is configurable in the `master.xml` file.
POTENTIALDUPLICATE	The sequence number assigned each potential duplicate transaction record in sbyn_potentialduplicates (column name "potentialduplicateid").
TRANSACTIONNUMBER	The sequence number assigned to each transaction in the OHMPI application. This number is stored in sbyn_transaction (column name "transactionnumber").
ASSUMEDMATCH	The sequence number assigned to each assumed match transaction record in sbyn_assumedmatch (column name "assumedmatchid").
AUDIT	The sequence number assigned to each audit log record in sbyn_audit (column name "audit_id").
MERGE	The sequence number assigned to each merge transaction in sbyn_merge (column name "merge_id").
SBYN_APPL	The sequence number assigned to each application listed in sbyn_appl (column name "appl_id").
SBYN_COMMON_HEADER	The sequence number assigned to each common table data type listed in sbyn_common_header (column name "common_header_id").
SBYN_COMMON_DETAIL	The sequence number assigned to each common table data element listed in sbyn_common_detail (column name "common_detail_id").
OBJECT_NAME	Each parent and child object system record table is assigned a sequential ID. The column names are named after the object (for example, sbyn_address has a sequential column named "addressid"). The parent object ID is included in each child object table.
*OBJECT_NAME*SBR	Each parent and child object SBR table is assigned a sequential ID. The column names are named after the object (for example, sbyn_addresssbr has a sequential column named "addressid"). The parent object ID is included in each child object SBR table.

SBYN_SYSTEMOBJECT

This table stores information about the system records in the database, including their local ID and source system pairs. It also stores transactional information, such as the create or update date and function. The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–18 SBYN_SYSTEMOBJECT Table Description

Column Name	Data Type	Description
SYSTEMCODE	VARCHAR2(20)	The processing code of the system associated with the local ID. This column cannot be null.

Table 3–18 (Cont.) SBYN_SYSTEMOBJECT Table Description

Column Name	Data Type	Description
LID	VARCHAR2(25)	The local ID associated with the system and EUID (the associated EUID is found in sbyn_ enterprise). This column cannot be null.
CHILDTYPE	VARCHAR2(20)	The type of object being processed (currently only the name of the parent object). This column is reserved for future use.
CREATEUSER	VARCHAR2(30)	The user ID of the person who created the system record.
CREATEFUNCTION	VARCHAR2(20)	The type of transaction that created the system record.
CREATEDATE	DATE	The date the system record was created.
UPDATEUSER	VARCHAR2(30)	The user ID of the person who last updated the system record.
UPDATEFUNCTION	VARCHAR2(20)	The type of transaction that last updated the system record.
UPDATEDATE	DATE	The date the system record was last updated.
STATUS	VARCHAR2(15)	The status of the system record. The status can be one of these values: ■ active ■ inactive ■ merged

SBYN_SYSTEMS

This table stores information about each system integrated into the Oracle Healthcare Master Person Index environment, including the system's processing code and name, a brief description, the format of the local IDs, and whether any of the system information should be masked. The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–19 SBYN_SYSTEMS Table Description

Column Name	Data Type	Description
SYSTEMCODE	VARCHAR2(20)	The unique processing code of the system.
DESCRIPTION	VARCHAR2(120)	A brief description of the system, or the system name. This is the value that appears in the tree view panes of the MIDM for each system and local ID pair.
STATUS	CHAR(1)	The status of the system in the OHMPI application. **A** indicates active and **D** indicates deactivated.

Table 3–19 (Cont.) SBYN_SYSTEMS Table Description

Column Name	Data Type	Description
ID_LENGTH	NUMBER	The length of the local identifiers assigned by the system. This length does not include any additional characters added by the input mask.
FORMAT	VARCHAR2(60)	The required data pattern for the local IDs assigned by the system. For more information about possible values and using Java patterns, see "Patterns" in the class list for `java.util.regex` in the Javadocs provided with the Java™ 2 Platform, Standard Edition (J2SE™ platform). Note that the data pattern is also limited by the input mask described below. All regex patterns are supported if there is no input mask.
INPUT_MASK	VARCHAR2(60)	A mask used by the MIDM to add punctuation to the local ID. For example, the input mask **DD-DDD-DDD** inserts a hyphen after the second and fifth characters in an 8-digit ID. These character types can be used. ■ **D** - Numeric character ■ **L** - Alphabetic character ■ **A** - Alphanumeric character
VALUE_MASK	VARCHAR2(60)	A mask used to strip any extra characters that were added by the input mask for database storage. The value mask is the same as the input mask, but with an "x" in place of each punctuation mark. Using the input mask described above, the value mask is **DDxDDDxDDD**. This strips the hyphens before storing the ID.
CREATE_DATE	DATE	The date the system information was inserted into the database.
CREATE_USERID	VARCHAR2(20)	The logon ID of the user who inserted the system information into the database.
UPDATE_DATE	DATE	The most recent date the system's information was updated.
UPDATE_USERID	VARCHAR2(20)	The logon ID of the user who last updated the system's information.

SBYN_SYSTEMSBR

This table stores transactional information about the system records for the SBR, such as the create or update date and function. The sbyn_systemsbr table is indirectly linked to the sbyn_systemobjects table through sbyn_enterprise. The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–20 SBYN_SYSTEMSBR Table Description

Column Name	Data Type	Description
EUID	VARCHAR2(20)	The EUID associated with system record (the associated system and local ID are found in sbyn_ enterprise). This column cannot be null.
CHILDTYPE	VARCHAR2(20)	The type of object being processed (currently only the name of the parent object). This column is reserved for future use.
CREATESYSTEM	VARCHAR2(20)	The system in which the system record was created.
CREATEUSER	VARCHAR2(30)	The user ID of the person who created the system record.
CREATEFUNCTION	VARCHAR2(20)	The type of transaction that created the system record.
CREATEDATE	DATE	The date the system object was created.
UPDATEUSER	VARCHAR2(30)	The user ID of the person who last updated the system record.
UPDATEFUNCTION	VARCHAR2(20)	The type of transaction that last updated the system record.
UPDATEDATE	DATE	The date the system object was last updated.
STATUS	VARCHAR2(15)	The status of the enterprise record. The status can be one of these values: ■ active ■ inactive ■ merged
REVISIONNUMBER	NUMBER(38)	The revision number of the SBR. This is used for version control.

SBYN_TRANSACTION

This table stores a history of changes made to each record in the OHMPI application, allowing you to view a transaction history and to undo certain actions, such as merging two object profiles. The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–21 SBYN_TRANSACTION Table Description

Column Name	Data Type	Description
TRANSACTIONNUMBER	VARCHAR2(20)	The unique number of the transaction.
LID1	VARCHAR2(25)	This column is reserved for future use.
LID2	VARCHAR2(25)	The local ID of the second system record involved in the transaction.

Table 3–21 (Cont.) SBYN_TRANSACTION Table Description

Column Name	Data Type	Description
EUID1	VARCHAR2(20)	This column is reserved for future use.
EUID2	VARCHAR2(20)	The EUID of the second object profile involved in the transaction.
FUNCTION (Oracle/MySQL) OPERATION (SQL Server)	VARCHAR2(20)	The type of transaction that occurred, such as update, add, merge, and so on.
SYSTEMUSER	VARCHAR2(30)	The logon ID of the user who performed the transaction.
TIMESTAMP	TIMESTAMP	The date and time the transaction occurred.
DELTA	BLOB	A list of the changes that occurred to system records as a result of the transaction.
SYSTEMCODE	VARCHAR2(20)	The processing code of the source system in which the transaction originated.
LID	VARCHAR2(25)	The local ID of the system record involved in the transaction.
EUID	VARCHAR2(20)	The EUID of the enterprise record involved in the transaction.

SBYN_USER_CODE

This table is similar to the sbyn_common_header and sbyn_common_detail tables in that it stores processing codes and drop-down list values. This table is used when the value of one field is dependent on the value of another. For example, if you store credit card information, you could list each credit card type and specify a required format for the credit card number field. The data stored in this table includes the processing code, a brief description, and the format of the dependent fields.

The following table lists Oracle data types. For information about the differences in data types between database vendors, see Table 3–2, " Oracle, MySQL, and SQL Server Data Type Differences".

Table 3–22 SBYN_USER_CODE Table Description

Column Name	Data Type	Description
CODE_LIST	VARCHAR2(20)	The code list name of the user code type (using the credit card example above, this might be similar to "CREDCARD"). This column links the values for each list.
CODE	VARCHAR2(20)	The processing code of each user code element.
DESCRIPTION	VARCHAR2(50)	A brief description or name for the user code. This is the value that appears in the drop-down list.

Table 3–22 (Cont.) SBYN_USER_CODE Table Description

Column Name	Data Type	Description
FORMAT	VARCHAR2(60)	The required data pattern for the field that is constrained by the user code. For more information about possible values and using Java patterns, see "Patterns" in the class list for `java.util.regex` in the Javadocs provided with the J2SE platform. Note that the data pattern is also limited by the input mask described below. All regex patterns are supported if there is no input mask.
INPUT_MASK	VARCHAR2(60)	A mask used by the MIDM to add punctuation to the constrained field. For example, the input mask **DD-DDD-DDD** inserts a hyphen after the second and fifth characters in an 8-digit ID. These character types can be used. ■ **D** - Numeric character ■ **L** - Alphabetic character ■ **A** - Alphanumeric character
VALUE_MASK	VARCHAR2(60)	A mask used to strip any extra characters that were added by the input mask for database storage. The value mask is the same as the input mask, but with an "x" in place of each punctuation mark. Using the input mask described above, the value mask is **DDxDDDxDDD**. This strips the hyphens before storing the ID.

Viewing a Sample Database Model

The diagrams on the following pages illustrate the table structure and relationships for a sample Oracle Healthcare Master Person Index Oracle database designed for storing information about companies. The diagrams display attributes for each database column, such as the field name, data type, whether the field can be null, and primary keys. They also show directional relationships between tables and the keys by which the tables are related. This diagram is very similar to SQL Server, with the exception of a few column name changes and some different data types as noted in the tables above.

Figure 3–1 Sample Database Model (along with the following two diagrams

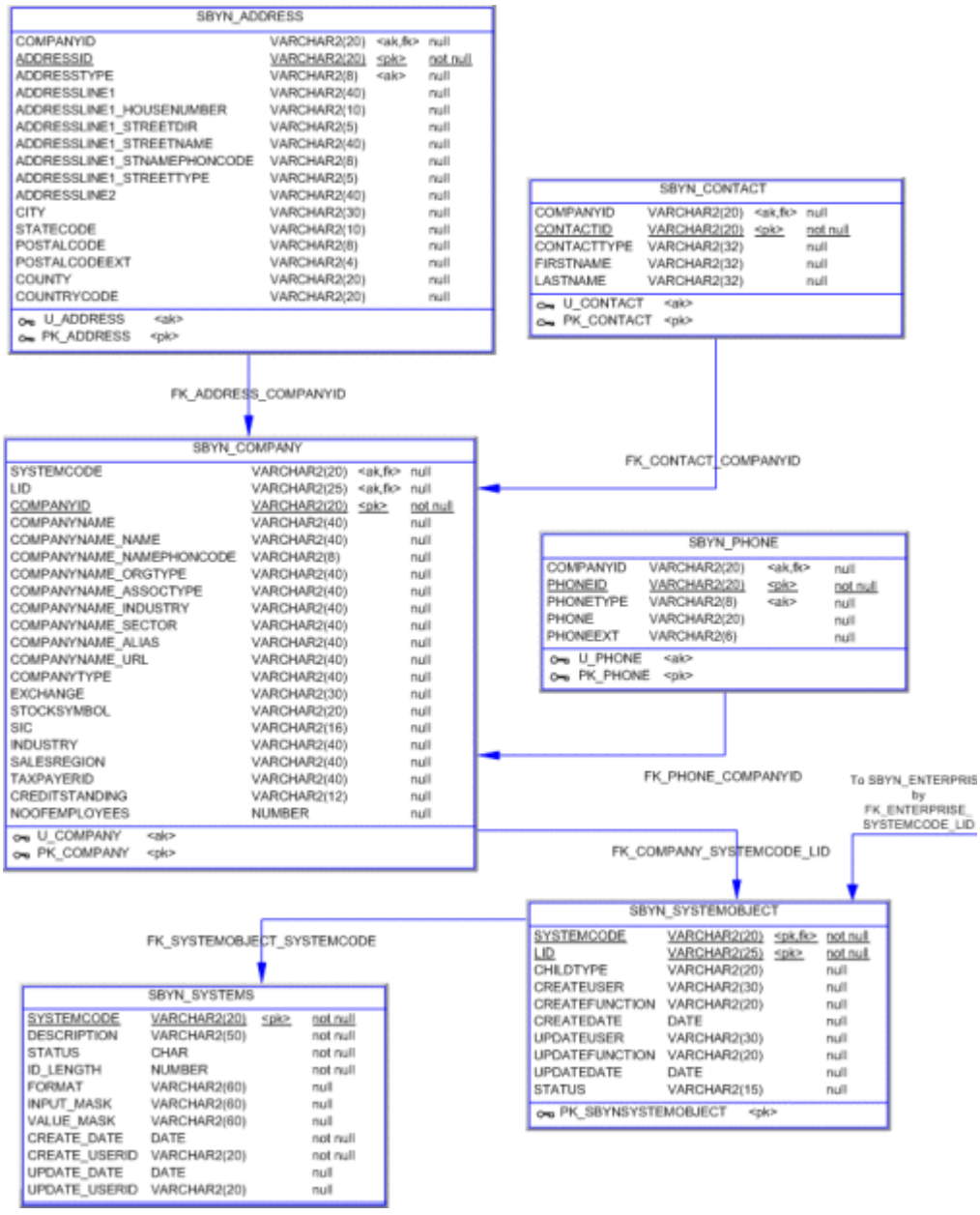


Figure 3–2 Sample Database Model (along with the previous and following diagrams)

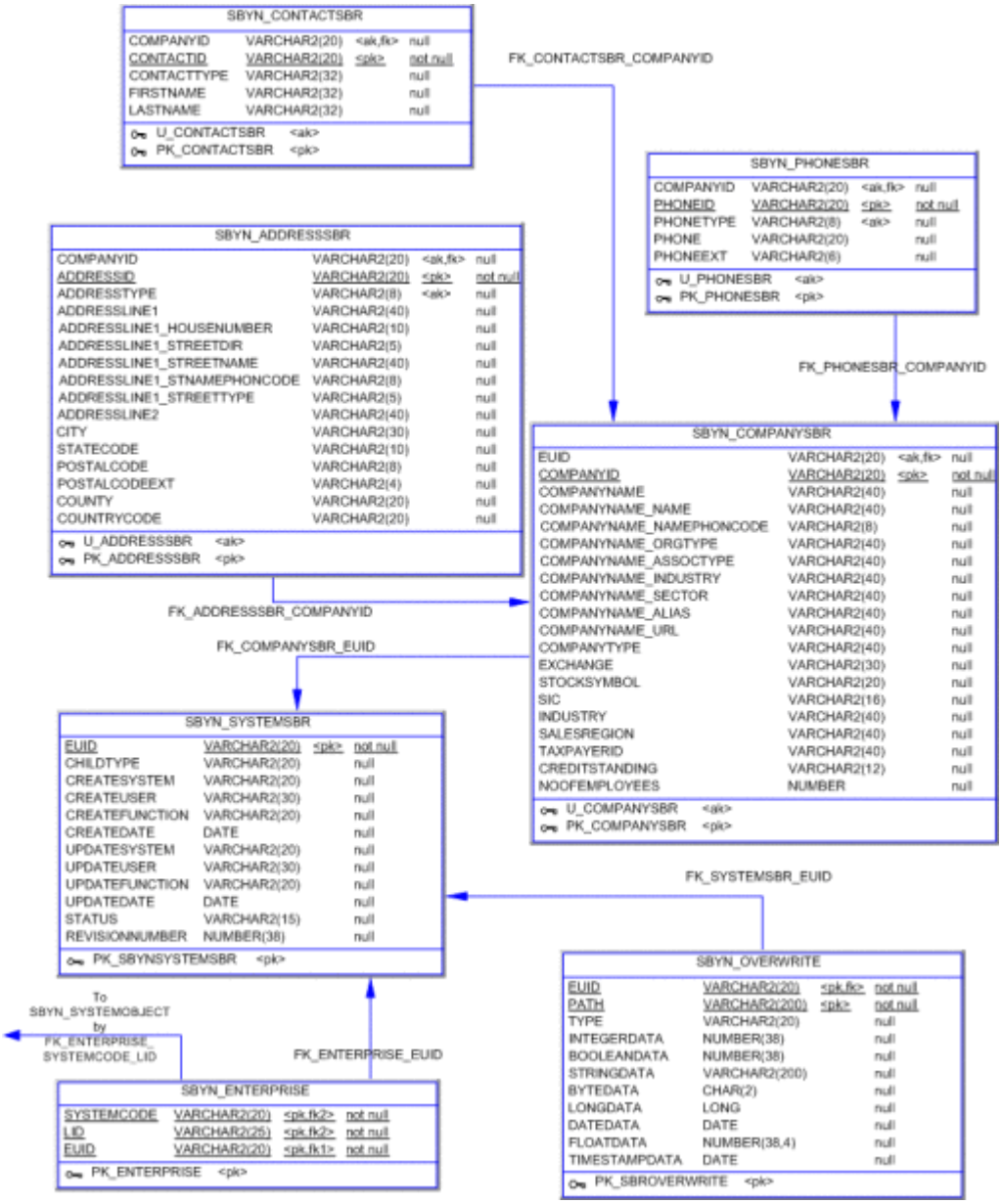
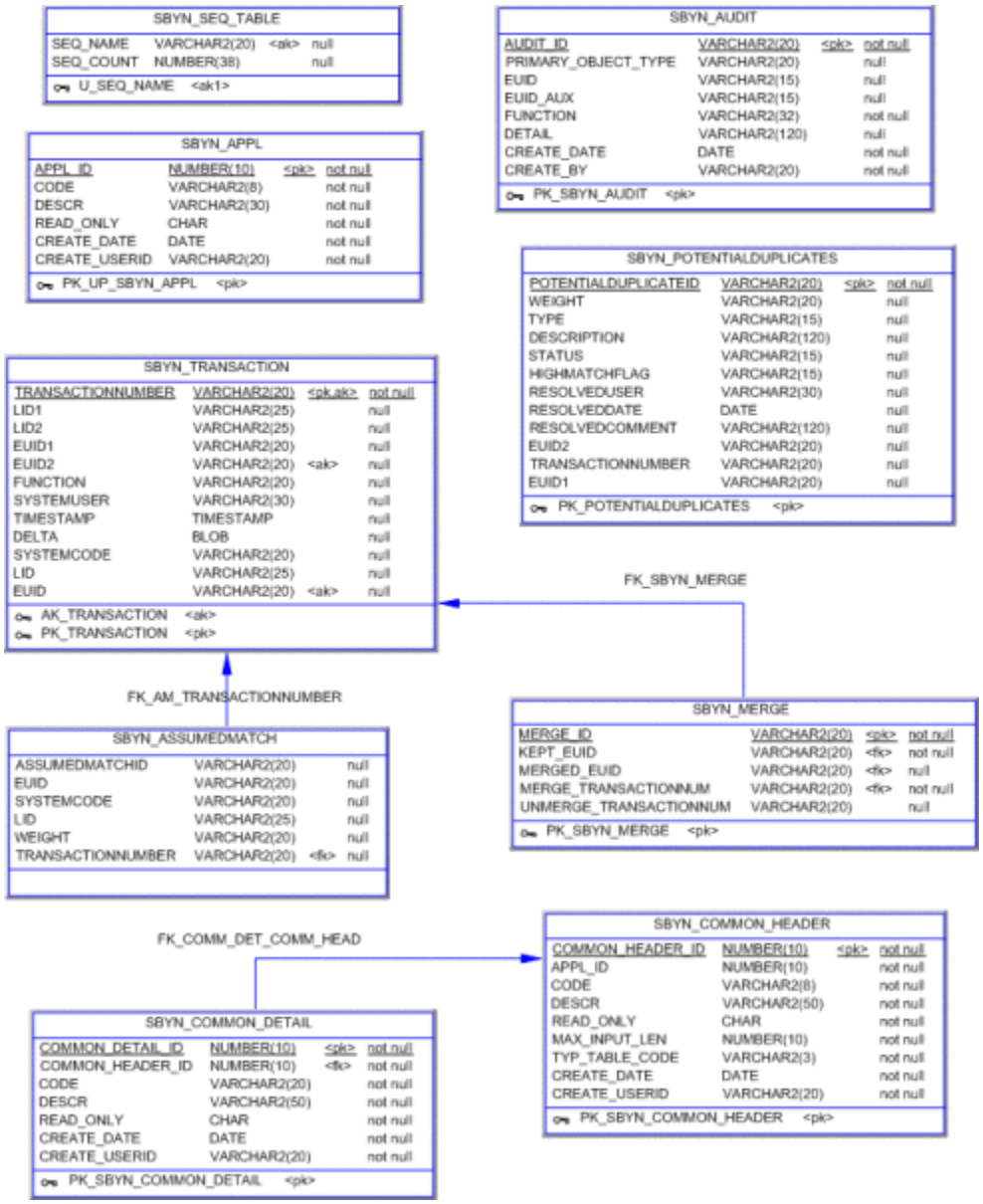


Figure 3–3 Sample Database Model (along with the previous two diagrams)



4

Working with the Java API

This chapter provides an overview of the Java API for an Oracle Healthcare Master Person Index (OHMPI) application. For detailed information about the JAVA API classes and methods, refer to the Oracle Healthcare Master Person Index Javadocs. Unless otherwise noted, all classes and methods described in this chapter are public. Methods inherited from classes other than those described in this chapter are listed, but not described:

This chapter includes the following sections:

Understanding Java Class Types

Oracle Healthcare Master Person Index provides several Java classes and methods to use to transform and process data in a Master Person Index or IHE Profile project. The master person index API is specifically designed to help you maintain the integrity of the data in the database by providing specific methods for updating, adding, and merging records in the database.

Oracle Healthcare Master Person Index provides a set of JAVA API classes that can be used with any object structure for the OHMPI application.

JAVA API Classes

JAVA API classes provide the methods you need to perform basic data cleansing and processing functions against incoming data, such as performing searches, reviewing potential duplicates, adding and updating records, and merging and unmerging records. The primary class containing these functions is the `MasterController` class, which includes the `executeMatch` methods. Several classes support the `MasterController` class by defining additional objects and functions. Documentation for the JAVA API classes methods is provided in Javadoc format.

Java API Methods

activateEnterpriseObject

Description

This method changes the status of a deactivated enterprise object back to active.

Syntax

```
public void activateEnterpriseObject(java.lang.String euid)
```

Parameters

Name	Description
euid	The EUID associated with the enterprise object to activate.

Returns

None

Throws

- ProcessingException
- UserException

activateSystemObject

Description

This method changes the status of a deactivated system object back to active.

Syntax

```
public void activateSystemObject(SystemObjectPK systemKey)
```

Parameters

Name	Description
systemKey	The system code and local ID of the system object to activate.

Returns
None

Throws
- ProcessingException
- UserException

addSystemObject

Description
This method adds the system object to the enterprise object associated with the specified EUID.

Syntax
```
public void addSystemObject(java.lang.String euid, SystemObject sysobj)
```

Parameters

Name	Description
euid	The EUID of the enterprise object to which you want to add the system object.
sysobj	The system object to add to the enterprise object.

Returns
None

Throws
- ProcessingException
- UserException

calculatePotentialDuplicates

Description
This method calculates potential duplicates for the specified EUID and transaction ID.

Syntax
```
public void calculatePotentialDuplicates(java.lang.String euid, java.lang.String
transID)
```

Parameters

Name	Description
euid	The EUID for which potential duplicates are calculated.
transID	The transaction number for which potential duplicates are calculated.

Returns
None

Throws

- ProcessingException

calculateSBR

Description

This method calculates a new single best record (SBR) for an enterprise object that is updated.

Syntax

```
public SBR calculateSBR(EnterpriseObject eo)
```

Parameters

Name	Description
eo	The enterprise object whose SBR is recalculated.

Returns

The recalculated SBR of the enterprise object.

Throws

- ProcessingException
- UserException

countAssumedMatches

Description

This method counts the number of assumed match records matching the date criteria specified in search object.

Syntax

```
public int countAssumedMatches(AssumedMatchSearchObject amso)
```

Parameters

None

Returns

Count of the assumed match records matching the search criteria.

Throws

- ProcessingException
- UserException

countPotentialDuplicates

Description

This method counts the number of potential duplicate records matching the criteria specified in search object.

Syntax

```
public int countPotentialDuplicates(PotentialDuplicateSearchObject pdso)
```

Parameters

None

Returns

Count of the potential duplicate records matching the search criteria.

Throws

- ProcessingException
- UserException

createEnterpriseObject

Description

This method creates a new enterprise object to add to the master index database using the information in the specified system object.

Syntax

```
public EnterpriseObject createEnterpriseObject(SystemObject sysobj)
```

Parameters

Name	Description
sysobj	The system object to use as a basis for the enterprise object.

Returns

The enterprise object created from the specified system object.

Throws

- ProcessingException
- UserException

deactivateEnterpriseObject

Description

This method changes the status of an enterprise object from active to inactive and deletes all potential duplicate listings for that object.

Syntax

```
public void deactivateEnterpriseObject(java.lang.String euid)
```

Parameters

Name	Description
euid	The EUID associated with the enterprise object to deactivate.

Returns
None

Throws
- ProcessingException
- UserException

deactivateSystemObject

Description
This method changes the status of an active system object to inactive.

Syntax
```
public void deactivateSystemObject(SystemObjectPK systemKey)
```

Parameters

Name	Description
systemKey	The system code and local ID of the system object to deactivate.

Returns
None

Throws
- ProcessingException
- UserException

deduplicateSystemObject

Description
This method performs a query against the MPI database using search options specified in DeduplicationOption and attributes of system object. If a match is found, it performs an update using the system object and returns the updated enterprise object. If no match is found, a new enterprise object is created.

It always matches the entity with the highest match score for BLOCKER-SEARCH. The range search is not supported.

The deduplicateSystemObject API supports all the functionalities of block queries, matching policies except MostRecentReceived, survivorship rules, and update policies.

> **Note:** To merge and unmerge enterprise objects which contain the system-assigned local Ids, see Deduplication Merge Policy and Deduplication Unmerge Policy in *Oracle Healthcare Master Person Index User's Guide*.

Syntax
```
public DeduplicationResult deduplicateSystemObject(SystemObject so,
DeduplicationOption option) throws ProcessingException, UserException;
```

Parameters

Name	Description
SystemObject	The source system object with system code, and with or without the local ID assigned.
DeduplicationOption	Options for the method. ■ **searchId**: The name of the query to be used. This release supports only BLOCKER-SERACH. ■ **matchThreshold**: Match threshold is used for determining the match. That can be above or lower the assumed match threshold. It provides dynamic flexibility. The default value is 0.0. When you pass the default value, it uses assumed match threshold configured in master.xml. ■ **transactionOption**: Valid options are TRANSACTION_LOG_DELTA_ENABLED and TRANSACTION_LOG_DELTA_DISABLED. The default value is TRANSACTION_LOG_DELTA_ENABLED. ■ **weighted**: It will always be 'weighted' even if you set the value as false.

Returns

DeduplicationResult that includes:

- **Euid**: The euid of the enterprise object created or updated.
- **EnterpriseObject**: The enterprise object created or updated.
- **resultCode**: The status of the deduplication result. The valid values are MATCH_UPDATE, MATCH_NO_CHANGE, NEW_EO, and SYS_ID_MATCH.
- **matchScore**: The match score if a match is found.
- **transactionId**: The transaction ID.
- **potentialDuplicates**: A list of potential duplicates if any potential duplicate is found.

Throws

- ProcessingException
- UserException

deleteSystemObject

Description

This method permanently deletes a system object from its associated enterprise object.

Syntax

```
public void deleteSystemObject(SystemObjectPK systemKey)
```

Parameters

Name	Description
systemKey	The system code and local ID of the system object to delete.

Returns

None

Throws

- ProcessingException
- UserException

executeMatch

`executeMatch` is one of four methods that process a system object based on the configuration defined for the Master Index Manager Service and associated runtime components. This method searches for possible matches in the database, and if it finds a match, it replaces the system object (instead of updating the system object, which is what `executeMatchUpdate` does).

The following runtime components configure `executeMatch`.

- The Query Builder defines the blocking queries used for matching.
- Threshold configuration specifies which blocking query to use and specifies matching parameters, including duplicate and match thresholds and whether potential duplicates are automatically recalculated for updated records.
- The pass controller and block picker classes specify how the blocking query is executed.

Syntax

```
public MatchResult executeMatch(SystemObject sysObj)
```

Parameters

Name	Description
sysObj	The system object to process into the database.

Returns

A match result object containing the results of the matching process.

Throws

- ProcessingException
- UserException

executeMatchDupRecalc

`executeMatchDupRecalc` is one of four methods that process a system object based on the configuration defined for the Master Index Manager Service and associated runtime components. It is configured by the same components as `executeMatch`, and is similar to `executeMatch` but allows you to control whether potential duplicates are recalculated when an object is updated regardless of whether the update mode is set to optimistic or pessimistic.

`executeMatchDupRecalc` differs from `executeMatch` in two ways:

- It provides an override flag for the update mode specified in the Threshold configuration file.

- The match result object returned by this method includes an indicator of whether match fields were modified. This helps you determine whether you need to process potential duplicates for this object at a later time.

> **Note:** To process potential duplicates at a later time, call `calculatePotentialDuplicates`.

Syntax

```
public MatchResult executeMatchDupRecalc(SystemObject sysObj, java.lang.Boolean
performPessimistic)
```

Parameters

Name	Description
sysObj	The system object to process into the database.
performPessimistic	A Boolean indicator of whether to recalculate potential duplicates on update or to defer it until later. Specify **true** to recalculate on update and **false** to defer the recalculation.

Returns

A match result object containing the results of the matching process.

Throws

- ProcessingException
- UserException

executeMatchGui

`executeMatchGui` is identical to `executeMatch`, but it is only called by the Enterprise Data Manager. It processes a system object based on the configuration defined for the Master Index Manager Service and associated runtime components. This method searches for possible matches in the database, and if it finds a match, it replaces the system object (instead of updating the system object).

The following runtime components configure `executeMatchGui`:

- The Query Builder defines the blocking queries used for matching.
- Threshold configuration specifies which blocking query to use and specifies matching parameters, including duplicate and match thresholds.
- The pass controller and block picker classes specify how the blocking query is executed.

Syntax

```
public MatchResult executeMatchGui(SystemObject sysObj)
```

Parameters

Name	Description
sysObj	The system object to process into the database.

Returns

A match result object containing the results of the matching process.

Throws

- ProcessingException
- UserException

executeMatchUpdate

`executeMatchUpdate` is one of four methods that process a system object based on the configuration defined for the Master Index Manager Service and associated runtime components. It is configured by the same components as `executeMatch`. This method searches for possible matches in the database, and if it finds a match, it updates the system object (instead of replacing the system object, which is what `executeMatch` does).

The following runtime components configure executeMatchUpdate.

- The Query Builder defines the blocking queries used for matching.
- Threshold configuration specifies which blocking query to use and specifies matching parameters, including duplicate and match thresholds.
- The pass controller and block picker classes specify how the blocking query is executed.

Syntax

```
public MatchResult executeMatchUpdate(SystemObject sysObj)
```

Parameters

Name	Description
sysObj	The system object to process into the database.

Returns

A match result object containing the results of the matching process.

Throws

- ProcessingException
- UserException

executeMatchUpdateDupRecalc

Description

`executeMatchUpdateDupRecalc` is one of four methods that process a system object based on the configuration defined for the Master Index Manager Service and associated runtime components. It is configured by the same components as `executeMatch`, and is similar to `executeMatchUpdate` but allows you to control whether potential duplicates are recalculated when an object is updated regardless of whether the update mode is set to optimistic or pessimistic.

`executeMatchUpdateDupRecalc` differs from `executeMatchUpdate` in two ways:

- It provides an override flag for the update mode specified in the Threshold configuration file.
- The match result object returned by this method includes an indicator of whether match fields were modified. This helps you determine whether you need to process potential duplicates for this object at a later time.

Syntax

```
public MatchResult executeMatchUpdateDupRecalc(SystemObject sysObj,
java.lang.Boolean performPessimistic)
```

> **Note:** To process potential duplicates at a later time, call `calculatePotentialDuplicates`.

Parameters

Name	Description
sysObj	The system object to process into the database.
performPessimistic	A Boolean indicator of whether to recalculate potential duplicates on update or to defer it until later. Specify **true** to recalculate on update and **false** to defer the recalculation.

Returns

A match result object containing the results of the matching process.

Throws

- ProcessingException
- UserException

getAssumedMatchThreshold

Description

This method retrieves the Assumed Match threshold.

Syntax

```
public float getAssumedMatchThreshold()
```

getConfigurationValue

Description

This method retrieves the configuration for a master controller parameter, such as the EUID length, duplicate threshold, or match threshold.

Syntax

```
public java.lang.Object getConfigurationValue(java.lang.String param)
```

Parameters

Name	Description
param	This parameter is defined in the Threshold configuration file of the Master Index Project.

Returns
An object containing the value of the specified parameter.

Throws
- ProcessingException
- UserException

getDatabaseStatus

Description
This method retrieves the status of the master index database.

Syntax
```
public java.lang.String getDatabaseStatus()
```

Parameters
None

Returns
The status of the master index database.

Throws
- ProcessingException
- UserException

getDuplicateThreshold

Description
This method retrieves the potential duplicate threshold.

Syntax
```
public float getDuplicateThreshold()
```

getEnterpriseObject

Description
This method returns the enterprise object associated with the specified EUID.

Syntax
```
public EnterpriseObject getEnterpriseObject(java.lang.String euid)
```

Parameters

Name	Description
euid	The EUID of the enterprise object you want to retrieve.
options	A list of ePaths that define which types of objects to retrieve to create the resulting EnterpriseObject.
key	The system object primary key.

Returns

The enterprise object associated with the specified EUID. Returns null if no enterprise object with the specified EUID is found.

Throws

- ProcessingException
- UserException

getEUID

Description

This method returns the EUID associated with the system code and local ID specified in the SystemObjectPK object.

Syntax

```
public java.lang.String getEUID(SystemObjectPK key)
```

Parameters

Name	Description
key	The system object key containing the system code and local ID to use as search criteria.

Returns

The EUID associated with the given system object key. Returns null if no results are found.

Throws

- ProcessingException
- UserException

getLinkValues

Description

This method returns a map with (fieldName, actual value for link) for the given EO.

Syntax

```
public java.util.Map getLinkValues(EnterpriseObject eo)
```

Parameters

Name	Description
eo	The EnterpriseObject that has LINKs.

Returns

resultMap map (fieldName, actual value for link) for the given EO.

Throws

- ObjectException
- ConnectionInvalidException
- OPSException
- ProcessingException

getMergeHistory

Description

This method retrieves the history of the merge transactions associated with the specified EUID.

Syntax

```
public MergeHistoryNode getMergeHistory(java.lang.String euid)
```

Parameters

Name	Description
euid	The EUID associated with the merge history to retrieve.

Returns

The merge history tree for the specified EUID.

Throws

- ProcessingException
- UserException

getRevisionNumber

Description

This method retrieves the SBR revision number for the specified EUID.

Syntax

```
public java.lang.Integer getRevisionNumber(java.lang.String euid)
```

Parameters

Name	Description
euid	The EUID containing the SBR revision number to retrieve.

Returns

The revision number for the SBR.

Throws

ProcessingException

getSBR

Description

This method returns the single best record (SBR) object associated with the specified EUID.

Syntax

```
public SBR getSBR(java.lang.String euid)
```

Parameters

Name	Description
euid	The EUID of the enterprise object whose SBR you want to retrieve.

Returns

The SBR object associated with the specified EUID. Returns null if no SBR associated with the specified EUID is found.

Throws

- ProcessingException
- UserException

getSystemObject

Description

This method returns the system object associated with the system code and local ID contained in the specified SystemObjectPK object.

Syntax

```
public SystemObject getSystemObject(SystemObjectPK key)
```

Parameters

Name	Description
key	The system object key containing the local ID and system code to lookup.

Returns
The system object associated with the specified local ID and system code.

Throws
- ProcessingException
- UserException

insertAuditLog

Description
This method inserts an audit log record of a transaction into the database.

Syntax
```
public void insertAuditLog(AuditDataObject auditObject)
```

Parameters

Name	Description
auditObject	The audit log record to insert.

Returns
None

Throws
- ProcessingException
- UserException

lookupAssumedMatches

Description
This method returns an iterator of AssumedMatchSummary objects based on the criteria contained in the assumed match search object (AssumedMatchSearchObject class).

Syntax
```
public AssumedMatchIterator lookupAssumedMatches(AssumedMatchSearchObject obj)
```

Parameters

Name	Description
obj	An instance of AssumedMatchSearchObject containing the potential duplicate search criteria.

Returns
An iterator of search results (AssumedMatchSummary objects).

Throws
- ProcessingException

- UserException

lookupAuditLog

Description

This method looks up an audit log record based on the criteria contained in an audit search object.

Syntax

```
public AuditIterator lookupAuditLog(AuditSearchObject obj)
```

Parameters

Name	Description
obj	An instance of AuditSearchObject containing the audit log search criteria.

Returns

An iterator of audit log matches to the given search criteria.

Throws

- ProcessingException
- UserException

lookupPotentialDuplicates

Description

This method returns an iterator of PotentialDuplicateSummary objects based on the criteria contained in the potential duplicate search object (PotentialDuplicateSearchObject class).

Syntax

```
public PotentialDuplicateIterator
lookupPotentialDuplicates(PotentialDuplicateSearchObject obj)
```

Parameters

Name	Description
obj	An instance of PotentialDuplicateSearchObject containing the potential duplicate search criteria.

Returns

An iterator of search results (PotentialDuplicateSummary objects).

Throws

- ProcessingException
- UserException

lookupSystemDefinition

Description

This method retrieves the attributes of an external system from the master index database based on the system code.

Syntax

```
public SystemDefinition lookupSystemDefinition(java.lang.String systemCode)
```

Parameters

Name	Description
systemCode	The system code of an external system.

Returns

A set of system attributes for the system identified the given system code.

Throws

ProcessingException

lookupSystemDefinitions

Description

This method retrieves the attributes of an external system from the master index database, such as the system code, masking flags, local ID format and so on.

Syntax

```
public SystemDefinition[] lookupSystemDefinitions()
```

Parameters

None

Returns

An array of system attributes.

Throws

ProcessingException

lookupSystemObjectPKs

Description

This method returns an array of all system objects associated with the specified EUID.

Syntax

```
public SystemObjectPK[] lookupSystemObjectPKs(java.lang.String euid)
```

Parameters

Name	Description
euid	The EUID of the enterprise object containing the system objects to retrieve.

Returns

An array of system objects associated with the specified EUID.

Throws

- ProcessingException
- UserException

lookupSystemObjects

Description

This method looks up the active system objects associated with the specified EUID.

Syntax

```
public SystemObject[] lookupSystemObjects(java.lang.String euid)
```

Parameters

Name	Description
euid	The EUID whose associated system objects will be retrieved.

Returns

An array of system objects that are associated with the specified EUID.

Throws

- ProcessingException
- UserException

lookupTransaction

Description

This method returns a transaction summary for the transaction associated with the specified transaction number.

Syntax

```
public TransactionSummary lookupTransaction(java.lang.String transId)
```

Parameters

Name	Description
transId	The transaction number for the transaction to look up.

Returns

The transaction summary for the specified transaction ID.

Throws

- ProcessingException
- UserException

lookupTransactions

Description

This method returns an array of transaction summaries based on the search criteria contained in the specified transaction search object.

Syntax

```
public TransactionIterator lookupTransactions(TransactionSearchObject obj)
```

Parameters

Name	Description
obj	The transaction search object containing the search criteria.

Returns

An array of transaction summaries matching the given search criteria.

Throws

- ProcessingException
- UserException

mergeEnterpriseObject

Description

This method merges two enterprise objects based on the specified EUID and enterprise object.

Syntax

```
public MergeResult mergeEnterpriseObject(java.lang.String sourceEUID,
EnterpriseObject destinationEO, boolean calculateOnly)
```

Parameters

Name	Description
sourceEUID	The EUID of the enterprise object that will not survive the merge.
destinationEO	The EUID of the enterprise object that will survive the merge.
calculateOnly	An indicator of whether to commit changes to the database or to simply compute the merge results. Specify **false** to commit the changes.

Returns

The results of the merge operation.

Throws

- ProcessingException
- UserException

mergeMultipleEnterpriseObjects

Description

This method merges multiple enterprise objects based on the specified EUIDs and enterprise object.

Syntax

```
public MergeResult[] mergeMultipleEnterpriseObjects(java.lang.String[]
sourceEUIDs, EnterpriseObject destinationEO, java.lang.String[]
srcRevisionNumbers, java.lang.String destRevisionNumber, boolean calculateOnly)
```

Parameters

Name	Description
sourceEUID	The EUID of the enterprise object that will not survive the merge.
destinationEO	The EUID of the enterprise object that will survive the merge.
srcRevisionNumbers	The SBR revision number of the non-surviving enterprise object.
destRevisionNumber	The SBR revision number of the surviving enterprise object.
calculateOnly	An indicator of whether to commit changes to the database or to simply compute the merge results. Specify **false** to commit the changes.

Returns

The results of the merge operation.

Throws

- ProcessingException
- UserException

mergeSystemObject

Description

Merges two system objects from the specified system. The system objects may belong to a single enterprise object or to two different enterprise objects.

Syntax

```
public MergeResult mergeSystemObject(java.lang.String systemCode, java.lang.String
sourceLID, java.lang.String destLID, boolean calculateOnly)
```

Parameters

Name	Description
systemCode	The processing code of the system to which the two system objects belong.
sourceLID	The local ID of the system object that will not survive the merge.
destLID	The local ID of the system object that will survive the merge.
calculateOnly	An indicator of whether to commit changes to the database or to simply compute the merge results. Specify **false** to commit the changes.

Returns

The results of the merge operation.

Throws

- ProcessingException
- UserException

previewUndoAssumedMatch

Description

This method previews the undo assumed match.

Syntax

```
public EnterpriseObject previewUndoAssumedMatch(java.lang.String assumedMatchId)
```

Parameters

Name	Description
assumedMatchId	Id of assumed match to be resolved

Returns

EUID of new EO

Throws

- ProcessingException
- UserException

resolvePotentialDuplicate

Description

This method flags a potential duplicate pair with Resolved or Auto Resolved status. If resolved, the pair can be marked as potential duplicates again during a future transaction. If auto-resolved, the pair is permanently flagged as resolved.

Syntax

```
public void resolvePotentialDuplicate(java.lang.String id, boolean autoResolve)
```

Parameters

Name	Description
id	The potential duplicate ID of the pair to be resolved.
autoResolve	A Boolean value indicating whether to resolve or auto-resolve the pair. Use **true** for auto-resolve and **false** for resolve.

Returns

None

Throws

- ProcessingException
- UserException

searchEnterpriseObject

Description

This method returns an iterator of enterprise objects that match the specified search criteria and options.

Syntax

```
public EOSearchResultIterator searchEnterpriseObject(EOSearchCriteria criteria,
EOSearchOptions searchOptions)
```

Parameters

Name	Description
criteria	An EOSearchCriteria object containing the search criteria.
searchOptions	An EOSearchOptions object defining attributes of the search.

Returns

An iterator containing the results of the query.

Throws

- ProcessingException
- UserException

transferSystemObject

Description

This method transfers a system object from one enterprise record to another enterprise record.

Syntax

```
public void transferSystemObject(java.lang.String toEUID, SystemObjectPK
systemKey)
```

Parameters

Name	Description
toEUID	The EUID of the enterprise object to which the system object will be transferred.
systemKey	The system code and local ID of the system object to transfer.

Returns

None

Throws

- ProcessingException
- UserException

undoAssumedMatch

Description

This method reverses an assumed match transaction, unmerging the two objects that were matched and creating a new enterprise object for the record that caused the assumed match. Potential duplicates are calculated for the new enterprise object.

Syntax

```
public java.lang.String undoAssumedMatch(java.lang.String assumedMatchId)
```

Parameters

Name	Description
assumedMatchId	The ID of the assumed match transaction to reverse.

Returns

The EUID of the newly created enterprise object resulting from the reversing assumed match.

Throws

- ProcessingException
- UserException

unmergeEnterpriseObject

Description

This method unmerges the two enterprise objects that were involved in the most recent merge transaction for the specified EUID. When the calculateOnly is set to true, the unmerge results is calculated, but the changes are not committed to the database.

Syntax

```
public MergeResult unmergeEnterpriseObject(java.lang.String euid, boolean
calculateOnly)
```

Parameters

Name	Description
euid	The EUID of the enterprise object to be unmerged.
calculateOnly	An indicator of whether to commit the unmerge to the database or to calculate the changes for viewing. Specify **true** to calculate for viewing and **false** to commit to the database.

Returns

The result of the unmerge transaction.

Throws

- ProcessingException
- UserException

unmergeSystemObject

Description

This method unmerges the two system objects that were involved in the most recent merge transaction for the specified local ID. When the calculateOnly is set to true, the unmerge results is computed, but the changes are not committed to the database.

Syntax

```
public MergeResult unmergeSystemObject(java.lang.String systemCode,
java.lang.String sourceLID, java.lang.String destLID, boolean calculateOnly)
```

Parameters

Name	Description
systemCode	The system code of the system object to be unmerged.
sourceLID	The local ID of the non-surviving system object.
destLID	The local ID of the surviving system object.
calculateOnly	An indicator of whether to commit unmerge to the database or to calculate the changes for viewing. Specify true to calculate for viewing; specify false to commit to the database.

Returns

The result of the unmerge transaction.

Throws

- ProcessingException
- UserException

unresolvePotentialDuplicate

Description

This method changes the status of a resolved or auto-resolved potential duplicate record pair back to unresolved and places the records back in the potential duplicate listing.

Syntax

```
public void unresolvePotentialDuplicate(java.lang.String id)
```

Parameters

Name	Description
id	The potential duplicate ID of the records to be unresolved.

Returns

None

Throws

- ProcessingException
- UserException

updateEnterpriseDupRecalc

Description

This method updates the database to reflect the new values of the specified enterprise object. This method is similar to `updateEnterpriseObject` but it allows you to control whether potential duplicate recalculation is performed when an enterprise object is updated, regardless of whether the update mode is set to optimistic or pessimistic.

This method differs from `updateEnterpriseObject` in two ways:

- It provides an override flag for the update mode specified in the Threshold configuration file.
- The update result object returned by this method includes an indicator of whether match fields were modified. This helps you determine whether you need to process potential duplicates for this object at a later time.

> **Note:** To process potential duplicates at a later time, call `calculatePotentialDuplicates`.

Syntax

```
public UpdateResult updateEnterpriseDupRecalc(EnterpriseObject eo,
java.lang.Boolean performPessimistic)
```

Parameters

Name	Description
eo	The enterprise object (EnterpriseObject class) to be updated.

Name	Description
performPessimistic	A Boolean indicator of whether to defer potential duplicate processing. Specify **true** to recalculate potential duplicates on update and **false** to defer recalculation to a later time.

Returns

The UpdateResult object created from the update.

Throws

- ProcessingException
- UserException

updateEnterpriseObject

Description

This method updates the database to reflect the new values of the specified enterprise object. If the enterprise object is deactivated during the update, potential duplicates are deleted for that object. If the enterprise object is still active, was changed during the transaction, and the update mode is set to pessimistic, the application checks whether any key fields were updated in the SBR of the enterprise object. If key fields were updated, potential duplicates are recalculated.

Syntax

```
public void updateEnterpriseObject(EnterpriseObject eo)
```

Parameters

Name	Description
eo	The enterprise object (EnterpriseObject class) to be updated.

Returns

None

Throws

- ProcessingException
- UserException

updateSBR

Description

This method updates SBR by collecting the values from MAP to the SBR that specified by EUID.

Syntax

```
public EnterpriseObject updateSBR(java.util.Map mapSystems, EnterpriseObject eo,
boolean removalFlag)
```

Parameters

Name	Description
mapSystems	The Map consists of epath as key and System as value from which the filed should take for updating SBR.

Returns

None

Throws

- ProcessingException
- UserException

updateSystemObject

Description

This method updates the existing system object in the database with the given system object.

Syntax

```
public void updateSystemObject(SystemObject sysobj)
```

Parameters

Name	Description
sysobj	The updated system object.

Returns

None

Throws

- ProcessingException
- UserException

5

Inbound Message Processing with Custom Logic

You can customize the way the execute match methods process inbound messages by defining custom plug-ins that include decision-point methods. This chapter describes the standard inbound processing logic as described in "Inbound Message Processing Logic" on page 2-6, including how the decision-point methods alter the process.

This chapter includes the following section:

Custom Decision Point Logic

There are several decision points in the match process that can be defined by custom logic using custom plug-ins. The steps below are identical to those outlined in "Inbound Message Processing Logic" on page 2-6, but include descriptions of the decision points, which are listed in italic font. If no custom logic is defined, the decisions default to **false**, and processing is identical to that described in "Inbound Message Processing Logic" on page 2-6.

For more information about the methods and plug-ins, see the *Oracle Healthcare Master Person Index User's Guide*. For detailed information about the methods, see the Javadocs provided with Oracle Healthcare Master Person Index. The methods are contained in the ExecuteMatchLogics class in the package `com.sun.mdm.index.master`.

1. When a message is received by the master person index application, a search is performed for any existing records with the same local ID and system as those contained in the message. This search only includes records with a status of **A**, meaning only active records are included. If a matching record is found, an existing EUID is returned.
2. If an existing record is found with the same system and local ID as the incoming message, it is assumed that the two records represent the same entity. Using the EUID of the existing record, the OHMPI application performs an update of the record's information in the database.

   Custom plug-in decision point: If `disallowUpdate` is set to **true**, the update is not allowed and a `MatchResult` object is returned with a result code of **12**. If `disallowUpdate` is set to **false** and `rejectUpdate` is set to **true**, the update is not allowed and a `MatchResult` object is returned with a result code of **13**.

   - If the update does not make any changes to the object's information, no further processing is required and the existing EUID is returned.

- If there are changes to the object's information, the updated record is inserted into database, and the changes are recorded in the sbyn_transaction table.
- If there are changes to key fields (that is, fields used for matching or for the blocking query) and the update mode is set to pessimistic, potential duplicates are reevaluated for the updated record.

3. If no records are found that match the record's system and local identifier, a second search is performed using the blocking query. A search is performed on each of the defined query blocks to retrieve a candidate pool of potential matches.

   Custom plug-in decision point: If `bypassMatching` is set to **true**, the search steps are bypassed and, if `disallowAdd` is set to **false**, a new record is added. If `disallowAdd` is set to **true**, the record is not added and a `MatchResult` object is returned with a result code of **11**.

   Each record returned from the search is weighted using the fields defined for matching in the inbound message.

4. After the search is performed, the number of resulting records is calculated.
   - If a record or records are returned from the search with a matching probability weight above the match threshold, the OHMPI application performs exact match processing (see Step 5).
   - If no matching records are found, the inbound message is treated as a new record. A new EUID is generated and a new record is inserted into the database.

5. If records were found within the high match probability range, exact match processing is performed as follows:
   - If only one record is returned from this search with a matching probability that is equal to or greater than the match threshold, additional checking is performed to verify whether the records originated from the same system (see Step 6).
   - If more than one record is returned with a matching probability that is equal to or greater than the match threshold and exact matching is set to **false**, then the record with the highest matching probability is checked against the incoming message to see if they originated from the same system (see Step 6).
   - If more than one record is returned with a matching probability that is equal to or greater than the match threshold and exact matching is **true**, a new EUID is generated and a new record is inserted into the database.

     Custom plug-in decision point: If `disallowAdd` is set to **true**, the new record is not inserted and a `MatchResult` object is returned with a result code of **11**.
   - If no record is returned from the database search, or if none of the matching records have a weight in the exact match range, a new EUID is generated and a new record is inserted into the database.

     Custom plug-in decision point: If `disallowAdd` is set to **true**, the new record is not inserted and a `MatchResult` object is returned with a result code of **11**.

6. When records are checked for same system entries, the OHMPI application tries to retrieve an existing local ID using the system of the new record and the EUID of the record that has the highest match weight.
   - If a local ID is found and same system matching is set to **true**, a new record is inserted and the two records are considered to be potential duplicates. These records are marked as same system potential duplicates.

Custom plug-in decision point: If `disallowAdd` is set to **true**, the new record is not inserted and a `MatchResult` object is returned with a result code of **11**.

- If a local ID is found and same system matching is set to **false**, it is assumed that the two records represent the same entity. Using the EUID of the existing record, the OHMPI application performs an update, following the process described in Step 2 earlier.

  Custom plug-in decision point: If `rejectAssumedMatch` is set to **true** and `disallowAdd` is set to **false**, a new record is added; if `disallowAdd` is set to **true**, the new record is not inserted and a `MatchResult` object is returned with a result code of **11**. If `rejectAssumedMatch` and `disallowUpdate` are set to **false**, the existing record is updated; if `disallowUpdate` is set to **true**, the update is not performed and a `MatchResult` object is returned with a result code of **13**.

- If no local ID is found, it is assumed that the two records represent the same entity and an assumed match occurs. Using the EUID of the existing record, the master person index application performs an update, following the process described in Step 2 earlier.

  Custom plug-in decision point: If `rejectAssumedMatch` is set to **true** and `disallowAdd` is set to **false**, a new record is added; if `disallowAdd` is set to **true**, the new record is not inserted and a `MatchResult` object is returned with a result code of **11**. If `rejectAssumedMatch` and `disallowUpdate` are set to **false**, the existing record is updated; if `disallowUpdate` is set to **true**, the update is not performed and a `MatchResult` object is returned with a result code of **13**.

7. If a new record is inserted, all records that were returned from the blocking query are weighed against the new record using the matching algorithm. If a record is updated and the update mode is pessimistic, the same occurs for the updated record. If the matching probability weight of a record is greater than or equal to the potential duplicate threshold, the record is flagged as a potential duplicate. For more information about thresholds, see *Oracle Healthcare Master Person Index Configuration Reference*.

6

Match Types and Field Names

When you create an Oracle Healthcare Master Person Index (OHMPI) application, you can select a Match Type for each field in the OHMPI wizard. Each match type defines a different type of standardization, normalization, phonetic encoding, and matching logic in the `mefa.xml` file. The following sections describe each match type you can specify in the wizard and how each affects the logic in the `mefa.xml` file.

This chapter includes the following sections:

Match Types and Standardization Types

For each field that will be used for matching in the OHMPI application, you can select a match type in the wizard. When you select a match type for a field, Oracle Healthcare Master Person Index automatically adds that field to the match string in the `mefa.xml` file and, in many cases, generates additional fields in the object structure that are not visible on the wizard. These fields are used for searching and matching and they should not be modified.

If new fields are generated, they are automatically incorporated into the configuration files and the database script that creates the master person index tables. These fields store standardized, normalized, or phonetic versions of the field, depending on the type of matching you choose. In addition, these fields are assigned a match type in the match string in the `mefa.xml` file. They might also be defined for standardization in the `mefa.xml` file, in which case they will also be assigned a standardization type. The types described in the next section pertain to the OHMPI Match Engine only.

Note: The match types specified in the `mefa.xml` file for the fields in the match string are not always the same as the match types you specify in the wizard. The match types you can specify from the Configuration Editor more closely match those that are shown in the `mefs.xml` file. This discussion pertains more to the match types on the wizard. For more information, see *Oracle Healthcare Master Person Index Match Engine Reference*.

Oracle Match Engine Match Types

The Oracle Healthcare Master Person Index wizard match types for the OHMPI Match Engine fall into four primary categories.

- "BusinessName Match Types"
- "Address Match Types"
- "Miscellaneous Match Types"

The actual standardization and match types entered into the `mefa.xml` file vary for each match type you select in the wizard. The match and standardization types for each type of field are listed in the following descriptions. The match types entered into the `mefa.xml` file correspond to the match types defined in the match configuration file, `MatchConfigFile.cfg`.

Person Match Types

The Person match types include PersonLastName and PersonFirstName. These match types are used to normalize and phonetically encode name fields for person matching. For each field with one of these match types, the wizard adds two fields to the object structure for phonetic and standardized versions. If you specify a field with a person match type for blocking in the wizard, the phonetic version of the name is automatically added to the blocking query. The following fields are created when you specify one of the Person match types for a field (*field_name* refers to the name of the field specified for Person matching).

- *field_name*_Std - This field contains the normalized version of the name.
- *field_name*_Phon - This field contains the phonetic version of the name.

The corresponding standardization and match types in the `mefa.xml` file are listed in Table 6–1, " Person Name Standardization and Match Types".

Table 6–1 Person Name Standardization and Match Types

MPI Wizard Match Type	Match Field File Standardization Type	mefa.xml Match Type
PersonLastName	PersonName	LastName
PersonFirstName	PersonName	FirstName

BusinessName Match Types

The BusinessName match type is designed to help parse, normalize, and phonetically encode a business name. BusinessName matching adds several fields to the object structure and to the match string. If you specify a business name field for blocking, each parsed business name field is added to the blocking query. The corresponding standardization type in the `mefa.xml` file for all fields selected for BusinessName matching is also BusinessName. The actual match type assigned to each field varies depending on the type of information in each field.

Table 6–2, " BusinessName Match Types" lists the fields created when you select the BusinessName match type for a field along with their corresponding match types in the `mefa.xml` file (*field_name* refers to the name of the field selected for BusinessName matching).

Note: Only specify this type of matching for one business name field; otherwise, the wizard will create duplicate entries in the object structure. If more than one field contains the business name, you can add those fields to the standardization structure in the `mefa.xml` file after the wizard creates the configuration files.

Table 6–2 BusinessName Match Types

Field Name	Description	Added to Match String	mefa.xml Match Type
*field_name*_Name	The parsed and normalized version of the business name.	Yes	PrimaryName
*field_name*_NamePhon	The phonetic version of the business name.	No	
*field_name*_OrgType	The parsed organization type of the business name.	Yes	OrgTypeKeyword
*field_name*_AssocType	The association type for the business.	Yes	AssocTypeKeyword
*field_name*_Industry	The name of the industry for the business.	Yes	IndustryTypeKeyword
*field_name*_Sector	The name of the industry sector (industries are a subset of sectors).	Yes	IndustrySectorList
*field_name*_Alias	An alias for the business name.	No	
*field_name*_Url	The web site URL for the business.	Yes	Url

Address Match Types

The Address match type is designed to help parse, normalize, and phonetically encode an address for matching or standardizing address information. Address matching adds several fields to the object structure and to the match string. If you specify an address field for blocking, the parsed fields are added to the blocking query. The corresponding standardization type for fields selected for Address matching is Address. The actual match type assigned to each field varies depending on the type of information in each field.

The fields created when you select the Address match type for a field are listed in Table 6–3, " Address Match Types", along with their corresponding match types in the `mefa.xml` file (*field_name* refers to the name of the field selected for Address matching).

> **Note:** Only specify this type of matching for one street address field; otherwise, the wizard will create duplicate entries in the object structure. If more than one field contains the street address, you can define the additional fields in the standardization structure in the `mefa.xml` file after the wizard creates the configuration files.

Table 6–3 Address Match Types

Field Name	Description	Added to Match String?	mefa.xml Match Type
*field_name*_HouseNo	The parsed street number of the address.	Yes	HouseNumber
*field_name*_StDir	The parsed and normalized street direction of the address.	Yes	StreetDir
*field_name*_StName	The parsed and normalized street name of the address.	Yes	StreetName

Table 6–3 (Cont.) Address Match Types

Field Name	Description	Added to Match String?	mefa.xml Match Type
*field_name*_StPhon	The phonetic version of the street name.	No	
*field_name*_StType	The parsed and normalized street type of the address, such as Boulevard, Street, Drive, and so on.	Yes	StreetType

If you want to search on street addresses but do not want to use these fields for matching, select the Address match type for only one street address field in the wizard. When the wizard is complete, you can remove the address fields from the match string in the `mefa.xml` file.

Miscellaneous Match Types

Several additional match types are defined in the wizard for the OHMPI Match Engine. These match types are used to indicate matching on string, date, or number fields other than those described above or to indicate matching on a field that contains a single character (such as the gender field, which might accept **F** for female or **M** for male). These match types do not define standardization for the specified field and do not add any fields to the object structure. If you specify one of these match types for a field in the wizard, the field is added to the match string with a match type of String, Date, Number, or Char.

7

Web Services

This chapter lists the web services that are available with this release of Oracle Healthcare Master Person Index.

Available Web Services

The following sections provide a list of the available web services, including a description of the action each one performs.

- "Oracle Healthcare Master Person Index Web Services"
- "IHE Standard Web Services"

Oracle Healthcare Master Person Index Web Services

The OHMPI web service is available for use in the client application. The WSDL of the web service is generated by the application server when the OHMPI application is deployed.

The WSDL file is accessible at:

- GlassFish Application Server:
  http://HOST:PORT/APPLICATION-NAMEEJBService/APPLICATION-NAMEEJB?wsdl

  For example, http://localhost:8080/PersonEJBService/PersonEJB?wsdl
- WebLogic Application Server:
  http://HOST:PORT/APPLICATION-NAMEEJB/APPLICATION-NAMEEJBService?WSDL

  For example, http://localhost:7001/PersonEJB/PersonEJBService?WSDL

The xsd for the WSDL is accessible at:

- GlassFish Application Server:
  http://HOST:PORT/APPLICATION-NAMEEJBService/APPLICATION-NAMEEJB?xsd=1

  For example, http://localhost:8080/PersonEJBService/PersonEJB?xsd=1
- WebLogic Application Server:
  http://HOST:PORT/APPLICATION-NAMEEJB/APPLICATION-NAMEEJBService?xsd=1

  For example, http://localhost:7001/PersonEJB/PersonEJBService?xsd=1

For more information on the available web service operations and message, see the WSDL file and XSD file.

Web Service Operation Descriptions

- "updateEnterpriseRecord"
- "updateSystemRecord"

activateEnterpriseRecord

Description
This method changes the status of a deactivated enterprise object back to active.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object to activate.

Output Message
None

Fault Message
- ProcessingException
- UserException

activateSystemRecord

Description
This method changes the status of a deactivated system object back to active.

Input Message

Name	Type	Description
systemCode	String	The processing code of the system associated with the system record to be activated.
localID	String	The local identifier associated with the system record to be activated.

Output Message
None

Fault Message
- ProcessingException
- UserException

addSystemRecord

Description
This method adds the system object to the enterprise object associated with the specified EUID.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object to which you want to add the system object.
sysObjBean	System*ObjectName*	The Bean for the system object to be added to the enterprise object.

Output Message

None

Fault Message

- ProcessingException
- UserException

deactivateEnterpriseRecord

Description

This method changes the status of an active enterprise object to inactive.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object to deactivate.

Output Message

None

Fault Message

- ProcessingException
- UserException

deactivateSystemRecord

Description

This method changes the status of an active system object to inactive.

Input Message

Name	Type	Description
systemCode	String	The system code of the system object to deactivate.
localid	String	The local ID of the system object to deactivate.

Output Message

None

Fault Message

- ProcessingException
- UserException

deduplicateSystemRecord

Description

This method performs a query against the MPI database using search options specified in DeduplicationOption and attributes of system object. If a match is found, it performs an update using the system object and returns the updated enterprise object. If no match is found, a new enterprise object is created.

It always matches the entity with the highest match score for BLOCKER-SEARCH. The range search is not supported.

The deduplicateSystemRecord API supports all the functionalities of block queries, matching policies except MostRecentReceived, survivorship rules, and update policies.

Input Message

Name	Type	Description
System {primary object name}	SystemObjectName	The source system object bean with system code, and with or without the local ID assigned.
DeduplicationOption Bean	ObjectNameBean	Options for the method. ■ **searchId**: The name of the query to be used. This release supports only BLOCKER-SERACH. ■ **matchThreshold**: Match threshold is used for determining the match. That can be above or lower the assumed match threshold. It provides dynamic flexibility. The default value is 0.0. When you pass the default value, it uses assumed match threshold configured in master.xml. ■ **transactionOption**: Valid options are TRANSACTION_LOG_DELTA_ENABLED and TRANSACTION_LOG_DELTA_ DISABLED. The default value is TRANSACTION_LOG_DELTA_ ENABLED. ■ **weighted**: It will always be 'weighted' even if you set the value as false.

Output Message

DeduplicationResultBean that includes:

- **Euid**: The euid of the enterprise object created or updated.
- **EnterpriseObject**: The enterprise object created or updated.
- **resultCode**: The status of the deduplication result. The valid values are MATCH_ UPDATE, MATCH_NO_CHANGE, NEW_EO, and SYS_ID_MATCH.
- **matchScore**: The match score if a match is found.
- **transactionId**: The transaction ID.

- **potentialDuplicates**: A list of potential duplicates if any potential duplicate is found.

Fault Message

- ProcessingException
- UserException

deleteSystemRecord

Description

This method permanently deletes a system object from its associated enterprise object.

Input Message

Name	Type	Description
systemCode	String	The system code of the system object to be deleted.
localid	String	The local ID of the system object to be deleted.

Output Message

None

Fault Message

- ProcessingException
- UserException

executeMatch

Description

`executeMatch` is one of two methods you can call to process an incoming system object based on the configuration defined for the matching and associated runtime components (the second method is "executeMatchUpdate"). This process searches for possible matches in the database and contains the logic to add a new record or update existing records in the database. One of the two execute match methods should be used for inserting or updating a record in the database.

The following runtime components configure `executeMatch`.

- The Query Builder defines the blocking queries used for matching.
- The Threshold file (`master.xml`) specifies which blocking query to use and specifies matching parameters, including duplicate and match thresholds.
- The pass controller and block picker classes specify how the blocking query is executed.

> **Note:** If `executeMatch` determines that an existing system record will be updated by the incoming record, it replaces the entire existing record with the information in the new record. This could result in loss of data; for example, if the incoming record does not include all address information, existing address information could be lost. To avoid this, use the `executeMatchUpdate` method instead.

Input Message

Name	Type	Description
sysObjBean	System*ObjectName*	The Bean for the system object to be added to or updated in the enterprise object.

Output Message

A match result object containing the results of the matching process.

Fault Message

- ProcessingException
- UserException

executeMatchUpdate

Description

Like "executeMatch", `executeMatchUpdate` processes the system object based on the configuration defined for the matching and associated runtime components. It is configured by the same runtime components as `executeMatch`. One of these two execute match methods should be used for inserting or updating a record in the database.

The primary difference between these two methods is that when `executeMatchUpdate` finds that an incoming record matches an existing record, only the changed data is updated. With `executeMatch`, the entire existing record would be replaced by the incoming record. The `executeMatchUpdate` method differs from `executeMatch` in the following ways:

- If a partial record is received, `executeMatchUpdate` only updates fields whose values are different in the incoming record. Unless the `clearFieldIndicator` field is used, empty or null fields in the incoming record do not update existing values.
- The `clearFieldIndicator` field can be used to null out specific fields.
- Child objects in the existing record are not deleted if they are not present in the incoming record.
- Child objects in the existing record are updated if the same key field value is found in both the incoming and existing records.
- To allow a child object to be removed from the parent object when using `executeMatchUpdate`, a new "delete" method is added to each child object bean.

Input Message

Name	Type	Description
sysObjBean	System*ObjectName*	The Bean for the system object to be added to or updated in the enterprise object.

Output Message

A match result object containing the results of the matching process.

Fault Message

- ProcessingException

- UserException

getEnterpriseRecordByEUID

Description

This method returns the enterprise object associated with the specified EUID.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object you want to retrieve.

Output Message

An enterprise object associated with the specified EUID or null if the enterprise object is not found.

Fault Message

- ProcessingException
- UserException

getEnterpriseRecordByLID

Description

This method returns the enterprise object associated with the specified system code and local ID pair.

Input Message

Name	Type	Description
systemCode	String	The system code of a system associated with the enterprise object to find.
localid	String	A local ID associated with the specified system.

Output Message

An enterprise object or null if the enterprise object is not found.

Fault Message

- ProcessingException
- UserException

getEUID

Description

This method returns the EUID of the enterprise object associated with the specified system code and local ID.

Input Message

Name	Type	Description
systemCode	String	A known system code for the enterprise object.
localid	String	The local ID corresponding with the given system.

Output Message

A string containing an EUID or null if the EUID is not found.

Fault Message

- ProcessingException
- UserException

getLIDs

Description

This method retrieves the local ID and system pairs associated with the given EUID.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object whose local ID and system pairs you want to retrieve.

Output Message

An array of system object keys (System*ObjectName*PK objects) or null if no results are found.

Fault Message

- ProcessingException
- UserException

getLIDsByStatus

Description

This method retrieves the local ID and system pairs that are of the specified status and that are associated with the given EUID.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object whose local ID and system pairs to retrieve.
status	String	The status of the local ID and system pairs to retrieve.

Output Message

An array of system object keys (System*ObjectName*PK objects) or null if no system object keys are found.

Fault Message

- ProcessingException
- UserException

getMergeHistory

Description

This method retrieves the history of the merge transactions associated with the specified EUID.

Input Message

Name	Type	Description
euid	String	The EUID associated with the merge history to retrieve.

Output Message

A mergeHistoryNode object containing the merge history retrieved.

Fault Message

- ProcessingException
- UserException

getSBR

Description

This method retrieves the single best record (SBR) associated with the specified EUID.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object whose SBR you want to retrieve.

Output Message

An SBR object or null if no SBR associated with the specified EUID is found.

Fault Message

- ProcessingException
- UserException

getSystemRecord

Description

This method retrieves the system object associated with the given system code and local ID pair.

Input Message

Name	Type	Description
systemCode	String	The system code of the system object to retrieve.
localid	String	The local ID of the system object to retrieve.

Output Message

A system object containing the results of the search or null if no system objects are found.

Fault Message

- ProcessingException
- UserException

getSystemRecordsByEUID

Description

This method returns the active system objects associated with the specified EUID.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object whose system objects you want to retrieve.

Output Message

An array of system objects associated with the specified EUID.

Fault Message

- ProcessingException
- UserException

getSystemRecordsByEUIDStatus

Description

This method returns the system objects of the specified status that are associated with the given EUID.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object whose system objects you want to retrieve.

Name	Type	Description
status	String	The status of the system objects you want to retrieve.

Output Message

An array of system objects associated with the specified EUID and status, or null if no system objects are found.

Fault Message

- ProcessingException
- UserException

lookupAssumedMatches

Description

This method returns an array of assumed match result records based on the criteria contained in the assumed match search object.

Input Message

Name	Type	Description
amsoBean	AssumedMatchSearchObjectBean	An object containing the criteria for an assumed match search.

Output Message

An array of AssumedMatchResult objects representing the matches to an assumed match search.

Fault Message

- ProcessingException
- UserException

lookupLIDs

Description

This method first looks up the EUID associated with the specified source system and source local ID. It then retrieves the local ID and system pairs of the specified status that are associated with that EUID and are from the specified destination system.

> **Note:** Both systems must be of the specified status or an error will occur.

Input Message

Name	Type	Description
sourceSystemCode	String	The system code of the known system and local ID pair.

Name	Type	Description
sourceLID	String	The local ID of the known system and local ID pair.
destSystemCode	String	The system from which the local ID and system pairs to retrieve originated.
status	String	The status of the local ID and system pairs to retrieve.

Output Message

An array of system object keys (System*ObjectName*PK objects).

Fault Message

- ProcessingException
- UserException

lookupPotentialDuplicates

Description

This method returns an array of potential duplicate result records based on the criteria contained in the potential duplicate search object.

Input Message

Name	Type	Description
pdsoBean	PotentialDuplicateSearch ObjectBean	An object containing the criteria for a potential duplicate search.

Output Message

An array of PotentialDuplicateResult objects representing the matches to a potential duplicate search.

Fault Message

- ProcessingException
- UserException

lookupTransaction

Description

This method returns a summary of the transaction associated with the specified transaction number.

Input Message

Name	Type	Description
transId	String	The transaction number for the transaction to look up.

Output Message

A TransactionSummaryBean object representing the transaction summary for the specified transaction ID.

Fault Message

- ProcessingException
- UserException

lookupTransactions

Description

This method returns an array of the transaction summaries based on the search criteria contained in the transaction search object.

Input Message

Name	Type	Description
obj	TransactionSearchObject	The transaction search object containing the search criteria.

Output Message

An array of transaction summaries matching the given search criteria.

Fault Message

- ProcessingException
- UserException

mergeEnterpriseRecord

Description

This method merges two enterprise objects, specified by their EUIDs.

Input Message

Name	Type	Description
fromEUID	String	The EUID of the enterprise object that will not survive the merge.
toEUID	String	The EUID of the enterprise object that will survive the merge.
calculateOnly	boolean	An indicator of whether to commit changes to the database or to simply compute the merge results. Specify **false** to commit the changes.

Output Message

A merge result object containing the results of the merge.

Fault Message

- ProcessingException
- UserException

mergeSystemRecord

Description

This method merges two system objects, specified by their local IDs, from the specified system. The system objects can belong to a single enterprise object or to two different enterprise objects.

Input Message

Name	Type	Description
systemCode	String	The processing code of the system to which the two system objects belong.
sourceLID	String	The local ID of the system object that will not survive the merge.
destLID	String	The local ID of the system object that will survive the merge.
calculateOnly	boolean	An indicator of whether to commit changes to the database or to simply compute the merge results. Specify **false** to commit the changes.

Output Message

A merge result object containing the results of the merge.

Fault Message

- ProcessingException
- UserException

resolvePotentialDuplicate

Description

This method flags a potential duplicate pair with Resolved or Auto Resolved status. If resolved, the pair can be marked as potential duplicates again during a future transaction. If auto-resolved, the pair is permanently flagged as resolved.

Input Message

Name	Type	Description
id	String	The potential duplicate ID of the pair to be resolved.
autoResolve	boolean	A Boolean value indicating whether to resolve or auto-resolve the pair. Use true for auto-resolve, and use false for resolve.

Output Message

None

Fault Message

- ProcessingException
- UserException

search

Description

This method performs a query against the database using the query specified in *queryName* and the criteria contained in the specified object bean.

Input Message

Name	Type	Description
objBean	*ObjectName*Bean	The search criteria for the query.
queryName	String	The name of the query to use.
weightOption	boolean	An indicator of whether to calculate the score of matching. Specify true to calculate.

Output Message

An array of search result records.

Fault Message

- ProcessingException
- UserException

searchBlock

Description

This method performs a blocking query against the database using the blocking query specified in the `master.xml` and the criteria contained in the specified object bean.

Input Message

Name	Type	Description
objBean	*ObjectName*Bean	The search criteria for the blocking query.

Output Message

An array of search result records.

Fault Message

- ProcessingException
- UserException

searchExact

Description

This method performs an exact match search using the criteria specified in the object bean. Only records that exactly match the search criteria are returned in the search results object.

Input Message

Name	Type	Description
objBean	*ObjectName*Bean	The search criteria for the exact match search.

Output Message

An array of search result records.

Fault Message

- ProcessingException
- UserException

searchPhonetic

Description

This method performs search using phonetic values for some of the criteria specified in the object bean. This type of search allows for typographical errors and misspellings.

Input Message

Name	Type	Description
objBean	*ObjectName*Bean	The search criteria for the phonetic search.

Output Message

An array of search result records.

Fault Message

- ProcessingException
- UserException

transferSystemRecord

Description

This method transfers a system record from one enterprise record to another enterprise record.

Input Message

Name	Type	Description
toEUID	String	The EUID of the enterprise record to which the system record will be transferred.
systemCode	String	The processing code of the system record to transfer.
localID	String	The local ID of the system record to transfer.

Output Message

None

Fault Message

- ProcessingException
- UserException

undoAssumedMatch

Description

This method reverses an assumed match transaction, unmerging the two objects that were matched and creating a new enterprise object for the record that caused the assumed match. Potential duplicates are calculated for the new enterprise object.

Input Message

Name	Type	Description
assumedMatchId	String	The ID of the assumed match transaction to reverse.

Output Message

The EUID of the newly created enterprise object resulting from the reversing assumed match.

Fault Message

- ProcessingException
- UserException

unmergeEnterpriseRecord

Description

This method unmerges the two enterprise objects that were involved in the most recent merge transaction for the specified EUID. When the calculateOnly is set to true, the unmerge results is calculated, but the changes are not committed to the database.

Input Message

Name	Type	Description
euid	String	The EUID of the enterprise object to be unmerged.
calculateOnly	String	An indicator of whether to commit the unmerge to the database or to calculate the changes for viewing. Specify true to calculate for viewing; specify false to commit to the database.

Output Message

The MergePersonResult object, the result of the unmerge transaction.

Fault Message

- ProcessingException
- UserException

unmergeSystemRecord

Description

This method unmerges the two system objects that were involved in the most recent merge transaction for the specified local ID. When the calculateOnly is set to true, the unmerge results is computed, but the changes are not committed to the database.

Input Message

Name	Type	Description
systemCode	String	The system code of the system object to be unmerged.
sourceLID	String	The local ID of the non-surviving system object.
destLID	String	The local ID of the surviving system object.
calculateOnly	String	An indicator of whether to commit unmerge to the database or to calculate the changes for viewing. Specify true to calculate for viewing; specify false to commit to the database.

Output Message

The MergePersonResult object, the result of the unmerge transaction.

Fault Message

- ProcessingException
- UserException

unresolvePotentialDuplicate

Description

This method changes the status of a resolved or auto-resolved potential duplicate record pair back to unresolved and places the records back in the potential duplicate listing.

Input Message

Name	Type	Description
id	String	The potential duplicate ID of the records to be unresolved.

Output Message

None

Fault Message

- ProcessingException
- UserException

updateEnterpriseRecord

Description

This method updates the fields in an existing enterprise object with the values specified in the fields the enterprise object passed in as a parameter. When updating

an enterprise object, attempting to change a field that is not updateable will cause an exception. This method does not update the SBR; the survivor calculator updates the SBR once the changes are made to the associated system records. To update SBR via link or lock, you must provide a list of SBROverWriteBean in the SBR<ObjectName>. The list of SBROverWriteBean is a list of SBR fields to be updated.

Input Message

Name	Type	Description
eoBean	Enterprise*ObjectName*	The enterprise object containing the values that will update the existing enterprise object.

Output Message

None

Fault Message

- ProcessingException
- UserException

updateSystemRecord

Description

This method updates the existing system object in the database with the given system object.

Input Message

Name	Type	Description
sysObjBean	System*ObjectName*	The system object to be updated to the enterprise object.

Output Message

None

Fault Message

- ProcessingException
- UserException

IHE Standard Web Services

The following are standard web services for IHE PIXv3/PDQv3 (HL7 v3 based) profiles. They are defined by IHE. The WSDLs and associated schemas are available at ftp://ftp.ihe.net/TF_Implementation_Material/ITI/.

- `pixManagerPRPAIN201301UV02`

  Adds a new patient record to the PIX Manager.

- `pixManagerPRPAIN201302UV02`

  Updates an existing patient record in the PIX Manager.

- `pixManagerPRPAIN201304UV02`

Merges two existing patient records in the PIX Manager.

- `pixManagerPRPAIN201309UV02`

  PIX query to get patient identifiers in selected or all domains against the PIX Manager.

- `pdqSupplierPRPAIN201305UV02`

  Patient demographics query against the PDQ Supplier.

- `pdqSupplierQUQIIN000003UV01Cancel`

  Cancels an existing patient demographics query against the PDQ Supplier.

- `pdqSupplierQUQIIN000003UV01Continue`

  Continues an existing patient demographics query against the PDQ Supplier.