## Contents

**Preface**

<table>
<thead>
<tr>
<th>Audience</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation Accessibility</td>
<td>ix</td>
</tr>
<tr>
<td>Related Documents</td>
<td>ix</td>
</tr>
<tr>
<td>Finding Information and Patches on My Oracle Support</td>
<td>x</td>
</tr>
<tr>
<td>Finding Oracle Documentation</td>
<td>xii</td>
</tr>
</tbody>
</table>

**Conventions**

**1 Introduction to Oracle Healthcare Master Person Index**

| Introducing Oracle Healthcare Master Person Index                        | 1-1|

**2 Understanding Operational Processes**

| Learning About Message Processing                                       | 2-1|
| Inbound Message Processing                                              | 2-2|
| About Inbound Messages                                                  | 2-3|
| Outbound Message Processing                                             | 2-3|
| About Outbound Messages                                                 | 2-4|
| Outbound XSD Structure                                                  | 2-4|
| Outbound Message Trigger Events                                         | 2-5|
| Sample Outbound Message                                                 | 2-5|
| Inbound Message Processing Logic                                         | 2-6|
| Primary Function Processing Logic                                        | 2-11|
| activateEnterpriseObject                                                 | 2-11|
| activateSystemObject                                                     | 2-12|
| addSystemObject                                                          | 2-12|
| createEnterpriseObject                                                   | 2-13|
| deactivateEnterpriseObject                                               | 2-13|
| deactivateSystemObject                                                   | 2-13|
| deleteSystemObject                                                       | 2-13|
| mergeEnterpriseObject                                                    | 2-14|
| mergeSystemObject                                                        | 2-15|
| transferSystemObject                                                     | 2-16|
| undoAssumedMatch                                                         | 2-17|
| unmergeEnterpriseObject                                                  | 2-18|
| unmergeSystemObject                                                      | 2-19|
updateEnterpriseDupRecalc ................................................................. 2-20
updateEnterpriseObject ............................................................... 2-21
updateSystemObject ................................................................. 2-22

3 The Database Structure
Introducing the Structure of the Database Tables ........................................ 3-1
Understanding Database Table Details ........................................... 3-3
SBYN_OBJECT_NAME ............................................................................. 3-3
SBYN_OBJECT_NAMESBR ........................................................................ 3-4
SBYN_CHILD_OBJECT ........................................................................... 3-4
SBYN_CHILD_OBJECTSBR ...................................................................... 3-5
SBYN_API ................................................................................................. 3-5
SBYN_ASSUMEDMATCH ........................................................................... 3-6
SBYN_AUDIT ............................................................................................... 3-6
SBYN_COMMON_DETAIL ............................................................................ 3-7
SBYN_COMMON_HEADER ......................................................................... 3-8
SBYN_ENTERPRISE ..................................................................................... 3-9
SBYN_MERGE .............................................................................................. 3-9
SBYN_OVERWRITE .................................................................................... 3-9
SBYN_POTENTIALDUPLICATES ............................................................. 3-10
SBYN_SEQ_TABLE ..................................................................................... 3-11
SBYN_SYSTEMOBJECT ............................................................................... 3-12
SBYN_SYSTEMS ......................................................................................... 3-13
SBYN_SYSTEMSBR .................................................................................... 3-14
SBYN_TRANSACTION .................................................................................. 3-15
SBYN_USER_CODE ..................................................................................... 3-16
Viewing a Sample Database Model .................................................. 3-17

4 Working with the Java API
Understanding Java Class Types .................................................. 4-1
Static Classes ......................................................................................... 4-1
Dynamic Web Services Classes ..................................................... 4-1
Dynamic Web Service Methods .................................................. 4-2
Dynamic API Method Descriptions ........................................ 4-2
activateEnterpriseRecord .............................................................. 4-3
activateSystemRecord ............................................................... 4-3
daddSystemRecord .............................................................................. 4-4
deactivateEnterpriseRecord .......................................................... 4-4
deactivateSystemRecord .............................................................. 4-5
executeMatch ......................................................................................... 4-5
gexecuteMatchUpdate ................................................................. 4-6
gEnterpriseRecordByEUID ............................................................. 4-7
gEnterpriseRecordByLID ............................................................. 4-7
gEUID ........................................................................................................ 4-8
gLIDs ........................................................................................................... 4-8
gLIDsByStatus ......................................................................................... 4-9
gSBR ........................................................................................................... 4-9
getSystemRecord................................................................. 4-10
getSystemRecordsByEUID .................................................. 4-10
getSystemRecordsByEUIDStatus ....................................... 4-11
lookupLIDs......................................................................... 4-11
lookupPotentialDuplicates..................................................... 4-12
mergeEnterpriseRecord ..................................................... 4-12
mergeSystemRecord.......................................................... 4-13
searchBlock.......................................................................... 4-14
searchExact.......................................................................... 4-14
searchPhonetic..................................................................... 4-15
transferSystemRecord.......................................................... 4-15
updateEnterpriseRecord ...................................................... 4-16
updateSystemRecord........................................................... 4-16

Web Services Helper Classes ............................................. 4-17
Enterprise Object Classes .................................................. 4-17
  Definition ........................................................................ 4-17
  Methods........................................................................ 4-17
    EnterpriseObjectName.................................................. 4-18
    countSystemObjectName............................................. 4-18
    deleteSystemObject................................................... 4-19
    deleteSystemObjectNameBean..................................... 4-19
    getEUID................................................................. 4-20
    getSBRObjectName.................................................. 4-20
    getStatus..................................................................... 4-20
    getSystemObjectName................................................ 4-21
    getSystemObjectName................................................ 4-21
    getSystemObjectNameCount....................................... 4-21
    removeSystemObject.................................................. 4-22
    removeSystemObjectNameBean................................. 4-22
    setEUID................................................................. 4-23
    setStatus..................................................................... 4-23
    setSystemObjectName................................................ 4-23
SystemObjectName............................................................ 4-24
  Definition........................................................................ 4-24
  Methods........................................................................ 4-24
    Inherited Methods....................................................... 4-24
      Clear Field Indicator............................................. 4-25
    SystemObjectName................................................... 4-25
      getClearFieldIndicator......................................... 4-25
      getField.................................................................. 4-26
      getName.................................................................. 4-26
      setClearFieldIndicator......................................... 4-26
      setName.................................................................. 4-26
    SystemObjectName................................................... 4-26
  Definition........................................................................ 4-27
  Methods........................................................................ 4-27
Parent Beans...................................................................... 4-28
  Definition........................................................................ 4-28
  Methods........................................................................ 4-28
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 OPHMPI.

Audience

This document is intended for users of OPHMPI 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 Release 2.0 documentation set:

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

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

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 a 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** (Part
  Number E24708-01)

- **Oracle Healthcare Master Person Index United Kingdom Patient Solution User’s Guide**
  (Part Number E24710-01)

- **Oracle Healthcare Master Person Index United States Patient Solution User’s Guide** (Part
  Number E24709-01)

- **Oracle Healthcare Master Person Index Provider Index User’s Guide** (Part Number
  E25035-01)
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, "Oracle Healthcare Master Person Index Processing Flow" illustrates the flow of information through an OHMPI application that includes a JMS Topic to which updates to the index are published.
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, "Inbound Message Processing Data Flow" 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.
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, "Outbound Message Processing Data Flow" illustrates the flow of data for a message outbound from an OHMPI application.

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, "Outbound XSD SBR Attributes" describes the components of the SBR portion of the outbound OTD.

Table 2–1 Outbound XSD SBR Attributes

<table>
<thead>
<tr>
<th>Node</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUID</td>
<td>The EUID of the record that was inserted or modified.</td>
</tr>
<tr>
<td>Status</td>
<td>The status of the record.</td>
</tr>
<tr>
<td>CreateFunction</td>
<td>The date the record was first created.</td>
</tr>
<tr>
<td>CreateUser</td>
<td>The logon ID of the user who created the record.</td>
</tr>
<tr>
<td>UpdateSystem</td>
<td>The processing code of the external system from which the updates to an existing record originated.</td>
</tr>
<tr>
<td>ChildType</td>
<td>The name of the parent object.</td>
</tr>
<tr>
<td>CreateSystem</td>
<td>The processing code of the external system from which the record originated.</td>
</tr>
</tbody>
</table>
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.
Learning About Message Processing

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 (Part Number E25249-01).

- OneExactMatch parameter
- SameSystemMatch parameter

2-6 Oracle Healthcare Master Person Index Understanding Message Processing Reference
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 (Part Number E25249-01).

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 (Part Number E25249-01).

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

- Message containing a system and user identifier, and subject to processing, is received.
- Inbound message used to build D5 with match criteria table.
- Are the section and local data pair found?
  - No: Perform update on the matching request (synchronous or asynchronous execution).
  - Yes: Match matching algorithm search.
- Are records matched with matching algorithm input? (Can also be retrieved by user-defined procedures)
  - No: Go to next.
  - Yes: Perform a new record event.

Processing complete.
Figure 2–5 Inbound Message Processing Using executeMatch (continued)
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 those 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, "activateEnterpriseObject Processing" illustrates the processing steps.
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, “activateSystemObject Processing” illustrates the processing steps.

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, “addSystemObject Processing” illustrates the processing steps.
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, “deactivateSystemObject Processing” illustrates the processing steps.

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
Learning About Message Processing

key fields were updated, potential duplicates are recalculated for the enterprise record. 
Figure 2-11, "deleteSystemObject Processing" illustrates the processing steps.

**Figure 2-11 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 2-12, "mergeEnterpriseObject Processing" illustrates the processing steps, and includes the check for SBR differences, which only occurs in two of the merge methods.
mergeEnterpriseObject

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 Javadoc 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.

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.

**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.

Figure 2–12  mergeEnterpriseObject Processing
Learning About Message Processing

enterprise record. Figure 2–13, “mergeSystemObject Processing” illustrates the processing steps, and includes the check for SBR differences, which only occurs in two of the merge methods.

**Figure 2–13 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 2–14, “transferSystemObject Processing” illustrates the processing steps.
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 2–15, “undoAssumedMatch Processing” illustrates the processing steps.
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 2–16, “unmergeEnterpriseObject Processing” illustrates the processing steps and includes the check for SBR updates.
Learning About Message Processing

Figure 2–16  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 2–17, “unmergeSystemObject Processing” illustrates the processing steps, assuming the system record unmerge involves two enterprise records and including the check for SBR updates.
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 2–18, "updateEnterpriseDupRecalc Processing" illustrates the processing steps.
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 SRR of the enterprise record. If key fields were updated, potential duplicates are recalculated. Figure 2–19, “updateEnterpriseObject Processing” illustrates the processing steps.

Figure 2–19  updateEnterpriseObject Processing
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 2–20, “updateSystemObject Processing”, illustrates the processing steps and includes the check for SBR changes though it only occurs with one of the methods.

**Figure 2–20  updateSystemObject Processing**
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"
- "Understanding Database Table Details"
- "Viewing a Sample Database Model"

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>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBYN_OBJECT_NAME</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 3–1 Oracle Healthcare Master Person Index Database Tables
Introducing the Structure of the Database Tables

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBYN_OBJECT_NAMESBR</td>
<td>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_company_sbr; a table storing person objects is named sbyn_person_sbr. Only one table stores parent object information for SBRs.</td>
</tr>
<tr>
<td>SBYN_CHILD_OBJECT</td>
<td>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.</td>
</tr>
<tr>
<td>SBYN_CHILD_OBJECTSBR</td>
<td>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_address_sbr; a table storing comment objects is named sbyn_comment_sbr. One SBR database table is created for each child object defined in the object structure.</td>
</tr>
<tr>
<td>SBYN_APPL</td>
<td>Lists the applications with which each item in stc_common_header is associated. Currently the only item in this table is MPI.</td>
</tr>
<tr>
<td>SBYN_ASSUMEDMATCH</td>
<td>Stores information about records that were automatically matched by the OHMPI application.</td>
</tr>
<tr>
<td>SBYN_AUDIT</td>
<td>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.</td>
</tr>
<tr>
<td>SBYN_COMMON_DETAIL</td>
<td>Contains all of the processing codes associated with the items listed in sbyn_common_header.</td>
</tr>
<tr>
<td>SBYN_COMMON_HEADER</td>
<td>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.</td>
</tr>
<tr>
<td>SBYN.ENTERPRISE</td>
<td>Stores the local ID and system pairs, along with their associated EUID.</td>
</tr>
<tr>
<td>SBYN.MERGE</td>
<td>Stores information about all merge and unmerge transactions processed from either external systems or the MIDM.</td>
</tr>
<tr>
<td>SBYN.OVERWRITE</td>
<td>Stores information about fields that are locked for updates in an SBR.</td>
</tr>
<tr>
<td>SBYN.POTENTIALDUPLICATES</td>
<td>Stores a list of potential duplicate records and flags potential duplicate pairs that have been resolved.</td>
</tr>
<tr>
<td>SBYN_SEQ_TABLE</td>
<td>Stores the sequential codes that are used in other tables in the database, such as EUIDs, transaction numbers, and so on.</td>
</tr>
<tr>
<td>SBYN_SYSTEMOBJECT</td>
<td>Stores information about the system objects in the database, including the local ID and system, create date and user, status, and so on.</td>
</tr>
<tr>
<td>SBYN_SYSTEMS</td>
<td>Stores a list of systems in your organization, along with defining information.</td>
</tr>
<tr>
<td>SBYN_SYSTEMSBR</td>
<td>Stores transaction information about an SBR, such as the create or update date, status, and so on.</td>
</tr>
<tr>
<td>SBYN_TRANSACTION</td>
<td>Stores a history of changes to each record stored in the database.</td>
</tr>
</tbody>
</table>
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.

**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 \textit{object\_name} id column (where \textit{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 \textit{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 3–2 Oracle, MySQL, and SQL Server Data Type Differences

<table>
<thead>
<tr>
<th>Oracle Data Type</th>
<th>SQL Server Data Type</th>
<th>MySQL Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>Varbinary(MAX)</td>
<td>blob</td>
</tr>
<tr>
<td>DATE</td>
<td>DateTime</td>
<td>datetime</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Int</td>
<td>integer</td>
</tr>
<tr>
<td>LONG</td>
<td>Varchar(MAX)</td>
<td>mediumtext</td>
</tr>
<tr>
<td>NUMBER</td>
<td>Numeric</td>
<td>decimal</td>
</tr>
<tr>
<td>TIMESTAMP*</td>
<td>Date/Time</td>
<td>datetime</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>Varchar</td>
<td>varchar</td>
</tr>
</tbody>
</table>

### Table 3–3 SBYN_OBJECT_NAME Table Description

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEMCODE</td>
<td>VARCHAR2(20)</td>
<td>The system code for the system record.</td>
</tr>
<tr>
<td>LID</td>
<td>VARCHAR2(25)</td>
<td>A local identification code assigned by the specified system.</td>
</tr>
</tbody>
</table>
Understanding Database Table Details

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

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUID</td>
<td>VARCHAR2(20)</td>
<td>The enterprise unique identifier assigned by the OHMPI application.</td>
</tr>
<tr>
<td>OBJECT_NAMEID</td>
<td>VARCHAR2(20)</td>
<td>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 &quot;Company&quot;, the name of this column is &quot;companyid&quot;; if the parent object is &quot;Person&quot;, the name of this column is &quot;personid&quot;.</td>
</tr>
</tbody>
</table>

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 3–4 SBYN_CHILD_OBJECT Table Description

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUID</td>
<td>VARCHAR2(20)</td>
<td>The enterprise unique identifier assigned by the OHMPI application.</td>
</tr>
<tr>
<td>OBJECT_NAMEID</td>
<td>VARCHAR2(20)</td>
<td>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 &quot;Company&quot;, the name of this column is &quot;companyid&quot;; if the parent object is &quot;Person&quot;, the name of this column is &quot;personid&quot;.</td>
</tr>
</tbody>
</table>
The sbyn_child_objectsbr 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 3–5 SBYN_CHILD_OBJECT Table Description

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_NAMEID</td>
<td>VARCHAR2(20)</td>
<td>The unique ID for the parent object associated with the child object in the system record.</td>
</tr>
<tr>
<td>CHILD_OBJECTID</td>
<td>VARCHAR2(20)</td>
<td>The unique ID for each record in the table. This column cannot be null.</td>
</tr>
</tbody>
</table>

### SBYN_CHILD_OBJECTSBR

The sbyn_child_objectsbr 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

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Column Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_NAMEID</td>
<td>VARCHAR2(20)</td>
<td>The unique ID for the parent object associated with the child object in the SBR.</td>
</tr>
<tr>
<td>CHILD_OBJECTID</td>
<td>VARCHAR2(20)</td>
<td>The unique ID for each record in the table. This column cannot be null.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPL_ID</td>
<td>NUMBER(10)</td>
<td>The unique sequence number code for the listed application.</td>
</tr>
<tr>
<td>CODE</td>
<td>VARCHAR2(20)</td>
<td>A unique code for the application.</td>
</tr>
<tr>
<td>DESCR</td>
<td>VARCHAR2(30)</td>
<td>A brief description of the application.</td>
</tr>
<tr>
<td>READ_ONLY</td>
<td>CHAR(1)</td>
<td>An indicator of whether the current entry can be modified. If the value of this column is Y, the entry cannot be modified.</td>
</tr>
<tr>
<td>CREATE_DATE</td>
<td>DATE</td>
<td>The date the application entry was created.</td>
</tr>
</tbody>
</table>
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**

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

**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**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT_ID</td>
<td>VARCHAR2(20)</td>
<td>The unique identification code for the audit record. This column cannot be null.</td>
</tr>
<tr>
<td>PRIMARY_OBJECT_TYPE</td>
<td>VARCHAR2(20)</td>
<td>The name of the parent object as defined in object.xml.</td>
</tr>
</tbody>
</table>
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 3–9 (Cont.) SBYN_AUDIT Table Description

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUID</td>
<td>VARCHAR2(15)</td>
<td>The EUID whose information was accessed during an MIDM transaction.</td>
</tr>
<tr>
<td>EUID_AUX</td>
<td>VARCHAR2(15)</td>
<td>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.</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>VARCHAR2(32)</td>
<td>The type of transaction that caused the audit record to be written. This column cannot be null.</td>
</tr>
<tr>
<td>OPERATION</td>
<td>VARCHAR2(32)</td>
<td>A brief description of the transaction that caused the audit record to be written.</td>
</tr>
<tr>
<td>DETAIL</td>
<td>VARCHAR2(120)</td>
<td>The date the transaction that created the audit record was performed. This column cannot be null.</td>
</tr>
<tr>
<td>CREATE_DATE</td>
<td>DATE</td>
<td>The user ID of the person who performed the transaction that caused the audit log. This column cannot be null.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON_DETAIL_ID</td>
<td>NUMBER(10)</td>
<td>The unique identification code of the common table data element.</td>
</tr>
<tr>
<td>COMMON_HEADER_ID</td>
<td>NUMBER(10)</td>
<td>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).</td>
</tr>
<tr>
<td>CODE</td>
<td>VARCHAR2(20)</td>
<td>The processing code for the common table data element.</td>
</tr>
<tr>
<td>DESCR</td>
<td>VARCHAR2(50)</td>
<td>A description of the common table data element.</td>
</tr>
</tbody>
</table>
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 3–10 SBYN_COMMON_DETAIL Table Description

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

### Table 3–11 SBYN_COMMON_HEADER Table Description

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON_HEADER_ID</td>
<td>VARCHAR2(10)</td>
<td>The unique identification code of the common table data type.</td>
</tr>
<tr>
<td>APPL_ID</td>
<td>VARCHAR2(10)</td>
<td>The application ID from sbyn_appl that corresponds to the application for which the common table data type is used.</td>
</tr>
<tr>
<td>CODE</td>
<td>VARCHAR2(8)</td>
<td>A unique processing code for the common table data type.</td>
</tr>
<tr>
<td>DESCR</td>
<td>VARCHAR2(50)</td>
<td>A description of the common table data type.</td>
</tr>
<tr>
<td>READ_ONLY</td>
<td>CHAR(1)</td>
<td>An indicator of whether an entry in the table is read-only (if this column is set to “Y”, the entry is read-only).</td>
</tr>
<tr>
<td>MAX_INPUT_LEN</td>
<td>NUMBER(10)</td>
<td>The maximum number of characters allowed in the code column for the common table data type.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEMCODE</td>
<td>VARCHAR2(20)</td>
<td>The processing code of the system associated with the local ID.</td>
</tr>
<tr>
<td>LID</td>
<td>VARCHAR2(25)</td>
<td>The local ID associated with the system and EUID.</td>
</tr>
<tr>
<td>EUID</td>
<td>VARCHAR2(20)</td>
<td>The EUID associated with the local ID and system.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERGE_ID</td>
<td>VARCHAR2(20)</td>
<td>The unique, sequential identification code of merge record. This column cannot be null.</td>
</tr>
<tr>
<td>KEPT_EUID</td>
<td>VARCHAR2(20)</td>
<td>The EUID of the record that was retained after the merge transaction. This column cannot be null.</td>
</tr>
<tr>
<td>MERGED_EUID</td>
<td>VARCHAR2(20)</td>
<td>The EUID of the record that was not retained after the merge transaction.</td>
</tr>
<tr>
<td>MERGE_TRANSACTIONNUM</td>
<td>VARCHAR2(20)</td>
<td>The transaction number associated with the merge transaction. This column cannot be null.</td>
</tr>
<tr>
<td>UNMERGE_TRANSACTIONNUM</td>
<td>VARCHAR2(20)</td>
<td>The transaction number associated with the unmerge transaction.</td>
</tr>
</tbody>
</table>

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".
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".

### SBYN_POTENTIALDUPLICATES

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POTENTIALDUPLICATEID</td>
<td>VARCHAR2(20)</td>
<td>The unique identification number of the potential duplicate transaction.</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>VARCHAR2(20)</td>
<td>The matching weight of the potential duplicate pair.</td>
</tr>
<tr>
<td>TYPE</td>
<td>VARCHAR2(15)</td>
<td>This column is reserved for future use.</td>
</tr>
</tbody>
</table>

---

### SBYN_OVERWRITE Table Description

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

---

This 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

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

---

This 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

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POTENTIALDUPLICATEID</td>
<td>VARCHAR2(20)</td>
<td>The unique identification number of the potential duplicate transaction.</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>VARCHAR2(20)</td>
<td>The matching weight of the potential duplicate pair.</td>
</tr>
<tr>
<td>TYPE</td>
<td>VARCHAR2(15)</td>
<td>This column is reserved for future use.</td>
</tr>
</tbody>
</table>
This table is used to control and maintain sequential identification numbers for various tables in the database. The numbers are assigned sequentially, but are cached in chunks of 1000 numbers for optimization. If the Repository server is reset before all allocated numbers are used, the unused numbers are discarded 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>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ_NAME</td>
<td>VARCHAR(20)</td>
<td>The name of the object for which the sequential ID is stored.</td>
</tr>
<tr>
<td>SEQ_COUNT</td>
<td>NUMBER(38)</td>
<td>The current value of the sequence. The next record will be assigned the current value plus one.</td>
</tr>
</tbody>
</table>
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 3–17 Default Sequence Numbers

<table>
<thead>
<tr>
<th>Sequence Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUID</td>
<td>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.</td>
</tr>
<tr>
<td>POTENTIALDUPLICATE</td>
<td>The sequence number assigned each potential duplicate transaction record in sbyn_potentialduplicates (column name “potentialduplicatesid”).</td>
</tr>
<tr>
<td>TRANSACTIONNUMBER</td>
<td>The sequence number assigned each transaction in the OHMPI application. This number is stored in sbyn_transaction (column name “transactionnumber”).</td>
</tr>
<tr>
<td>ASSUMEDMATCH</td>
<td>The sequence number assigned to each assumed match transaction record in sbyn_assumedmatch (column name “assumedmatchid”).</td>
</tr>
<tr>
<td>AUDIT</td>
<td>The sequence number assigned to each audit log record in sbyn_audit (column name “audit_id”).</td>
</tr>
<tr>
<td>MERGE</td>
<td>The sequence number assigned to each merge transaction in sbyn_merge (column name “merge_id”).</td>
</tr>
<tr>
<td>SBYN_APPL</td>
<td>The sequence number assigned to each application listed in sbyn_appl (column name “appl_id”).</td>
</tr>
<tr>
<td>SBYN_COMMON_HEADER</td>
<td>The sequence number assigned to each common table data type listed in sbyn_common_header (column name “common_header_id”).</td>
</tr>
<tr>
<td>SBYN_COMMON_DETAIL</td>
<td>The sequence number assigned to each common table data element listed in sbyn_common_detail (column name “common_detail_id”).</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>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.</td>
</tr>
<tr>
<td>OBJECT_NAMESBR</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Table 3–18 SBYN_SYSTEMOBJECT Table Description

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEMCODE</td>
<td>VARCHAR2(20)</td>
<td>The processing code of the system associated with the local ID. This column cannot be null.</td>
</tr>
</tbody>
</table>
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".

### SBYN_SYSTEMS

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEMCODE</td>
<td>VARCHAR2(20)</td>
<td>The unique processing code of the system.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR2(120)</td>
<td>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.</td>
</tr>
<tr>
<td>STATUS</td>
<td>CHAR(1)</td>
<td>The status of the system in the OHMFPY application. A indicates active and D indicates deactivated.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_LENGTH</td>
<td>NUMBER</td>
<td>The length of the local identifiers assigned by the system. This length does not include any additional characters added by the input mask.</td>
</tr>
<tr>
<td>FORMAT</td>
<td>VARCHAR2(60)</td>
<td>The required data pattern for the local IDs assigned by the system. For more information about possible values and using Java patterns, see &quot;Patterns&quot; 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.</td>
</tr>
<tr>
<td>INPUT_MASK</td>
<td>VARCHAR2(60)</td>
<td>A mask used by the MIDM to add punctuation to the local ID. For example, the input mask DD-DDD-DDDD inserts a hyphen after the second and fifth characters in an 8-digit ID. These character types can be used.</td>
</tr>
<tr>
<td>VALUE_MASK</td>
<td>VARCHAR2(60)</td>
<td>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 DDxDDDDxDDD. This strips the hyphens before storing the ID.</td>
</tr>
<tr>
<td>CREATE_DATE</td>
<td>DATE</td>
<td>The date the system information was inserted into the database.</td>
</tr>
<tr>
<td>CREATE_USERID</td>
<td>VARCHAR2(20)</td>
<td>The logon ID of the user who inserted the system information into the database.</td>
</tr>
<tr>
<td>UPDATE_DATE</td>
<td>DATE</td>
<td>The most recent date the system’s information was updated.</td>
</tr>
<tr>
<td>UPDATE_USERID</td>
<td>VARCHAR2(20)</td>
<td>The logon ID of the user who last updated the system’s information.</td>
</tr>
</tbody>
</table>
This table stores a history of changes made to each record in the OHMPLI 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>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTIONNUMBER</td>
<td>VARCHAR2(20)</td>
<td>The unique number of the transaction.</td>
</tr>
<tr>
<td>LID1</td>
<td>VARCHAR2(25)</td>
<td>This column is reserved for future use.</td>
</tr>
<tr>
<td>LID2</td>
<td>VARCHAR2(25)</td>
<td>The local ID of the second system record involved in the transaction.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE_LIST</td>
<td>VARCHAR2(20)</td>
<td>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.</td>
</tr>
<tr>
<td>CODE</td>
<td>VARCHAR2(20)</td>
<td>The processing code of each user code element.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR2(50)</td>
<td>A brief description or name for the user code. This is the value that appears in the drop-down list.</td>
</tr>
</tbody>
</table>
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)
This chapter provides an overview of the Java API for an Oracle Healthcare Master Person Index (OHMPI) application, and describes the dynamic classes and methods that are generated based on the object structure of the OHMPI application. For detailed information about the static 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 section:

- "Understanding Java Class Types"

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 static API classes that can be used with any object structure and any OHMPI application. It also generates several dynamic API classes that are specific to the object structure of each OHMPI application. The dynamic classes contain similar methods, but the number and names of methods change depending on the object structure.

Static Classes

Static 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 static methods is provided in Javadoc format.

Dynamic Web Services Classes

When you generate a Master Person Index or IHE Profile project, several dynamic methods are created in the Master Person Index EJB project that are specific to the object structure defined for the OHMPI application. This includes classes that define each parent and child object in the object structure and that allow you to work with the
data in each object. If the object structure is modified, regenerating the project updates
the dynamic methods for the new structure.

The EJB class that defines the primary web service methods is named after the parent
object, and defines the main methods you use to transform and manipulate data
between external systems and the OHMPI database. There are also several helper
classes that define the objects that are used as parameters and results for the primary
web services, and that define the objects involved in OHMPI processing.

The dynamic web services classes are located in the OHMPI EJB project in the
com.sun.mdm.index.webservice package under Source Packages. You can view the
primary EJB class under Web Services in the EJB project. Additional supporting API
classes are generated in the remaining Java packages under Source Packages.

Dynamic Web Service Methods

A set of Java methods are created in the OHMPI EJB project for use in the client
applications that call OHMPI functions. These methods wrap static Java API methods,
allowing them to work with the dynamic object classes. Many of these methods return
objects of the dynamic object type, or they use these objects as parameters. In the
following methods described for web services, ObjectName indicates the name of the
parent object.

Dynamic API Method Descriptions

- ‘activateEnterpriseRecord’
- ‘activateSystemRecord’
- ‘addSystemRecord’
- ‘deactivateEnterpriseRecord’
- ‘deactivateSystemRecord’
- ‘executeMatch’
- ‘executeMatchUpdate’
- ‘getEnterpriseRecordByEUID’
- ‘getEnterpriseRecordByLID’
- ‘getEUID’
- ‘getLIDs’
- ‘getLIDsByStatus’
- ‘getSBR’
- ‘getSystemRecord’
- ‘getSystemRecordsByEUID’
- ‘getSystemRecordsByEUIDstatus’
- ‘lookupLIDs’
- ‘lookupPotentialDuplicates’
- ‘mergeEnterpriseRecord’
- ‘mergeSystemRecord’
- ‘searchBlock’
Dynamic Web Service Methods

- "searchPhonetic"
- "searchExact"
- "transferSystemRecord"
- "updateEnterpriseRecord"
- "updateSystemRecord"

activateEnterpriseRecord

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

**Syntax**
```java
void activateEnterpriseRecord(String euid)
```

**Parameters**
- **Name**: euid
  - **Type**: String
  - **Description**: The EUID of the enterprise object to activate.

**Returns**
None.

**Throws**
- ProcessingException
- UserException

activateSystemRecord

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

**Syntax**
```java
void activateSystemRecord(String systemCode, String localId)
```

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

**Returns**
None.

**Throws**
- ProcessingException
Dynamic Web Service Methods

- UserException

addSystemRecord

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

**Syntax**
```java
void addSystemRecord(String euid, SystemObjectName sysObjBean)
```

where ObjectName is the name of the parent object.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>euid</td>
<td>String</td>
<td>The EUID of the enterprise object to which you want to add the system object.</td>
</tr>
<tr>
<td>sysObjBean</td>
<td>SystemObjectName</td>
<td>The Bean for the system object to be added to the enterprise object.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**Throws**
- ProcessingException
- UserException

deactivateEnterpriseRecord

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

**Syntax**
```java
void deactivateEnterpriseRecord(String euid)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>euid</td>
<td>String</td>
<td>The EUID of the enterprise object to deactivate.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**Throws**
- ProcessingException
- UserException
deactivateSystemRecord

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

Syntax
void deactivateSystemRecord(String systemCode, String localId)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>systemCode</td>
<td>String</td>
<td>The system code of the system object to deactivate.</td>
</tr>
<tr>
<td>localId</td>
<td>String</td>
<td>The local ID of the system object to deactivate.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
- ProcessingException
- UserException

evaluateMatch
evaluateMatch 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 evaluate match methods should be used for inserting or updating a record in the database.

The following runtime components configure evaluateMatch.

- 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 evaluateMatch 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.

Syntax

```java
MatchColResult evaluateMatch(SystemObjectName sysObjBean)
```

where ObjectName is the name of the parent object.
executeMatchUpdate

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.

Syntax

MatchColResult executeMatchUpdate(SystemObjectName sysObjBean)

where ObjectName is the name of the parent object.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysObjBean</td>
<td>SystemObjectName</td>
<td>The Bean for the system object to be added to or updated in the enterprise object.</td>
</tr>
</tbody>
</table>

Returns

A match result object containing the results of the matching process.
Dynamic Web Service Methods

Throws
- ProcessingException
- UserException

**getEnterpriseRecordByEUID**

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

Syntax
```java
EnterpriseObjectName getEnterpriseRecordByEUID(String euid)
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>euid</td>
<td>String</td>
<td>The EUID of the enterprise object you want to retrieve.</td>
</tr>
</tbody>
</table>

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

Throws
- ProcessingException
- UserException

**getEnterpriseRecordByLID**

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

Syntax
```java
EnterpriseObjectName getEnterpriseRecordByLID(String systemCode, String localId)
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>systemCode</td>
<td>String</td>
<td>The system code of a system associated with the enterprise object to find.</td>
</tr>
<tr>
<td>localId</td>
<td>String</td>
<td>A local ID associated with the specified system.</td>
</tr>
</tbody>
</table>

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

Throws
- ProcessingException
- UserException
Dynamic Web Service Methods

getEUID

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

**Syntax**
```java
String getEUID(String systemCode, String localID)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>systemCode</td>
<td>String</td>
<td>A known system code for the enterprise object.</td>
</tr>
<tr>
<td>localID</td>
<td>String</td>
<td>The local ID corresponding with the given system.</td>
</tr>
</tbody>
</table>

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

**Throws**
- ProcessingException
- UserException

getLIDs

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

**Syntax**
```java
SystemObjectNamePK[] getLIDs(String euid)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>euid</td>
<td>String</td>
<td>The EUID of the enterprise object whose local ID and system pairs you want to retrieve.</td>
</tr>
</tbody>
</table>

**Returns**
An array of system object keys (SystemObjectNamePK objects) or null if no results are found.

**Throws**
- 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.

Syntax
```
SystemObjectNamePK[] getLIDsByStatus(String euid, String status)
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>euid</td>
<td>String</td>
<td>The EUID of the enterprise object whose local ID and system pairs to retrieve.</td>
</tr>
<tr>
<td>status</td>
<td>String</td>
<td>The status of the local ID and system pairs to retrieve.</td>
</tr>
</tbody>
</table>

Returns
An array of system object keys (SystemObjectNamePK objects) or null if no system object keys are found.

Throws
- ProcessingException
- UserException

getSBR

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

Syntax
```
SBRK[] getSBR(String euid)
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>euid</td>
<td>String</td>
<td>The EUID of the enterprise object whose SBR you want to retrieve.</td>
</tr>
</tbody>
</table>

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

Throws
- ProcessingException
- UserException
getSystemRecord

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

Syntax
SystemObjectName getSystemRecord(String systemCode, String localid)

Parameters
- systemCode: String, The system code of the system object to retrieve.
- localid: String, The local ID of the system object to retrieve.

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

Throws
- Exception
- UserException

getSystemRecordsByEUID

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

Syntax
SystemObjectName[] getSystemRecordsByEUID(String euid)

Parameters
- euid: String, The EUID of the enterprise object whose system objects you want to retrieve.

Returns
An array of system objects associated with the specified EUID.

Throws
- Exception
- UserException
getSystemRecordsByEUIDStatus

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

**Syntax**
```java
SystemObjectName[] getSystemRecordsByEUIDStatus(String euid, String status)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>euid</td>
<td>String</td>
<td>The EUID of the enterprise object whose system objects you want to retrieve.</td>
</tr>
<tr>
<td>status</td>
<td>String</td>
<td>The status of the system objects you want to retrieve.</td>
</tr>
</tbody>
</table>

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

**Throws**
- 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.

**Syntax**
```java
SystemObjectNamePK[] lookupLIDs(String sourceSystemCode, String sourceLID, String destSystemCode, String status)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceSystemCode</td>
<td>String</td>
<td>The system code of the known system and local ID pair.</td>
</tr>
<tr>
<td>sourceLID</td>
<td>String</td>
<td>The local ID of the known system and local ID pair.</td>
</tr>
<tr>
<td>destSystemCode</td>
<td>String</td>
<td>The system from which the local ID and system pairs to retrieve originated.</td>
</tr>
</tbody>
</table>
Dynamic Web Service Methods

lookupPotentialDuplicates

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

Syntax
PotentialDuplicateResult[] lookupPotentialDuplicates(PotentialDuplicateSearchObjectBean pdsoBean)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdsoBean</td>
<td>PotentialDuplicateSearchObjectBean</td>
<td>An object containing the criteria for a potential duplicate search.</td>
</tr>
</tbody>
</table>

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

Throws
- ProcessingException
- UserException
- PageException
- RemoteException

mergeEnterpriseRecord

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

Syntax
MergeObjectNameResult mergeEnterpriseRecord(String fromEUID, String toEUID, boolean calculateOnly)

status String
The status of the local ID and system pairs to retrieve.

Returns
An array of system object keys (SystemObjectNamePK objects).

Throws
- ProcessingException
- UserException
- PageException
- RemoteException
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromEUID</td>
<td>String</td>
<td>The EUID of the enterprise object that will not survive the merge.</td>
</tr>
<tr>
<td>toEUID</td>
<td>String</td>
<td>The EUID of the enterprise object that will survive the merge.</td>
</tr>
<tr>
<td>calculateOnly</td>
<td>boolean</td>
<td>An indicator of whether to commit changes to the database or to simply compute the merge results. Specify false to commit the changes.</td>
</tr>
</tbody>
</table>

Returns

A merge result object containing the results of the merge.

Throws

- 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.

Syntax

```
MergeObjectResult mergeSystemRecord(String systemCode, String sourceLID, String destLID, boolean calculateOnly)
```

Parameters

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

Returns

A merge result object containing the results of the merge.

Throws

- ProcessingException
- UserException
Dynamic Web Service Methods

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.

Syntax
SearchObjectNameResult[] searchBlock(ObjectNameBean objBean)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objBean</td>
<td>ObjectNameBean</td>
<td>The search criteria for the blocking query</td>
</tr>
</tbody>
</table>

Returns
An array of search result records.

Throws
- 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.

Syntax
SearchObjectNameResult[] searchExact(ObjectNameBean objBean)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objBean</td>
<td>ObjectNameBean</td>
<td>The search criteria for the exact match search</td>
</tr>
</tbody>
</table>

Returns
An array of search result records.

Throws
- 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.

Syntax
SearchObjectResult[] searchPhonetic(ObjectNameBean objBean)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objBean</td>
<td>ObjectNameBean</td>
<td>The search criteria for the phonetic search.</td>
</tr>
</tbody>
</table>

Returns
An array of search result records.

Throws
- ProcessingException
- UserException

transferSystemRecord

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

Syntax
void transferSystemRecord(String toEUID, String systemCode, String localID)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toEUID</td>
<td>String</td>
<td>The EUID of the enterprise record to which the system record will be transferred.</td>
</tr>
<tr>
<td>systemCode</td>
<td>String</td>
<td>The processing code of the system record to transfer.</td>
</tr>
<tr>
<td>localID</td>
<td>String</td>
<td>The local ID of the system record to transfer.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
- 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.

**Syntax**
```java
void updateEnterpriseRecord(EnterpriseObjectName eoBean)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eoBean</td>
<td>EnterpriseObjectName</td>
<td>The enterprise object containing the values that will update the existing enterprise object.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**Throws**
- ProcessingException
- UserException

**updateSystemRecord**

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

**Syntax**
```java
void updateSystemRecord(SystemObjectName sysObjBean)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysObjBean</td>
<td>SystemObjectName</td>
<td>The system object to be updated to the enterprise object.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**Throws**
- ProcessingException
- UserException
Web Services Helper Classes

Helper classes are used in MPI web service implementation. The client application, such as web service client or business processes, should not use them. Instead, the WSDL of the MPI web service should be used in client applications. The helper classes include:

- "SystemObjectName"
- "Parent Beans"
- "Child Beans"
- "DestinationEO"
- "MergeObjectNameResult"
- "SBRObjectName"
- "SearchObjectNameResult"
- "SourceEO"
- "SystemObjectNamePK"

Helper classes are located in the OHMPI EJB project in the com.sun.mdm.index.webservice package.

Enterprise Object Classes

A Java class is created to represent the entire object, which includes the single best record (SBR) and any associated system objects. The methods in this class provide the ability to retrieve, set, and delete system objects, and to retrieve and set attributes of the enterprise object.

The name of the enterprise object class is the same as the name of the primary object, with the word "Enterprise" prepended. For example, if the primary object name is "Person", the name of the enterprise object class is EnterprisePerson. The methods in this class include a constructor method for the enterprise object, get and set methods for object attributes, and get, set, and delete methods for the system objects. Many methods have dynamic names based on the name of the parent object. In the following methods described for the enterprise object, ObjectName indicates the name of the parent object.

Definition

class ObjectNameObject

Methods

- "EnterpriseObjectName"
- "countSystemObjectName"
- "deleteSystemObject"
- "deleteSystemObjectBean"
- "getEUID"
- "getSBRObjectName"
- "getStatus"
- "getSystemObjectName"
Web Services Helper Classes

- "getSystemObjectName"
- "getSystemObjectNameCount"
- "removeSystemObject"
- "removeSystemObjectBean"
- "setEUID"
- "setStatus"
- "setSystemObjectName"

EnterpriseObjectName

Description
This is the constructor method for the enterprise object class. You can instantiate this class to create a new instance of the enterprise object.

Syntax
new EnterpriseObjectName()

Parameters
None.

Returns
An instance of the enterprise object.

Throws
None.

countSystemObjectName

Description
This method counts the system objects (class SystemObjectName) in the enterprise object. Use this method to loop through the system objects.

Syntax
int countSystemObjectName()

Parameters
None.

Returns
An integer representing the number of system objects.

Throws
None.
deleteSystemObject

**Description**
This method marks a system object for deletion from an instance of EnterpriseObjectName. The system object is deleted when the enterprise object is updated.

**Syntax**
void deleteSystemObject(String system, String lid)

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>String</td>
<td>The processing code of the system associated with the system record to delete.</td>
</tr>
<tr>
<td>lid</td>
<td>String</td>
<td>The local ID of the system record in the specified system.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**Throws**
ObjectException

deleteSystemObjectBean

**Description**
This method marks a system object for deletion from an instance of EnterpriseObjectName. The system object is deleted when the enterprise object is updated.

**Syntax**
void deleteSystemObjectBean(String system, String lid)

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>String</td>
<td>The processing code of the system associated with the system record to delete.</td>
</tr>
<tr>
<td>lid</td>
<td>String</td>
<td>The local ID of the system record in the specified system.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**Throws**
ObjectException
Web Services Helper Classes

getEUID

Description
This method retrieves the EUID from an instance of EnterpriseObjectName.

Syntax
String getEUID()

Parameters
None.

Returns
A string containing the EUID of the enterprise object.

Throws
ObjectException

getSBOwnerName

Description
This method retrieves the SBR from an instance of EnterpriseObjectName.

Syntax
SBRObjectName getSBOwnerName()

Parameters
None.

Returns
The SBR object from the enterprise object.

Throws
None.

getStatus

Description
This method retrieves the status of the enterprise object.

Syntax
String getStatus()

Parameters
None.

Returns
The status of the enterprise object.

Throws
ObjectException

4-20 Oracle Healthcare Master Person Index Understanding Message Processing Reference
getSystemObjectName

Description
This method retrieves all of the system objects associated with the enterprise object.

Syntax
SystemObjectName[] getSystemObjectName()

Parameters
None.

Returns
A collection of system objects.

Throws
None.

getSystemObjectName

Description
This method retrieves a system object based on its index in a list of system objects.

Syntax
SystemObjectName getSystemObjectName(int index)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>int</td>
<td>The index of the system object to retrieve from a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>list of system objects.</td>
</tr>
</tbody>
</table>

Returns
The system object at the specified index.

Throws
None.

getSystemObjectNameCount

Description
This method returns the number of system objects in a collection of system objects.

Syntax
int getSystemObjectNameCount()

Parameters
None.
Returns
An integer indicating the number of system objects in an enterprise object.

Throws
None.

removeSystemObject

Description
This method removes a system object from an enterprise object. Use this to remove the system object from the in-memory instance of EnterpriseObjectName. This does not result in the deletion of the system object from the database (see "deleteSystemObject").

Syntax
void removeSystemObject(String system, String lid)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>String</td>
<td>The processing code of the system associated with the system object to remove.</td>
</tr>
<tr>
<td>lid</td>
<td>String</td>
<td>The local ID of the system record in the specified system.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
ObjectException

removeSystemObjectBean

Description
This method removes a system object from an enterprise object. Use this to remove the system object from the in-memory instance of EnterpriseObjectName. This does not result in the deletion of the system object from the database (see "deleteSystemObjectBean").

Syntax
void removeSystemObjectBean(String system, String lid)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>String</td>
<td>The processing code of the system associated with the system object to remove.</td>
</tr>
<tr>
<td>lid</td>
<td>String</td>
<td>The local ID of the system record in the specified system.</td>
</tr>
</tbody>
</table>
Returns
None.

Throws
ObjectException

setEUID

Description
This method sets the value of the EUID field in an enterprise object.

Syntax
void setEUID(String euid)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>euid</td>
<td>String</td>
<td>The EUID of the enterprise object</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
ObjectException

setStatus

Description
This method sets the status of the enterprise object.

Syntax
void setStatus(String status)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>String</td>
<td>The status of the enterprise object</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
ObjectException

setSystemObjectName

Description
This method sets a collection of system objects in an enterprise object.
Web Services Helper Classes

Syntax

void setSystemObjectName(SystemObjectName[] systemObjectName)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>systemObjectName</td>
<td>SystemObjectName</td>
<td>A collection of system objects for the enterprise object</td>
</tr>
</tbody>
</table>

Returns

None.

Throws

ObjectException

SystemObjectName

In order to run `executeMatch` or `executeMatchUpdate` from a client application, you must define a variable of the class type `SystemObjectName`, where `ObjectName` is the name of a parent object. This class is passed as a parameter to the `execute match` methods. The class contains a constructor method and several get and set methods for system fields. It also includes one field that specifies the value of the clear field character (for more information, see "Clear Field Indicator").

In the methods described in this section, `ObjectName` indicates the name of the parent object, `Child` indicates the name of a child object, and `Field` indicates the name of a field defined for the parent object.

Definition

```java
class SystemObjectName
```

Methods

- `SystemObjectName`
- `getClearFieldIndicator`
- `getField`
- `getObjectName`
- `setClearFieldIndicator`
- `setField`
- `setObjectName`

Inherited Methods

The following methods are inherited from `java.lang.Object`.

- equals
- hashcode
- notify
- notifyAll
- toString
Clear Field Indicator
The clear field indicator allows you to specify whether to treat a field in the parent object as null when performing an update from an external system. When an update is performed in the OHMPI application, empty fields typically do not overwrite the value of an existing field. You can specify to nullify a field that already has an existing value in the OHMPI application by entering an indicator in that field. This indicator is specified by the clear field indicator. By default, the clear field indicator is set to double-quotes (""), so if a field is set to double-quotes, that field will be blanked out in the master person index if the incoming record has a field value of null and the record being updated has a field value other than null. If you do not want to use this feature, set the clear field indicator to null.

SystemObjectName

Description
This method is the user-defined system class for the parent object. You can instantiate this class to create a new instance of the system class.

Syntax
new SystemObjectName()

Parameters
None.

Returns
An instance of the SystemObjectName class.

Throws
ObjectException

getClearFieldIndicator

Description
This method retrieves the value of the clear field indicator.

Syntax
Object getClearFieldIndicator()

Parameters
None.

Returns
The clear field indicator.

Throws
None.
**get Field**

**Description**
This method retrieves the value of the specified system field. There are getter methods for the following fields: LocalId, SystemCode, Status, CreateDateTime, CreateFunction, CreateUser, and UpdateUser.

**Syntax**
```java
String getField()
```
or
```java
Date getField()
```

**Parameters**
None.

**Returns**
The value of the specified field. The type of value returned depends on the field from which the value was retrieved.

**Throws**
ObjectException

**getObjectName**

**Description**
This method retrieves the system record JavaBean.

**Syntax**
```java
ObjectNameBean getObjectName()
```

**Parameters**
None.

**Returns**
A JavaBean containing the system record.

**Throws**
None.

**setClearFieldIndicator**

**Description**
This method sets the value of the clear field indicator. By default, this is set to double quotes ("").
### setField

**Description**
This method sets the value of the specified system field. There are setter methods for the following fields: LocalId, SystemCode, Status, CreateDateTime, CreateFunction, CreateUser, and UpdateUser.

**Syntax**
```java
void setField(value)
```

**Parameters**
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
<td>The value that should be entered into a field to indicate that any existing values should be replaced with null.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**Throws**
None.

### setObjectName

**Description**
This method sets the system object JavaBean.

**Syntax**
```java
void setObjectName(ObjectNameBean anObjectName)
```

**Parameters**
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anObjectName</td>
<td>ObjectNameBean</td>
<td>The JavaBean for the parent object.</td>
</tr>
</tbody>
</table>
Returns
None.

Throws
ObjectException

Parent Beans
A JavaBean is created to represent the parent object defined in the object structure of the OHMPI application. The methods in this class are generated based on the object structure and provide the ability to create a parent object Bean and to set or retrieve the field values for that JavaBean.

The name of the parent JavaBean class is the same as the name of the parent object, with the word "Bean" appended. For example, if the parent object in your object structure is "Person", the name of the associated parent Bean class is PersonBean. The methods in this class include a constructor method for the parent object Bean, get and set methods for each field defined for the parent object, and methods to manipulate child objects. Most methods have dynamic names based on the name of the parent object and the fields and child objects defined for that object. In the methods described in this section, ObjectName indicates the name of the parent object, Child indicates the name of a child object, and Field indicates the name of a field defined for the parent object.

Definition
final class ObjectNameBean

Methods
- 'ObjectNameBean'
- 'countChild'
- 'countChildren'
- 'countChildren'
- 'deleteChild'
- 'getChild'
- 'getChild'
- 'getField'
- 'getObjectNameId'
- 'setChild'
- 'setChild'
- 'setField'
- 'setObjectNameId'

Inherited Methods
The following methods are inherited from java.lang.Object.
- equals
- hashcode
notify
notifyAll
toString
wait()
wait(long arg)
wait(long timeout, int nanos)

ObjectNameBean

Description
This method is the user-defined object Bean class. You can instantiate this class to create a new instance of the parent object Bean class.

Syntax
new ObjectNameBean()

Parameters
None.

Returns
An instance of the parent object Bean.

Throws
ObjectException

countChild

Description
This method returns the total number of child objects contained in a system object. The type of child object is specified by the method name, such as countPhone or countAddress.

Syntax
int countChild()

Parameters
None.

Returns
An integer indicating the number of child objects in a collection.

Throws
None.
countChildren

Description
This method returns a count of the total number of child objects belonging to a system object.

Syntax
int countChildren()

Parameters
None.

Returns
An integer representing the total number of child objects.

Throws
ObjectException

countChildren

Description
This method returns a count of the total number of child objects of a specific type that belong to a system object.

Syntax
int countChildren(String type)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>String</td>
<td>The type of child object to count, such as Phone or Address.</td>
</tr>
</tbody>
</table>

Returns
An integer representing the total number of child objects of the specified type.

Throws
None.

deleteChild

Description
This method removes the specified child object from the system object. The type of child object to remove is specified by the name of the method, and the specific child object to remove is specified by its unique identification code assigned by the master person index application.

Syntax
void deleteChild(String ChildId)
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildId</td>
<td>String</td>
<td>The unique identification code of the child object to delete.</td>
</tr>
</tbody>
</table>

Returns

None.

Throws

ObjectException

getChild

Description

This method retrieves an array of child object JavaBeans. Each getter method is named according to the child objects defined for the parent object. For example, if the parent object contains a child object named "Address", the getter method for this object is named `getAddress`. A getter method is created for each child object in the parent object.

Syntax

```java
ChildBean[] getChild()
```

Parameters

None.

Returns

An array of JavaBeans containing the type of child object specified by the method name.

Throws

None.

getChild

Description

This method retrieves a child object JavaBean based on its index in a list of child objects. Each getter method is named according to the child objects defined for the parent object. For example, if the parent object contains a child object named "Address", the getter method for this field is named `getAddress`. A getter method is created for each child object in the parent object.

Syntax

```java
ChildBean[] getChild(int i)
```
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>int</td>
<td>The index of the child object to retrieve from a list of child objects.</td>
</tr>
</tbody>
</table>

Returns
A JavaBean containing the child object specified by the index value. The method name indicates the type of child object returned.

Throws
ObjectException

getField

Description
This method retrieves the value of the field specified in the method name. Each getter method is named according to the fields defined for the parent object. For example, if the parent object contains a field named "FirstName", the getter method for this field is named getFirstName.

Syntax
String getField()

Parameters
None.

Returns
The value of the specified field. The type of data returned depends on the data type defined in the object structure.

Throws
ObjectException

getObjectNameld

Description
This method retrieves the unique identification code (primary key) of the parent object, as assigned by the master person index application.

Syntax
String getObjectNameld()

Parameters
None.
Returns
A string containing the unique ID of the parent object.

Throws
ObjectException

setChild

Description
This method adds a child object to the system object.

Syntax
void setChild(int index, ChildBean child)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>integer</td>
<td>The index number for the new child object.</td>
</tr>
<tr>
<td>child</td>
<td>ChildBean</td>
<td>The JavaBean containing the child object to add.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
None.

setChild

Description
This method adds an array of child objects of one type to the system object.

Syntax
void setChild(ChildBean[] children)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>ChildBean[]</td>
<td>The array of child objects to add.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
None.
**setField**

**Description**
This method sets the value of the field specified in the method name. Each setter method is named according to the fields defined for the parent object. For example, if the parent object contains a field named "DateOfBirth", the setter method for this field is named `setDateOfBirth`. A setter method is created for each field in the parent object, including any fields containing standardized or phonetic data.

**Syntax**
```java
void setField(value)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>varies</td>
<td>The value of the field specified by the method name. The type of value depends on the field being populated.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**Throws**
ObjectException

**setObjectNameId**

**Description**
This method sets the value of the `ObjectNameId` field in the parent object.

**Note:** This ID is set internally by the OHMPI application. You should never set this field manually.

**Syntax**
```java
void setObjectNameId(String value)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
<td>The value of the <code>ObjectNameId</code> field.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**Throws**
ObjectException
Child Beans

A JavaBean is created to represent each child object defined in the object structure of the OMBF application. The methods in these classes provide the ability to create a child object bean and to set or retrieve the field values for that object bean.

The name of each child object JavaBean class is the same as the name of each child object, with the word "Bean" appended. For example, if a child object in your object structure is named "Address", the name of the corresponding child class is AddressBean. The methods in this class include a constructor method for the child object JavaBean, and get and set methods for each field defined for the child object. Most methods have dynamic names based on the name of the child object and the fields defined for that object. In the following methods, Child indicates the name of a child object and Field indicates the name of a field defined for the child object.

**Definition**

```java
final class ChildBean
```

**Methods**

- "ChildBean"
- "delete"
- "getField"
- "getChildId"
- "setField"
- "setChildId"

**Inherited Methods**

The following methods are inherited from `java.lang.Object`.

- equals
- hashcode
- notify
- notifyAll
- toString
- wait()
- wait(long arg)
- wait(long timeout, int nanos)

**ChildBean**

**Description**

This method is the constructor for the user-defined object JavaBean class. You can instantiate this class to create a new instance of the JavaBean.

---

*Note: These JavaBeans also include two countChildren methods, which are not currently used.*
Syntax

```java
new ChildBean();
```

Parameters
None.

Returns
An instance of the child object Bean.

Throws
ObjectException

delete

Description
This method removes the child object from its corresponding parent object. This is used with the `executeMatchUpdate` function to update a system object by deleting one of the child objects from the object being processed.

Syntax
```java
void delete();
```

Parameters
None.

Returns
None.

Throws
ObjectException

getField

Description
This method retrieves the value of the field specified in the method name. Each getter method is named according to the fields defined for the child object. For example, if the child object contains a field named "ZipCode", the getter method for this field is named `getZipCode`.

Syntax
```java
String getField();
```

Parameters
None.
Returns
The value of the specified field. The type of data returned depends on the data type defined in the object structure.

Throws
ObjectException

getChildId
Description
This method retrieves the unique identification code (primary key) of the object, as assigned by the master person index application.

Syntax
String getChildId()

Parameters
None.

Returns
A string containing the unique ID of the child object.

Throws
ObjectException

setField

Description
This method sets the value of the field specified in the method name. Each setter method is named according to the fields defined for the child object. For example, if the child object contains a field named "Address", the setter method for this field is named setAddress. A setter method is created for each field in the child object, including any fields containing standardized or phonetic data.

Syntax
void setField(value)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>varies</td>
<td>The value of the field specified by the method name. The type of value depends on the data type of the field being populated.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
ObjectException
setChildId

Description
This method sets the value of the ChildId field in the child object.

Note: This ID is set internally by the OHMPI application. You should never set this field manually.

Syntax
void setChildId(String value)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
<td>The value of the ChildId field.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
ObjectException

DestinationEO

This class represents an enterprise object involved in a merge. This is the enterprise object whose EUID was kept in the final merge result record. A DestinationEO object is used when unmerging two enterprise objects.

Definition
class DestinationEO

Methods
- "getEnterpriseObjectName"
- "setEnterpriseObjectName"

getEnterpriseObjectName

Description
This method retrieves the surviving enterprise object from a merge transaction in order to allow the records to be unmerged.

Syntax
EnterpriseObjectName getEnterpriseObjectName()

where ObjectName is the name of the parent object.

Parameters
None.
Returns
The surviving enterprise object from a merge transaction.

Throws
ObjectException

setEnterpriseObjectName

Description
This method sets the surviving enterprise object of a merge transaction.

Syntax
void setEnterpriseObjectName(EnterpriseObjectName eo)

where ObjectName is the name of the parent object.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eo</td>
<td>EnterpriseObjectName</td>
<td>The destination enterprise object to set.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
ObjectException

MergeObjectNameResult

This class represents the return values for a call to the merge-related functions of the ObjectNameEJB class, where ObjectName is the name of the parent object.

Definition
class MergeObjectNameResult

Methods
- "MergeObjectNameResult"
- "getDestinationEO"
- "getSourceEO"
- "setDestinationEO"
- "setSourceEO"

MergeObjectNameResult

Description
This is the constructor method for the MergeObjectNameResult class.
Syntax
new MergeObjectNameResult()

where ObjectName is the name of the parent object.

Parameters
None.

Returns
A new instance of MergeObjectNameResult.

Throws
None.

getDestinationEO

Description
This method retrieves the surviving enterprise object from a merge result.

Syntax
DestinationEO getDestinationEO()

where ObjectName is the name of the parent object.

Parameters
None.

Returns
The surviving enterprise object from a merge result.

Throws
None.

ggetSourceEO

Description
This method retrieves the nonsurviving enterprise object from a merge result.

Syntax
SourceEO getSourceEO()

Parameters
None.

Returns
The nonsurviving enterprise object from a merge result.

Throws
None.
setDestinationEO

Description
This method sets the surviving enterprise object in a merge result.

Syntax
void setDestinationEO(DestinationEO eom)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eom</td>
<td>DestinationEO</td>
<td>The surviving record in the merge result.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
None.

setSourceEO

Description
This method sets the non-surviving enterprise object in a merge result.

Syntax
void setSourceEO(SourceEO eom)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eom</td>
<td>SourceEO</td>
<td>The non-surviving record in the merge result.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
None.

SBRObjectName

This class represents an SBR object in an enterprise object. The SBR represents the data about an entity that is considered to be the most accurate information from all external systems. Many of the methods in this class are named based on the name of the parent object, which is represented by ObjectName in the method descriptions below.

Definition
class SBRObjectName
Methods
- "SBRObjectName"
- "SBRObjectName"
- "getObjectName"
- "getStatus"
- "setObjectName"
- "setStatus"

SBRObjectName

Description
This is a constructor method for an empty SBRObjectName class.

Syntax
```
new SBRObjectName()
```

Parameters
None.

Returns
A new instance of SBRObjectName.

Throws
None.

SBRObjectName

Description
This is a constructor method for an SBRObjectName class. It takes an SBR object as its input.

Syntax
```
new SBRObjectName(SBR sbr)
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbr</td>
<td>SBRObjectName</td>
<td>The SBR object to use to construct the new instance of SBRObjectName.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
ObjectException
**getObjectName**

**Description**
This method retrieves the parent object in the SBR.

**Syntax**
```java
ObjectNameBean getObjectName()
```

**Parameters**
None.

**Returns**
The `ObjectNameBean` object associated with the SBR.

**Throws**
ObjectException

**getStatus**

**Description**
This method retrieves the status of the SBR.

**Syntax**
```java
String getStatus()
```

**Parameters**
None.

**Returns**
The status of the SBR.

**Throws**
ObjectException

**setObjectName**

**Description**
This method sets the parent object in the SBR.

**Syntax**
```java
void setObjectName(ObjectNameBean anObjectName)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anObjectName</td>
<td>ObjectNameBean</td>
<td>The parent object bean to set in the SBR.</td>
</tr>
</tbody>
</table>

**Returns**
None.
Throws
ObjectException

setStatus

Description
This method sets the status of the SBR.

Syntax
void setStatus(String status)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>String</td>
<td>The status of the SBR.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
ObjectException

SearchObjectNameResult

This class represents the results of a search. A SearchObjectNameResult object (where ObjectName is the name of the parent object) is returned as a result of a call to "searchBlock" on page 4-14, "searchExact" on page 4-14, or "searchPhonetic" on page 4-15.

Definition
class SearchObjectNameResult

Methods
- "getEUID"
- "getComparisonScore"
- "getObjectName"
- "setEUID"
- "setComparisonScore"
- "setObjectName"

getEUID

Description
This method retrieves the EUID of a search result record.

Syntax
String getEUID()
Parameters
None.

Returns
A string containing an EUID.

Throws
None.

getComparisonScore
Description
This method retrieves the weight that indicates how closely a search result record matched the search criteria.

Syntax
Float getComparisonScore()

Parameters
None.

Returns
A comparison weight.

Throws
None.

genericObject

Description
This method retrieves an object bean for a search result record.

Syntax
ObjectNameBean genericObject

where ObjectName is the name of the parent object.

Parameters
None.

Returns
An object bean.

Throws
None.

setEUID

Description
This method sets the EUID of a search result record.
Web Services Helper Classes

Syntax

```java
void setEUID(String euid)
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>euid</td>
<td>String</td>
<td>The EUID to set.</td>
</tr>
</tbody>
</table>

Returns

None.

Throws

None.

setComparisonScore

Description

This method sets the weight that indicates how closely a search result record matched the search criteria.

Syntax

```java
void setComparisonScore(float score)
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>Float</td>
<td>The matching comparison score to set.</td>
</tr>
</tbody>
</table>

Returns

None.

Throws

None.

setObjectName

Description

This method sets an object bean as a search result record.

Syntax

```java
void setObjectName(ObjectNameBean bean)
```

where ObjectName is the name of the parent object.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bean</td>
<td>ObjectNameBean</td>
<td>The object bean to set.</td>
</tr>
</tbody>
</table>
SourceEO

This class represents an enterprise object involved in a merge. This is the enterprise object whose EUID was not kept in the final merge result record. A SourceEO object is used when unmerging two enterprise objects.

Definition

class SourceEO

Methods

- "getEnterpriseObjectName"
- "setEnterpriseObjectName"

getEnterpriseObjectName

Description

This method retrieves the nonsurviving enterprise object from a merge transaction in order to allow the records to be unmerged.

Syntax

EnterpriseObjectName getEnterpriseObjectName()

where ObjectName is the name of the parent object.

Parameters

None.

Returns

The nonsurviving enterprise object from a merge transaction.

Throws

None.

setEnterpriseObjectName

Description

This method sets the nonsurviving enterprise object from a merge transaction in order to allow the records to be unmerged.

Syntax

void setEnterpriseObjectName(EnterpriseObjectName eo)

where ObjectName is the name of the parent object.
Web Services Helper Classes

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eo</td>
<td>EnterpriseObject</td>
<td>The enterprise object to set.</td>
</tr>
</tbody>
</table>

Returns
None.

Throws
ObjectException

SystemObjectNamePK

This class represents the primary keys in a system object, which include the processing code for the originating system and the local ID of the object in that system. The class is named for the primary object. For example, if the primary object is named "Person", this class is named SystemPersonPK. If the primary object is named "Company", this class is named SystemCompanyPK. The methods in these classes provide the ability to create an instance of the class and to retrieve the system processing code and the local ID.

Definition

```java
class SystemObjectNamePK
```

where ObjectName is the name of the parent object.

Methods

- "systemObjectNamePK"
- "getLocalId"
- "getSystemCode"

SystemObjectNamePK

Description

This method is the user-defined system primary key object. This object contains a system code and a local ID. Use this constructor method to create a new instance of a system primary key object.

Syntax

```java
new SystemObjectNamePK()
```

where ObjectName is the name of the parent object.

Parameters
None.

Returns
An instance of the system primary key object.
Throws
None.

**getLocalId**

Description
This method retrieves the local identifier from a system primary key object.

Syntax
```java
String getLocalId()
```

Parameters
None.

Returns
A string containing a local identifier.

Throws
None.

**getSystemCode**

Description
This method retrieves the system's processing code from a system primary key object.

Syntax
```java
String getSystemCode()
```

Parameters
None.

Returns
A string containing the processing code for a system.

Throws
None.
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"

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 “Master Person Index Match Processing Logic Plug-ins” in Oracle Healthcare Master Person Index User’s Guide (Part Number E25242-01). For detailed information about the methods, see the Javadoc 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.
Custom Decision Point Logic

- 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 (Part Number E25249-01).
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"
- "Oracle Match Engine Match Types"

**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.

**Oracle Match Engine Match Types**

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

- "Person Match Types"
Oracle Match Engine Match Types

- "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

<table>
<thead>
<tr>
<th>MPI Wizard Match Type</th>
<th>Match Field File Standardization Type</th>
<th>mefa.xml Match Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonLastName</td>
<td>PersonName</td>
<td>LastName</td>
</tr>
<tr>
<td>PersonFirstName</td>
<td>PersonName</td>
<td>FirstName</td>
</tr>
</tbody>
</table>

Note: Only specify 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.

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: 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.
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 meta.xml file. If more than one field contains the street address, you can define the additional fields in the standardization structure in the meta.xml file after the wizard creates the configuration files.

Note: Only specify this type of matching for one street address field; otherwise, the wizard will create duplicate entries in the object structure.

### Table 6-2 BusinessName Match Types

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>Added to Match String</th>
<th>meta.xml Match Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>field_name_Name</td>
<td>The parsed and normalized version of the business name.</td>
<td>Yes</td>
<td>PrimaryName</td>
</tr>
<tr>
<td>field_name_NamePhon</td>
<td>The phonetic version of the business name.</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>field_name_OrgType</td>
<td>The parsed organization type of the business name.</td>
<td>Yes</td>
<td>OrgTypeKeyword</td>
</tr>
<tr>
<td>field_name_AssocType</td>
<td>The association type for the business.</td>
<td>Yes</td>
<td>AssocTypeKeyword</td>
</tr>
<tr>
<td>field_name_Industry</td>
<td>The name of the industry for the business.</td>
<td>Yes</td>
<td>IndustryTypeKeyword</td>
</tr>
<tr>
<td>field_name_Sector</td>
<td>The name of the industry sector (industries are a subset of sectors).</td>
<td>Yes</td>
<td>IndustrySectorList</td>
</tr>
<tr>
<td>field_name_Alias</td>
<td>An alias for the business name.</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>field_name.Url</td>
<td>The web site URL for the business.</td>
<td>Yes</td>
<td>Url</td>
</tr>
</tbody>
</table>

### Table 6-3 Address Match Types

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>Added to Match String?</th>
<th>meta.xml Match Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>field_name_HouseNo</td>
<td>The parsed street number of the address.</td>
<td>Yes</td>
<td>HouseNumber</td>
</tr>
<tr>
<td>field_name_StDir</td>
<td>The parsed and normalized street direction of the address.</td>
<td>Yes</td>
<td>StreetDir</td>
</tr>
<tr>
<td>field_name_StName</td>
<td>The parsed and normalized street name of the address.</td>
<td>Yes</td>
<td>StreetName</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>Added to Match String?</th>
<th>mefa.xml Match Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>field_numr_StPhon</td>
<td>The phonetic version of the street name.</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>field_numr_StType</td>
<td>The parsed and normalized street type of the address, such as Boulevard, Street, Drive, and so on.</td>
<td>Yes</td>
<td>StreetType</td>
</tr>
</tbody>
</table>

Table 6–3 (Cont.) Address Match Types
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.

- "Non-standard Web Services for Oracle Healthcare Master Person Index"
- "Standard Web Services for IHE PIXv3/PDQv3 (HL7 v3 based) Profiles"

### Non-standard Web Services for Oracle Healthcare Master Person Index

The following are non-standard web services for OHMPI.

- **executeMatch**
  Receives a system object and performs standardization and blocking. It then matches with existing records and creates new record if required. It does not handle updates to existing system objects.

- **executeMatchUpdate**
  Same as **executeMatch**, but it does handle updates to existing system objects in a master person index. It also creates a system object if there is none. This is the most widely used web service.

- **searchExact**
  Does the alphabetic search (not standardized) for a person.

- **searchPhonetic**
  Does the phonetic search on phoneticized fields for a person.

- **searchBlock**
  Fetches a sub-set of the master person index database and searches on that sub-set for a person.

- **getEnterpriseRecordByEUID**
  Returns an enterprise object (SBR and all associated system objects) associated for a given EUID.

- **getEnterpriseRecordByLID**
Returns an enterprise object (SBR and all associated system objects) associated with a given system and local ID in that system.

- **getSBR**
  Returns an SBR for a given EUID.

- **getEUID**
  Returns an EUID for a given system and local ID in that system.

- **updateSystemRecord**
  Updates the system record in the master person index's database to reflect the new values of the given modified system object. Note that this will not run any match.

- **updateEnterpriseRecord**
  Generally used after getEnterpriseObject to update it. Updates the enterprise record (SBR and all associated system objects) to reflect the new values of the given modified enterprise object. Note that this will not run any match.

- **addSystemRecord**
  Adds a system object to the enterprise object specified by the EUID.

- **mergeEnterpriseRecord**
  Merges two enterprise records based on the given EUIDs (specifies which is the source and which is the target).

- **getSystemRecordsByEUID**
  Returns all system records for a given EUID.

- **getSystemRecordsByEUIDStatus**
  Returns all system records for a given EUID, including the system object's status (active/inactive).

- **getLIDs**
  Returns system IDs and local IDs belonging to the given EUID.

- **getLIDsByStatus**
  Returns system IDs and local IDs belonging to the given EUID + system object's status (active/inactive).

- **lookupLIDs**
  Returns an array of system object keys with the given status belonging to the given destination system whose EUID matches the source system code / LID.

- **mergeSystemRecord**
  Merges the two local IDs for a given system. Keys may belong to a single enterprise object, or may belong to two different enterprise objects.

- **activateSystemRecord**
  Turns an inactive system object back to active status.

- **activateEnterpriseRecord**
  Turns an inactive enterprise object back to active status.

- **deactivateSystemRecord**
Deactivates a system object based on a given key (system + local ID). Note that this is different than `deleteSystemObject` in that the record is not removed from the database, only its status is changed.

- `deactivateEnterpriseRecord` Deactivates an enterprise object based on given EUID.
- `transferSystemRecord` Transfers a system object from one enterprise object to another enterprise object.
- `lookupPotentialDuplicate` Returns an array of potential duplicate records based on search criteria.

### Standard Web Services for IHE PIXv3/PDQv3 (HL7 v3 based) Profiles

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/](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.