SUN SEEBEYOND

eINDEX™ SINGLE PATIENT VIEW REFERENCE GUIDE

Release 5.1.1

Sun microsystems

Version 20060612225308

Contents

List of Tables

Chapter 1

Introduction

This chapter provides an overview of this guide and the conventions used throughout, as well as a list of supporting documents and information about using this guide.

What's in This Chapter

1.1 About eIndex Single Patient View

Overview

The Sun SeeBeyond eIndex™ Single Patient View (eIndex SPV) provides a flexible framework to design and configure an enterprise-wide person master index that creates a single view of person information. eIndex SPV maintains the most current information about the people who participate throughout your organization and links information from different locations and computer systems. eIndex SPV provides accurate identification of patients throughout your healthcare enterprise, and cross-references a patient's local IDs using an enterprise-wide unique identification number (EUID). eIndex SPV also ensures accurate patient data by identifying potential duplicate records and providing the ability to merge or resolve duplicate records. All patient information is centralized in one shared index, enabling eIndex SPV to integrate data throughout the enterprise while allowing local systems to continue operating independently.

In eIndex SPV, you define the data structure of the information to be stored and cross-referenced. In addition, you define the logic that determines how data is updated, standardized, weighted, and matched. The structure and logic you define is stored in a group of XML configuration files, which are predefined but can be customized to meet your processing requirements. These files are defined within the context of an eGate Project and can be modified using the XML editor provided in the Enterprise Designer.

Features

eIndex SPV provides features and functions that allow you to customize the data structure, database, and logic of the master person index. eIndex SPV provides the following features:

- **Rapid Development** - eIndex SPV allows for rapid and intuitive development of a master person index, providing a predefined template that can easily be configured for your use.
- **Automated Component Generation** - eIndex SPV can regenerate scripts that create the appropriate database schemas and an eGate Object Type Definition (OTD) based on the changes you make to the object structure.
- **Configurable Survivor Calculator** - eIndex SPV 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 custom survivor strategies for the master index.
- **Flexible Architecture** - eIndex SPV provides a flexible platform that allows you customize the object structure, allowing you to design an application that specifically meets your data processing needs.
- **Configurable Matching Algorithm** - eIndex SPV provides standard support for the Sun SeeBeyond Match Engine (SBME). In addition, you can plug in a custom matching algorithm to the master index.
- **Custom Java API** - eIndex SPV generates a Java API that is customized to the object structure you define. You can call the methods in this API in the Collaborations that define the transformation rules for data processed by the master index.

1.2 What's New in This Release

This release provides added support for the Oracle 10g database platform as well as performance enhancements, standardization enhancements, and general maintenance fixes. For complete information about the changes included in this release, see the *eIndex Single Patient View Release Notes*.

1.3 About This Document

This guide provides comprehensive information about the database structure, the Java API, and message processing for eIndex SPV. As a component of the Java Composite Application Platform Suite, eIndex SPV helps you integrate information from disparate systems throughout your organization. This guide describes how messages are processed through the master index, provides a reference for the dynamic Java API, and describes the database structure. The master index is highly customizable, so your implementation might differ from some of the descriptions contained in this guide.

This guide is intended to be used with the *Sun SeeBeyond eIndex Single Patient View Configuration Guide* and the *Sun SeeBeyond eIndex Single Patient View User's Guide*, which provides information about basic components and features of eIndex SPV.

1.3.1 What's in This Document

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

- **Chapter 1 "Introduction"** gives a general preview of this document—its purpose, scope, and organization—and provides sources of additional information.
- **Chapter 2 "Understanding Operational Processes"** gives an overview of how inbound and outbound messages are processed, and includes information about how certain configuration attributes affect processing.
- **Chapter 3 "The Database Structure"** describes the database structure and how the structure is defined based on the object structure definition. It also provides a sample database diagram.
- **Chapter 4 "Working with the Java API"** gives implementation information about the eIndex SPV Java API, and provides a reference of the dynamic methods created for the method OTD and eInsight integration.
- **Appendix A "Inbound Message Processing with Custom Logic"** describes where the execute match functions check for custom logic and how that logic affects match processing.

1.3.2 Scope

This guide provides information about message processing in an eIndex SPV master index system and about the eIndex SPV Java API. The API is designed to help you transform data and transfer the information into and out of the master index database using eGate Collaborations, Services, and eWays. This guide also provides an overview of the data processing flow, based on the sample Project, and describes the database structure.

This guide provides information about the Java API Library, but does not serve as a complete reference (a complete reference is provided in the Javadocs for eIndex SPV). This guide compliments the *Sun SeeBeyond eIndex Single Patient View User's Guide*, the *Sun SeeBeyond eIndex Single Patient View Configuration Guide*, and the eIndex SPV Javadocs. Once you understand the default processing, you can configure eIndex SPV for your custom data and processing requirements.

This guide does not explain how to install eIndex SPV, or how to implement an eIndex SPV Project. For a list of publications that contain this information, see **"Related Documents" on page 12**.

1.3.3 Intended Audience

Any user who works with the connectivity components or uses the Java API should read this guide. A thorough knowledge of eIndex SPV is not needed to understand this guide. It is presumed that the reader of this guide is familiar with the eGate environment and GUIs, eGate Projects, Oracle database administration, and the Java programming language. The reader should also be familiar with the data formats used by the systems linked to the master index, the operating system(s) on which eGate and the master index database run, and current business processes and information system (IS) setup.

1.3.4 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

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

1.3.5 Screenshots

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

1.3.6 Related Documents

Sun has developed a suite of user's guides and related publications that are distributed in an electronic library. The following documents might provide information useful in creating your customized index. In addition, complete documentation of the eIndex SPV Java API is provided in Javadoc format.

- *Sun SeeBeyond eIndex Single Patient View User's Guide*
- *Sun SeeBeyond eIndex Single Patient View Configuration Guide*
- *Implementing the Sun SeeBeyond Match Engine with eIndex SPV*
- *Sun SeeBeyond eGate Integrator User's Guide*
- *Sun SeeBeyond eGate Integrator System Administration Guide*

1.4 Sun Microsystems, Inc. Web Site

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

http://www.sun.com

1.5 Documentation Feedback

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

CAPS_docsfeedback@sun.com

Chapter 2

Understanding Operational Processes

eIndex SPV uses a custom Java API library and the eGate Integrator to transform and route data into and out of the master index 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 master index system.

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

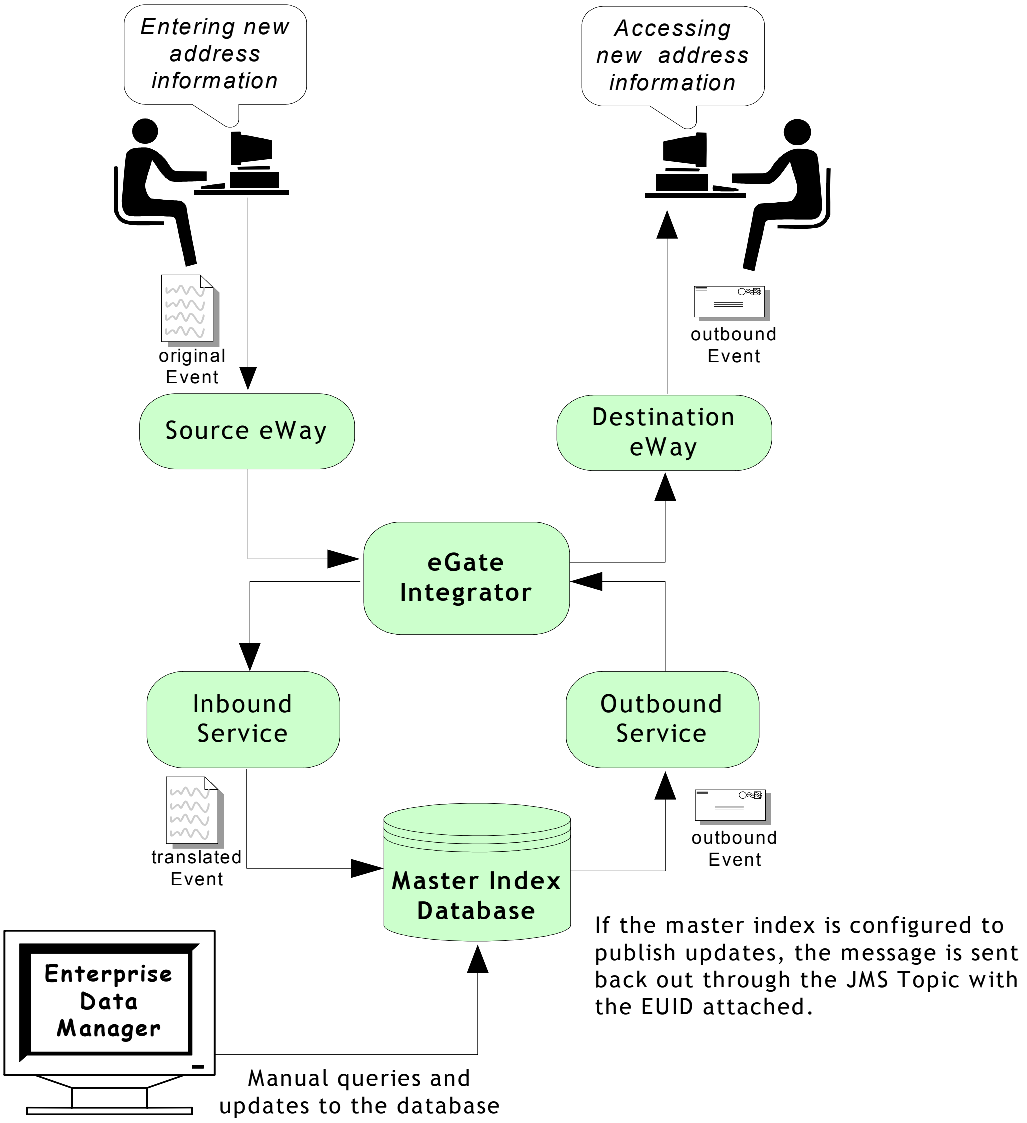
What's in This Chapter

2.1 Learning About Message Processing

This section of the chapter provides a summary of how inbound and outbound messages can be processed in an eIndex SPV environment. eIndex SPV cross-references records stored in various computer systems of an organization and identifies records that might represent or do represent the same patient. eIndex SPV uses the eGate Integrator, along with the connectivity components available through eGate, to connect to and share data with these external systems.

Figure 1 on page 15 illustrates the flow of information through a master index that includes a JMS Topic to which updates to the index are published.

Figure 1 Master Index Processing Flow


Entering new address information
Accessing new address information
original Event
outbound Event
Source eWay
Destination eWay
eGate Integrator
Inbound Service
Outbound Service
translated Event
Master Index Database
outbound Event
If the master index is configured to publish updates, the message is sent back out through the JMS Topic with the EUID attached.
Enterprise Data Manager
Manual queries and updates to the database

2.2 Inbound Message Processing

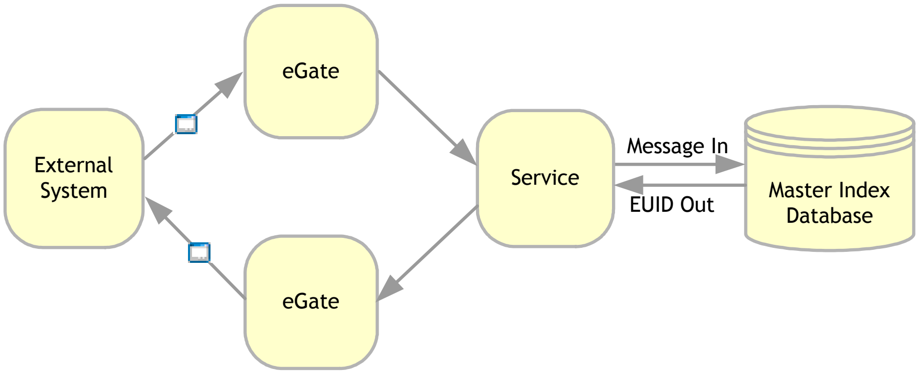
An inbound message refers to the transmission of data from external systems to the eGate Integrator and then to the master index database. These messages may be sent into the database via a number of Services. Inbound messages can be stored in journal files and tracked in the eGate log files. The steps below describe how inbound messages are processed.

1 Messages are created in an external system, and the enveloped message is transmitted to eGate via that system's eWay.

2. eGate identifies the message and the appropriate Service to which the message should be sent. The message is then routed to the appropriate Service for processing.
3. The message is modified into the appropriate format for the master index database, and validations are performed against the data elements of the message to ensure accurate delivery. The message is validated using the Java code in the Service's Collaboration and other information stored in the eIndex SPV configuration files.
4. If the message was successfully transmitted to the database, the appropriate changes to the database are processed.
5. After the master index processes the message, an enterprise-wide universal identifier (EUID) is returned (for either a new or updated record). That EUID can be sent back out through a different Service to the external system. Alternatively, the entire updated message can be published using the outbound OTD (see **"Outbound Message Processing" on page 22**).

Figure 2 below illustrates the flow of a message inbound to eIndex SPV.

Figure 2 Inbound Message Processing Data Flow



2.2.1 About Inbound Messages

The format of inbound messages is defined by the inbound OTD, located in the client Project for each external system. The inbound messages can either conform to the required format for eIndex SPV, or they can be mapped to the correct format in the Collaboration. The required format depends on how the object structure of the master index is defined (in the Object Definition file of the eIndex SPV Project).

In addition to the objects and fields defined in the Object Definition file, you can include standard eIndex SPV fields. For example, you must include the system and local ID fields, and you can also include transaction information, such as the date and time of the transaction, the transaction type, user ID, and so on. The inbound OTD in the eIndex SPV sample client Project includes system and local ID fields as well as transactional information fields. If incoming messages do not contain transactional information, eIndex SPV applies default values to certain fields (for example, the user ID defaults to "eGate" and the date and time fields default to the date and time that eIndex SPV processes the transaction).

2.2.2 The Default Inbound OTD

This section describes the default format of the data to be inserted into the eIndex SPV database. This format follows the format of the default object structure defined in the Object Definition file of the eIndex SPV Project. You can translate the data from external systems into this format using the Collaborations of the external systems. You can also modify the default OTD (in the eIndexClient Project) to use a different format if needed.

Formatting Guidelines

The default OTD contains two primary nodes: EVENT and REC. The EVENT node contains transactional information, and the REC node contains information about the person. The REC node structure should be based on the object structure defined in the Object Definition file. In order to comply with the sample OTD, the format of the data being transmitted into the eIndex SPV database needs to be reformatted as follows:

- Each record consists of two types of information: Transaction details and record details. These are delimited by a pair of angled brackets (<>).
- The records must be delimited. Each segment is separated by an ampersand (&), each field is separated by a pipe (|), and each sub-field is separated by a caret (^). When a field can repeat, each repetition is separated by a tilde (~). There are four segments, which appear as follows:

```
EVNT segment <> ID segment & DEMO Segment & AUX segment <>
```

For information about each field, see Tables 2 and 3. Note that most fields in eIndex SPV are configurable, so you are not restricted to the fields listed in the table.

Note: *The OTD should be reviewed for each site to simplify where applicable. For example, fields for which the sending systems do not collect data can be removed.*

Transaction Details

The following table describes the transaction details portion of the inbound OTD structure. When a field is required, that means the field must exist in the inbound message but it can be empty. Fields that cannot be null are determined by the object structure in the Object Definition file of the eIndex SPV Project.

Table 2 Default Inbound Message Structure - Transaction Information

Field	Description	Repeating?	Required?
SegmentId	“EVNT”	No	Yes
MessageId	Always leave this field blank. eIndex SPV determines the message ID.	No	Yes
EventTypeCode	Always leave this field blank. eIndex SPV automatically determines the transaction type.	No	Yes
UserId	The user ID of the user who performed the transaction.	No	Yes

Table 2 Default Inbound Message Structure - Transaction Information

Field	Description	Repeating?	Required?
AssigningSystem	The system code for the system on which the transaction was performed.	No	Yes
Source	The source code of the application on which the transaction was performed.	No	Yes
Department	The department code for the transaction.	No	Yes
TerminalId	The ID of the terminal on which the transaction was performed.	No	Yes
DateOfEvent	The date the transaction occurred in the format **YYYY-MM-DD**.	No	Yes
TimeOfEvent	The time the transaction occurred in the format **HH:MM:SS** using a 24-hour clock (for example, 23:59:59).	No	Yes

Record Details

The following table describes the main message portion of the inbound OTD structure.

Table 3 Default Inbound Message Structure - Record Information

Field	Description	Repeating?	Required?
SegmentId	**"ID"**	**No**	**Yes**
EUID	Leave this field blank. eIndex SPV determines the EUID after it processes the message.	No	Yes
LocalId	The patient's local identifier in a specified system. This field has two sub-fields: ♦ **Lid**: The local ID assigned to the person in the system of origin. ♦ **System**: The processing code of the system of origin. For example, if the local ID 12345 was assigned within the system SeeBeyond (with a processing code of SBYN), this field should appear as follows: |12345^SBYN|	No	Yes (both sub-fields are required)
NonUniqueId	The person's auxiliary identifiers. This node contains a repeating field, "NUI", that contains two sub-fields: ♦ **Id**: An auxiliary ID of the specified type. ♦ **Type**: The type of auxiliary ID specified. For example, if a person's account number is 003487 and the type code for account is ACCT, this field should appear as follows: |003487^ACCT| ***Note:*** *If auxiliary ID information is included, then both an ID and an ID type must be included.*	Yes (the NUI field is repeating)	Yes (the NUI field is required, but the sub-fields are not)

Table 3 Default Inbound Message Structure - Record Information

Field	Description	Repeating?	Required?
SegmentId	**"DEMO"**	**No**	**Yes**
PersonCategory	The code for the person category to which the person is assigned.	No	Yes
PersonName	The name of the person. This field consists of five sub-fields: ♦ **LastName**: The person's last name. ♦ **FirstName**: The person's first name. ♦ **MiddleName**: The person's middle name. ♦ **Title**: The processing code of the person's title. ♦ **Suffix**: The processing code of the person's suffix to their name. ***Note:*** *The last, first, and middle names are required, but the title and suffix are not.*	No	Yes
PersonAlias	The alias names for the person. This nodes consists of a repeating field, "PA", that includes three sub-fields: ♦ **LastName**: The alias last name. ♦ **FirstName**: The alias first name. ♦ **MiddleName**: The middle name of the alias.	Yes (the PA field is repeating)	Yes (the PA field is required, but the sub-fields are not)
AltName	Alternative names associated with this person. This field consists of five sub-fields: ♦ **MaidenName**: The person's maiden name. ♦ **SpouseName**: The name of the person's spouse. ♦ **MotherName**: The name of the person's mother. ♦ **FatherName**: The name of the person's father. ♦ **MotherMaiden**: The maiden name of the person's mother.	No	Yes (the field is required, but the sub-fields are not)
DateOfBirth	The person's date of birth, in **YYYY-MM-DD** format.	No	Yes
TimeOfBirth	The time the person was born, in **HH:MM:SS** format on a 24-hour clock.	No	Yes
Sex	The table code of the person's gender.	No	Yes
MaritalStatus	The table code of the person's marital status.	No	Yes
SSN	The person's social security number, with no punctuation.	No	Yes

Table 3 Default Inbound Message Structure - Record Information

Field	Description	Repeating?	Required?
DriverLicense	The driver license details for the person. This has two sub-fields: ♦ **StateCountry**: The state or country that issued the drivers license. ♦ **LicenseNumber**: The driver license number.	No	Yes (the field is required, but the sub-fields are not)
Race	The table code of the person's race.	No	Yes
EthnicGroup	The table code of the person's ethnic group.	No	Yes
Nationality	The table code of the person's nationality.	No	Yes
Religion	The table code of the person's religion.	No	Yes
Language	The table code of the language spoken by the person.	No	Yes
Death	Death information about the person. This field consists of three sub-fields: ♦ **DeathFlag**: An indicator of whether the person is deceased. Should be Y if deceased. ♦ **DateOfDeath**: If deceased, the date of death in **YYYY-MM-DD** format. ♦ **DeathCertificateNumber**: The ID number on the death certificate.	No	Yes (the field is required, but the sub-fields are not)
BirthPlace	The location in which the person was born. This field consists of three sub-fields: ♦ **BirthCity**: The city in which the person was born. ♦ **BirthState**: The state in which the person was born. ♦ **BirthCountry**: The country code where the person was born.	No	Yes (the field is required, but the sub-fields are not)
VIP	The table code of the person's VIP status.	No	Yes
VeteranStatus	The table code of the person's veteran status.	No	Yes
Military	The military details for the person. This field consists of three sub-fields: ♦ **MilitaryStatus**: The code of the person's military status. ♦ **RankGrade**: The person's military rank or grade. ♦ **MilitaryBranch**: The military branch in which the person has served.	No	Yes (the field is required, but the sub-fields are not)
Citizenship	The citizenship for the person.	No	Yes

Table 3 Default Inbound Message Structure - Record Information

Field	Description	Repeating?	Required?
Pension	The pension details for the person. This field consists of two sub-fields: ♦ **PensionNumber**: The person's pension card number. ♦ **ExpirationDate**: The expiration date of the pension card in **YYYY-MM-DD** format.	No	Yes (the field is required, but the sub-fields are not)
RepatriationNumber	The person's repatriation number.	No	Yes
DistrictOfResidence	The code of the district of residence in which the person resides.	No	Yes
LgaCode	The LGA code for the person.	No	Yes
Address	Address information for the person. This node consists of one field, "ADDR", that includes ten sub-fields: ♦ **AddressType**: The table code for the type of address. ♦ **Street1**: The first line of the street address. ♦ **Street2**: The second line of the street address. ♦ **Street3**: The third line of the street address. ♦ **Street4**: The fourth line of the street address. ♦ **City**: The city or suburb of the address. ♦ state_or_province: State or province ♦ **Zip**: The zip code of the address. ♦ **ZipExt**: The zip code extension of the address. ♦ **County**: The table code of the county in which the address is located. ♦ **Country**: The table code of the address's country. ***Note:*** *If address information is included in a message, by default the* ***AddressType*** *and* ***Street1*** *fields are required. Any other required fields are determined by the Object Definition file.*	Yes (the ADDR field is repeating)	Yes (the field is required, but the sub-fields are not)

Table 3 Default Inbound Message Structure - Record Information

Field	Description	Repeating?	Required?
Phone	Telephone information for the person. This node consists of one field, “PH”, that includes three sub-fields: ♦ **PhoneType**: The table code of the telephone type. ♦ **PhoneNumber**: The telephone number, with no punctuation characters. ♦ **PhoneExt**: The extension to the telephone number. *Note: If telephone information is included in a message, the **Type** and **PhoneNumber** fields must be present for each telephone number.*	Yes (the PH field is repeating)	Yes (the field is required, but the sub-fields are not)
SegmentId	**“AUX”**	**No**	**No**
Class (CL)	This field includes five miscellaneous sub-fields that can contain strings up to 20-characters.	Yes (maximum of five)	Yes (the field is required if there is an AUX segment, but the sub-fields are not)
String (STR)	Additional strings for site-specific purposes. This field contains 10 sub-fields. The first six are a maximum of 40 characters. Sub-fields seven to nine are a maximum of 100 characters. The tenth sub-field is a maximum of 255 characters.	Yes (maximum of ten)	Yes (see above)
Date (DT)	This field includes five miscellaneous date sub-fields in **YYYY-MM-DD** format.	Yes (maximum of five)	Yes (see above)

Sample Inbound Message

Below is a sample data record that follows the default format described in the previous tables.

```
EVNT|||JJONES|CBMC|CBMC|||2003-06-15|10:20:24<>
ID||239487209^CBMC|23438742^ACCT&DEMO|P|WARREN^ELIZABETH^JUNE
^PHD^|MILLER^ELIZABETH^J|MILLER^ANDREW^JULIE^MARK^MARTIN|19
60-05-14|15:01:08|F|M|555-44-4555|^|W|28||AG|ENGL|^^|^^|N|N|^^|USA|
^||||H^2347 SHORELINE DRIVE^UNIT 3^^^SHEFFIELD^CT^09877^^ CAPE
BURR^UNST~O^1490 WAYFIELD ROAD^FLOOR 5^SUITE 519^^CAPE
BURR^CT^09877^^^UNST|CH^9895557811^~CB^9895553214^1212&AUX|~~~~|
STANDARD MEMBERSHIP~~~~~~~~~|1999-09-12~2000-12-15~~~<>
```

2.3 Outbound Message Processing

An outbound message refers to the transmission of data from the master index database to any external system. Messages can be transmitted from the master index in two ways. The first way is by transmitting the output of **executeMatch** (an EUID). This is described earlier in **"Inbound Message Processing" on page 15**, and is only used for messages received from external systems.

The second way is by publishing updates from the master index 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 Enterprise Data Manager, the master index generates outbound messages in the format of the outbound OTD.

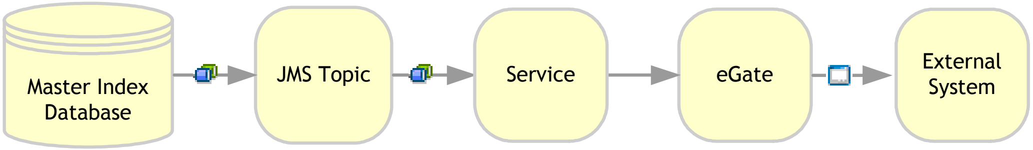
This section describes how the second type of outbound message is processed. A JMS Topic must be defined in the Connectivity Maps for the eIndex SPV server Project and the appropriate client Projects for this type of processing to occur.

1. When a message is received from an external system or data is entered through the EDM, the master index processes the information and generates an XML message, which is sent to the JMS Topic that is configured to publish messages from the master index.
2. Messages published by the JMS Topic are processed through a Service whose Collaboration uses the eIndex SPV outbound OTD. This Service modifies the message into the appropriate format.
3. eGate identifies the message and the external systems to which it should be sent, and then routes the message for processing via an external system eWay.

Note: *Outbound messages are stored and tracked in the eGate journal and log files.*

Figure 3 below illustrates the flow of data for a message outbound from the master index.

Figure 3 Outbound Message Processing Data Flow



2.3.1 About Outbound Messages

When you customize the object definition and generate the eIndex SPV application, an outbound OTD is created, the structure of which is based on the object definition. This OTD is used to publish changes in the master index 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 Collaboration and Service to output the data, or you can process all updates in the master index through a JMS Topic using the outbound OTD.

Outbound OTD Structure

The outbound OTD for eIndex SPV is named “OUTPerson”. This OTD contains eight primary nodes: Event, ID, SBR, and the standard Java methods **marshal**, **unmarshal**, **marshalToString**, **unmarshalFromString**, **marshalToBytes**, **unmarshalFromBytes**, and **reset**. 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 node is the portion of the OTD created from the Object Definition file. By default, the outbound OTD publishes messages in XML format. Table 4 describes the components of the SBR portion of the outbound OTD.

Table 4 Outbound OTD SBR Nodes

Node	Description
EUID	The EUID of the record that was inserted or modified.
Status	The status of the record.
CreateFunction	The date the record was first created.
CreateUser	The logon ID of the user who created the record.
UpdateSystem	The processing code of the external system from which the updates to an existing record originated.
ChildType	The name of the parent object.
CreateSystem	The processing code of the external system from which the record originated.
UpdateDateTime	The date and time the record was last updated.
CreateDateTime	The date and time the record was created.
UpdateFunction	The type of function that caused the record to be modified.
RevisionNumber	The revision number of the record.
UpdateUser	The logon ID of the user who last updated the record.
SystemObject	The patient's local identifier in a specified system. This field has three sub-fields: ♦ **LID**: The local ID assigned to the person in the system of origin. ♦ **System**: The processing code of the system of origin. ♦ **Status**: The status of the local ID in the enterprise record.

Table 4 Outbound OTD SBR Nodes

Node	Description
Person	The fields in this node are defined by the object structure (as defined in the Object Definition file). It is named by the parent object (Person) and contains all fields and child objects defined in the structure. This section varies depending on the customizations made to the object structure.

Outbound Message Trigger Events

When outbound messaging is enabled, the following transactions automatically generate an outbound message that is sent to the JMS Topic (if a JMS Topic has been incorporated into the eIndex SPV 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
- 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 eIndex SPV based on the default configuration. Your outbound messages might appear differently depending on how you configure the client Project connectivity components.

```
<?xml version="1.0" encoding="UTF-8"?>
<OutMsg Event="UPD" ID="00000000000000044005">
<SBR EUID="1000008001"  Status="active" CreateFunction="Add"
ChildType="Person" CreateSystem="System" UpdateFunction="Update"
RevisionNumber="5" CreateUser="eview" UpdateSystem="System"
UpdateDateTime="12/16/2003 17:40:44" CreateDateTime="12/16/2003
17:36:58" UpdateUser="eview">
<SystemObject SystemCode="CBMC" LID="434900094" Status="active">
</SystemObject>
<Person PersonId="00000000000000017000" PersonCatCode="PT"
LastName="WRAND" FirstName="ELIZABETH" MiddleName="SU" Suffix=""
Title="PHD" DOB="12/12/1972 00:00:00" Death="" Gender="F" MStatus="M"
```

```
SSN="555665555" Race="B" Ethnic="23" Religion="AG" Language="ENGL"
SpouseName="MARCUS" MotherName="TONIA" MotherMN="FLEMING"
FatherName="JOSHUA" Maiden="TERI" PobCity="KINGSTON" PobState=""
PobCountry="JAMAICA" VIPFlag="N" VetStatus="N"
FnamePhoneticCode="E421" LnamePhoneticCode="RAN"
MnamePhoneticCode="S250" MotherMNPhoneticCode="FLANANG"
MaidenPhoneticCode="TAR" SpousePhoneticCode="M622"
MotherPhoneticCode="T500" FatherPhoneticCode="J200"
DriversLicense="CT111333111" DriversLicenseSt="CT" Dod=""
DeathCertificate="" Nationality="USA" Citizenship="USA" PensionNo=""
PensionExpDate="" RepatriationNo="" DistrictOfResidence="" LgaCode=""
MilitaryBranch="NONE" MilitaryRank="NONE" MilitaryStatus="NONE"
DummyDate="" Class1="" Class2="" Class3="" Class4="" Class5=""
String1="ADMINISTRATION" String2="LEVEL 5" String3="EWRAY@HERE.MED"
String4="" String5="" String6="" String7="" String8="" String9=""
String10="" Date1="12/15/1995 00:00:00" Date2="12/31/2005 00:00:00"
Date3="" Date4="" Date5="" StdFirstName="ELIZABETH"
StdLastName="WRAND" StdMiddleName="SUSAN">
<Phone PhoneId="00000000000000011001" PhoneType="CC"
Phone="9895558768" PhoneExt="">
</Phone>
<Phone PhoneId="00000000000000011000" PhoneType="CH"
Phone="9895554687" PhoneExt="">
</Phone>
<Alias AliasId="00000000000000016001" LastName="TERI"
FirstName="ELIZABETH" MiddleName="SU" LnamePhoneticCode="TAR"
FnamePhoneticCode="E421" MnamePhoneticCode="S250"
StdFirstName="ELIZABETH" StdLastName="TERI" StdMiddleName="SUSAN">
</Alias>
<Address AddressId="00000000000000011001" AddressType="H"
AddressLine1="1220 BLOSSOM STREET" AddressLine2="UNIT 12"
AddressLine3="" AddressLine4="" City="SHEFFIELD" StateCode="CT"
PostalCode="09877" PostalCodeExt="" County="CAPEBURR"
CountryCode="UNST" HouseNumber="1220" StreetDir=""
StreetName="BLOSSOM" StreetNamePhoneticCode="BLASAN" StreetType="St">
</Address>
<AuxId AuxIdId="00000000000000010000" AuxIdDef="ACCT" Id="1155447">
</AuxId>
<AuxId AuxIdId="00000000000000010001" AuxIdDef="INS" Id="55488877">
</AuxId>
<Comment CommentId="00000000000000009000" CommentCode="1A"
EnterDate="12/12/2003 00:00:00" CommentText="UPDATED CLEARANCE TO
LEVEL 5">
</Comment>
</Person>
</SBR>
</OutMsg>
```

2.4 Inbound Message Processing Logic

When records are transmitted to the master index, 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 EDM uses **executeMatchGui**. For more information about how these methods differ, refer to the Javadocs provided with eIndex SPV.

In the sample Project configuration, these processes are defined in the Collaboration using the functions defined in the customized method OTD. The steps performed by that 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 Collaboration and Connectivity Map.

The steps outlined below refer to the following parameters and element in the eIndex SPV Threshold file (these are described in the *Sun SeeBeyond eIndex Single Patient View Configuration Guide*).

- OneExactMatch parameter
- SameSystemMatch parameter
- MatchThreshold parameter
- DuplicateThreshold parameter
- **update-mode** element

Important: *There are several decision points in the match process that can be defined by custom logic using custom plug-ins (for more information, see "Customizing Match Processing Logic" in the Sun SeeBeyond eIndex Single Patient View User's Guide). The decision points are not listed in the below steps, which instead define the default processing logic.* **Appendix A "Inbound Message Processing with Custom Logic"** *provides the same steps as below with the decision points included.*

1. When a message is received by the master index, 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 patient. Using the EUID of the existing record, the master index performs an update of the record's information in the database.
   - If the update does not make any changes to the patient's information, no further processing is required and the existing EUID is returned.
   - If there are changes to the patient'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 re-evaluated 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.

   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 master index 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 the Sun SeeBeyond eIndex Single Patient View Configuration Guide.*

6 When records are checked for same system entries, the master index 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 found and same system matching is set to *false*, it is assumed that the two records represent the same patient. Using the EUID of the existing record, the master index performs an update, following the process described in Step 2 earlier.
- If no local ID is found, it is assumed that the two records represent the same patient and an assumed match occurs. Using the EUID of the existing record, the master index 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, see the *Sun SeeBeyond eIndex Single Patient View Configuration Guide*).

The flow charts on the following pages provide a visual representation of the processes performed in the default configuration. Figures 4 and 5 represent the primary flow of information. Figure 6 expands on update procedures illustrated in Figures 4 and 5.

Figure 4 Inbound Message Processing in the Sample Project

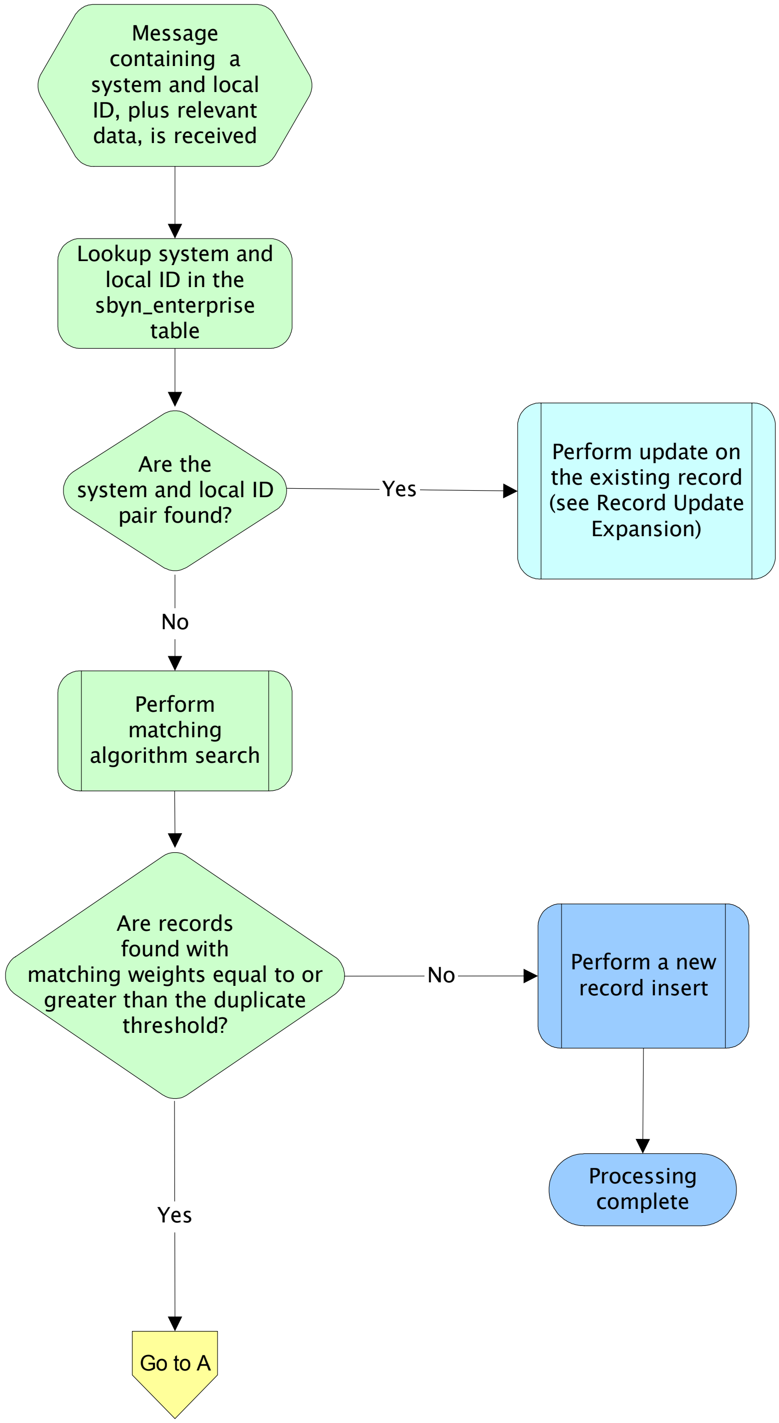

Message containing a system and local ID, plus relevant data, is received
Lookup system and local ID in the sbyn_enterprise table
Are the system and local ID pair found?
Yes
Perform update on the existing record (see Record Update Expansion)
No
Perform matching algorithm search
Are records found with matching weights equal to or greater than the duplicate threshold?
No
Perform a new record insert
Processing complete
Yes
Go to A

Figure 5 Inbound Message Processing (cont'd)

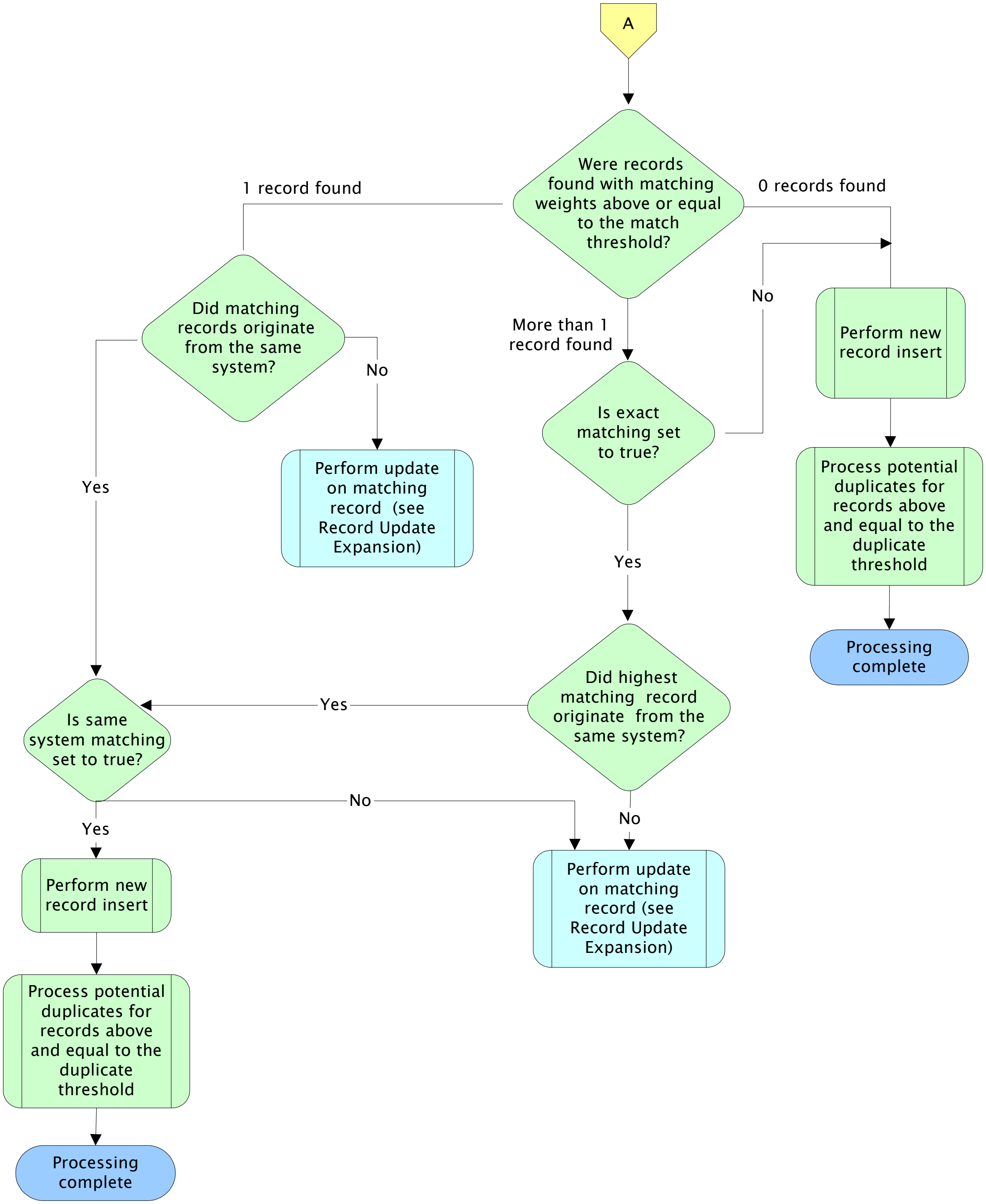
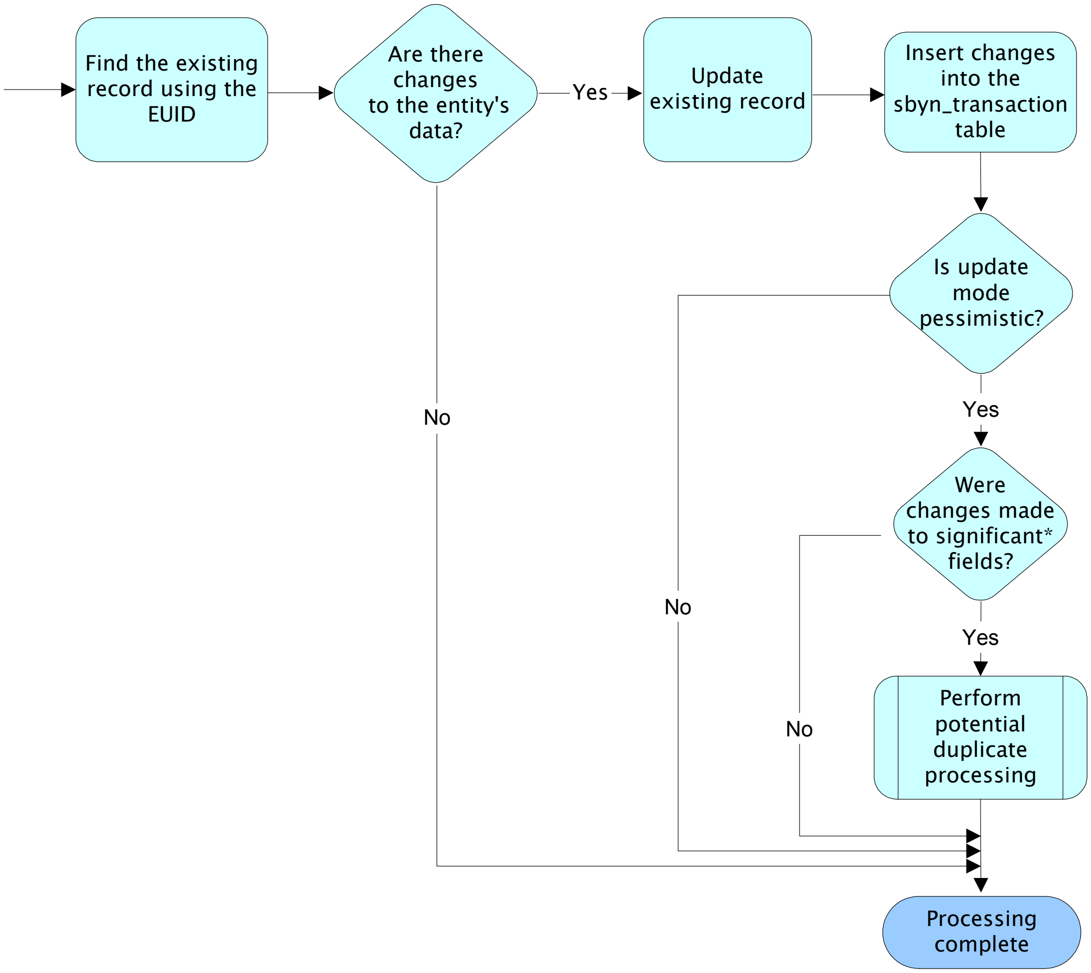

A
Were records found with matching weights above or equal to the match threshold?
1 record found
0 records found
More than 1 record found
Did matching records originate from the same system?
No
Yes
Perform update on matching record (see Record Update Expansion)
Is exact matching set to true?
No
Yes
Perform new record insert
Process potential duplicates for records above and equal to the duplicate threshold
Processing complete
Did highest matching record originate from the same system?
Yes
No
Is same system matching set to true?
No
Yes
Perform new record insert
Process potential duplicates for records above and equal to the duplicate threshold
Processing complete
Perform update on matching record (see Record Update Expansion)

Figure 6 Record Update Expansion


Find the existing record using the EUID
Are there changes to the entity's data?
Yes
Update existing record
Insert changes into the sbyn_transaction table
Is update mode pessimistic?
Yes
Were changes made to significant* fields?
Yes
Perform potential duplicate processing
No
No
No
Processing complete

* Significant fields for potential duplicate processing include those defined for matching and those included in the blocking query used for matching

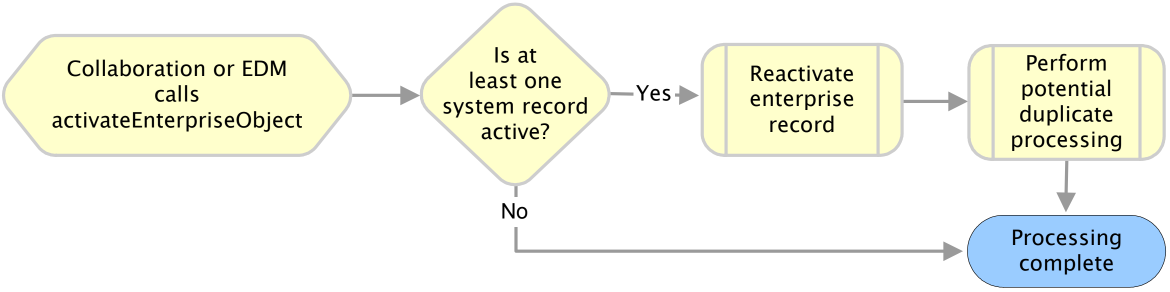
2.5 Primary Function Processing Logic

The primary functions of eIndex SPV can be performed from the Enterprise Data Manager or can be called from the Collaborations in the eIndex SPV Project. 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 located in the Master Controller class, and are fully described in the eIndex SPV Javadocs provided with eIndex SPV. In the following diagrams, significant fields for potential duplicate processing include fields defined for matching and fields included in the blocking query used for matching. In all of the methods described below, an entry is made in the transaction history table (sbyn_transaction).

2.5.1 activateEnterpriseObject

This method reactivates an enterprise record. The EDM calls this method when you select an EUID and then click **Activate EUID=<EUID_number>**, (where <EUID_number> is the EUID of the enterprise record to reactivate). Since all potential duplicates were deleted when the EUID was originally deactivated, potential duplicates are always recalculated, regardless of the update mode. Figure 7 illustrates the processing steps.

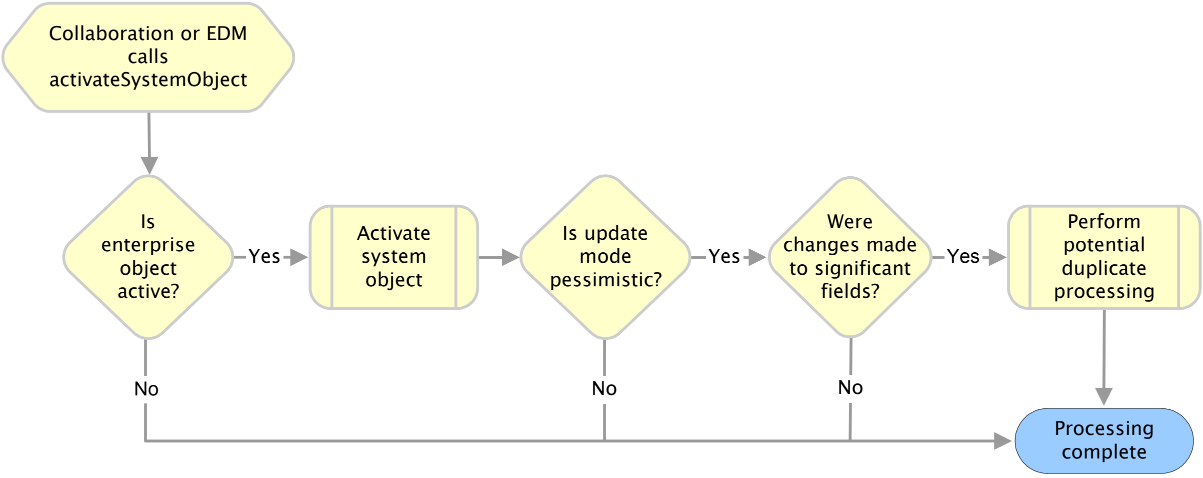
Figure 7 activateEnterpriseObject Processing


Collaboration or EDM calls activateEnterpriseObject
Is at least one system record active?
Yes
No
Reactivate enterprise record
Perform potential duplicate processing
Processing complete

2.5.2 activateSystemObject

This method reactivates a system record. The EDM calls this method when you select a system from the enterprise record tree and then click **Activate <system-ID>** (where *system* is the system code and *ID* is the local ID number for the system record to reactivate). 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 8 illustrates the processing steps.

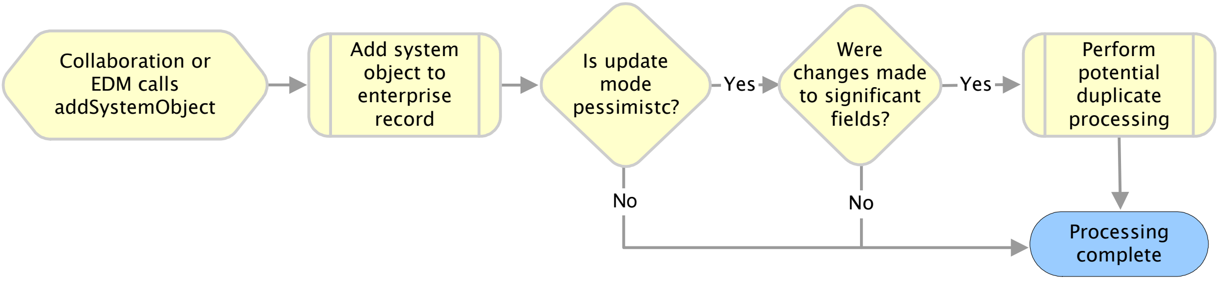
Figure 8 activateSystemObject Processing


Collaboration or EDM calls activateSystemObject
Is enterprise object active?
Yes
Activate system object
Is update mode pessimistic?
Yes
Were changes made to significant fields?
Yes
Perform potential duplicate processing
No
No
No
Processing complete

2.5.3 addSystemObject

This method adds a system record to an enterprise record. The EDM calls this method when you add a system record to an existing enterprise 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 and the update mode is set to pessimistic, potential duplicates are recalculated for the enterprise record. Figure 9 illustrates the processing steps.

Figure 9 addSystemObject Processing


Collaboration or EDM calls addSystemObject
Add system object to enterprise record
Is update mode pessimistc?
Yes
Were changes made to significant fields?
Yes
Perform potential duplicate processing
No
No
Processing complete

2.5.4 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 EDM and are designed for use in Collaborations.

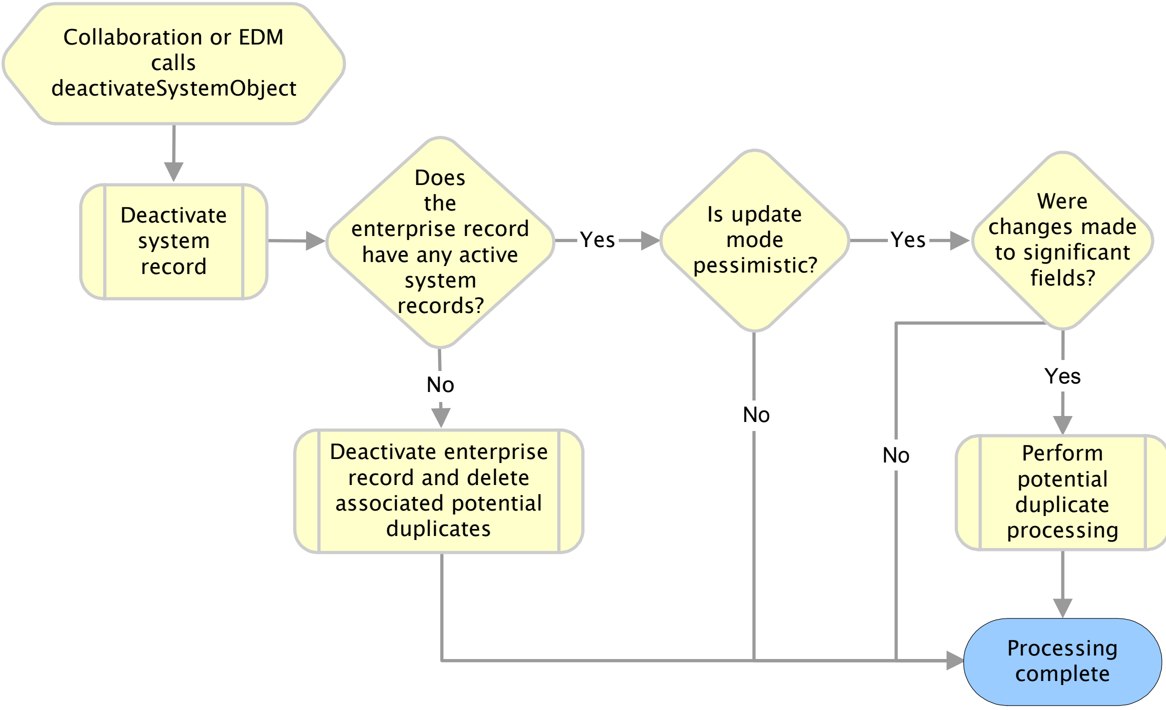
2.5.5 deactivateEnterpriseObject

This method deactivates an enterprise record specified by its EUID. The EDM calls this method when you select an enterprise record and then click **Deactivate EUID=<EUID_number>** (where <EUID_number> is the EUID of the enterprise record to deactivate). When an enterprise record is deactivated, all potential duplicate listings for that record are deleted.

2.5.6 deactivateSystemObject

This method deactivates a system record in an enterprise record. The EDM calls this method when you select a system from the enterprise record tree and then click **Deactivate <system-ID>** (where *system* is the system code and *ID* is the local ID number for the system record to deactivate). 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. (Note that if the system record is reactivated, then the enterprise record is recreated.) 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 10 illustrates the processing steps.

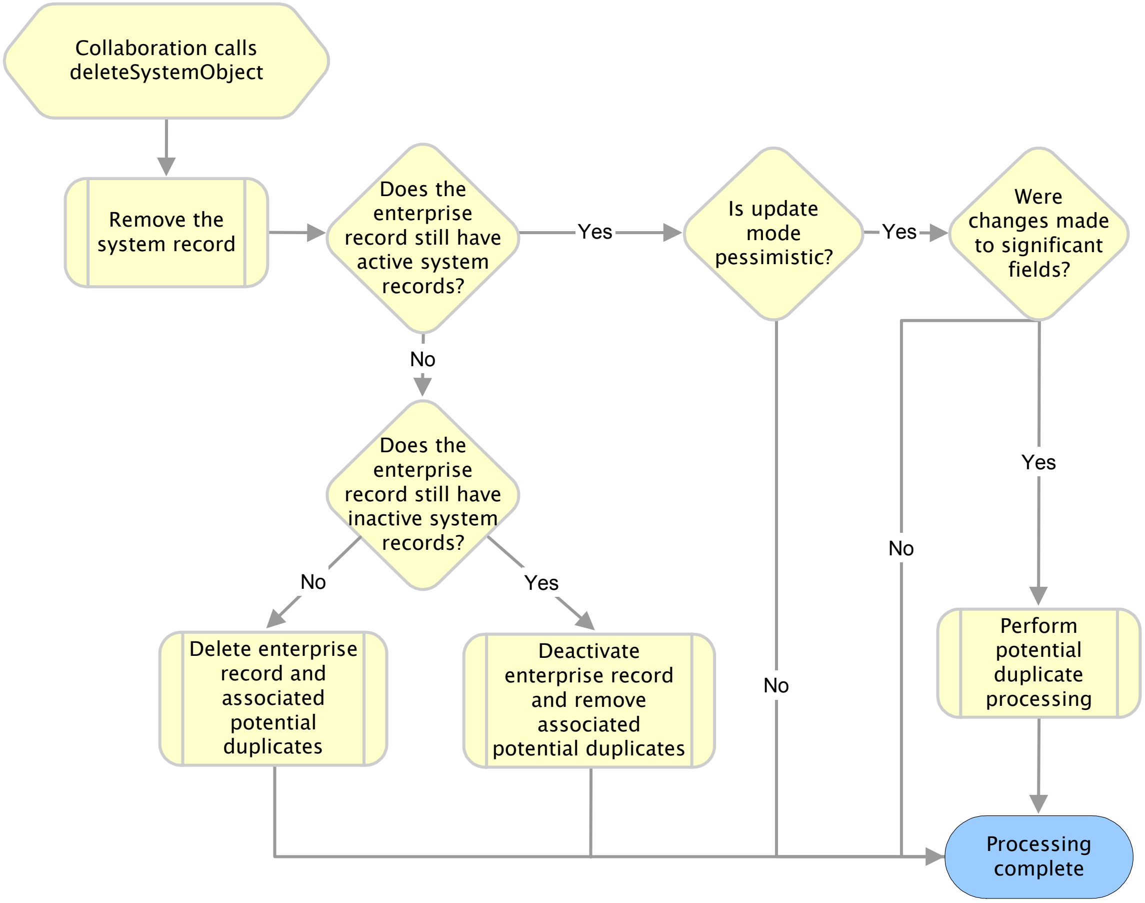
Figure 10 deactivateSystemObject Processing


Collaboration or EDM calls deactivateSystemObject
Deactivate system record
Does the enterprise record have any active system records?
Yes
No
Deactivate enterprise record and delete associated potential duplicates
Is update mode pessimistic?
Yes
No
Were changes made to significant fields?
Yes
No
Perform potential duplicate processing
Processing complete

2.5.7 deleteSystemObject

Unlike **deactivateSystemObject**, this method permanently removes a system record from an enterprise record. This method cannot be called from the EDM. If the enterprise record containing the deleted system record has no active system records remaining (but does have deactivated system records), the enterprise record is deactivated. If the enterprise record has no remaining system records after the system object is deleted, the enterprise record is also deleted. In both cases, any potential duplicate listings for that enterprise record are removed. If the enterprise record has active system records after the transaction and the update mode is set to “pessimistic”, the application checks whether any key fields were updated in the SBR. If key fields were updated, potential duplicates are recalculated for the enterprise record. Figure 11 illustrates the processing steps.

Figure 11 deleteSystemObject Processing

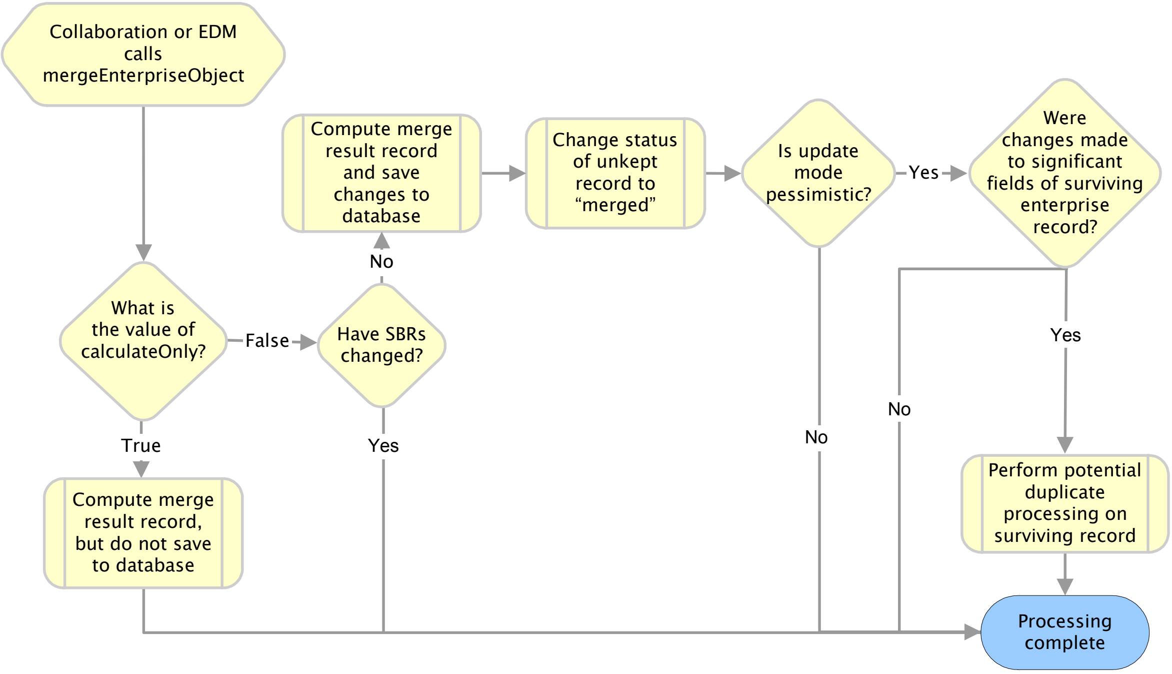

Collaboration calls deleteSystemObject
Remove the system record
Does the enterprise record still have active system records?
Yes
Is update mode pessimistic?
Yes
Were changes made to significant fields?
No
Does the enterprise record still have inactive system records?
No
Yes
Delete enterprise record and associated potential duplicates
Deactivate enterprise record and remove associated potential duplicates
No
No
Yes
Perform potential duplicate processing
Processing complete

2.5.8 mergeEnterpriseObject

There are four **mergeEnterpriseObject** methods that merge two enterprise records (see the Javadocs provided with eIndex SPV for more information about each). The EDM calls a merge method twice during a merge transaction. When you first click the **EUID Merge** arrow, 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 EDM 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 EDM 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 12 illustrates the processing steps, and includes the check for SBR differences, which only occurs in two of the merge methods.

Figure 12 mergeEnterpriseObject Processing


Collaboration or EDM calls mergeEnterpriseObject
What is the value of calculateOnly?
False
Have SBRs changed?
No
Compute merge result record and save changes to database
Change status of unkept record to "merged"
Is update mode pessimistic?
Yes
Were changes made to significant fields of surviving enterprise record?
Yes
No
No
True
Compute merge result record, but do not save to database
Yes
Perform potential duplicate processing on surviving record
Processing complete

2.5.9 mergeSystemObject

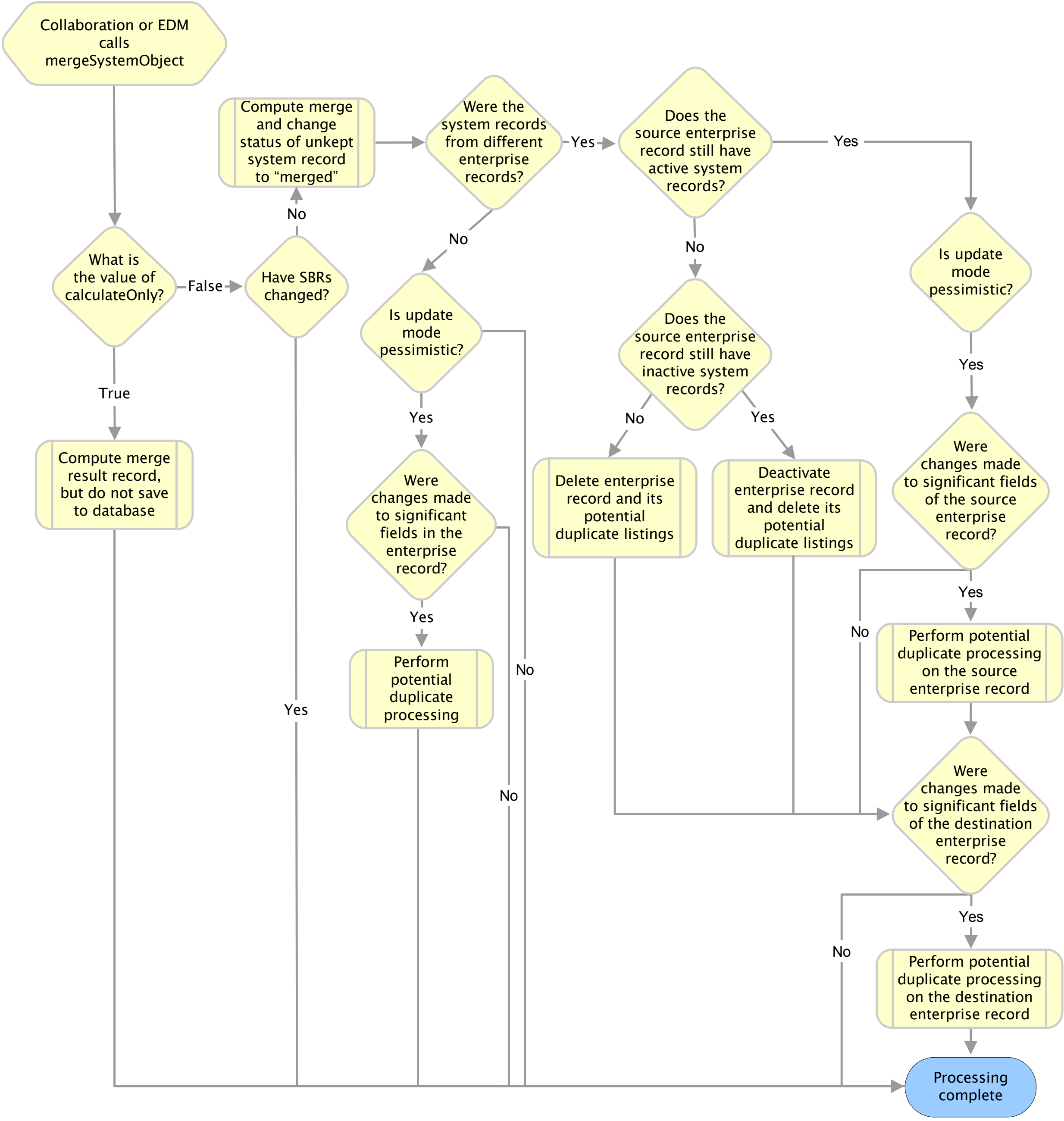
There are four methods that merge two system records, either from the same enterprise record or from two different enterprise records (for more information about each method, see the Javadocs provided with eIndex SPV). The system records must originate from the same external system. The EDM calls this method twice during a system record merge transaction. When you first click the **LID Merge** arrow, 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 EDM 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 the system record no longer has any active system records but does contain inactive system records, that enterprise record is deactivated and all associated potential duplicate listings are deleted. (Note that if the system records are unmerged, the enterprise record is reactivated.) If the enterprise record that contained the unkept system record no longer has any system records, that enterprise record is deleted along with any potential duplicate listings.

If both enterprise records are still active and the update mode is set to "pessimistic", the application checks whether any key fields were updated in the SBR for each enterprise record. If key fields were updated, potential duplicates are recalculated for each enterprise record. Figure 13 on page 37 illustrates the processing steps, and includes the check for SBR differences, which only occurs in two of the merge methods.

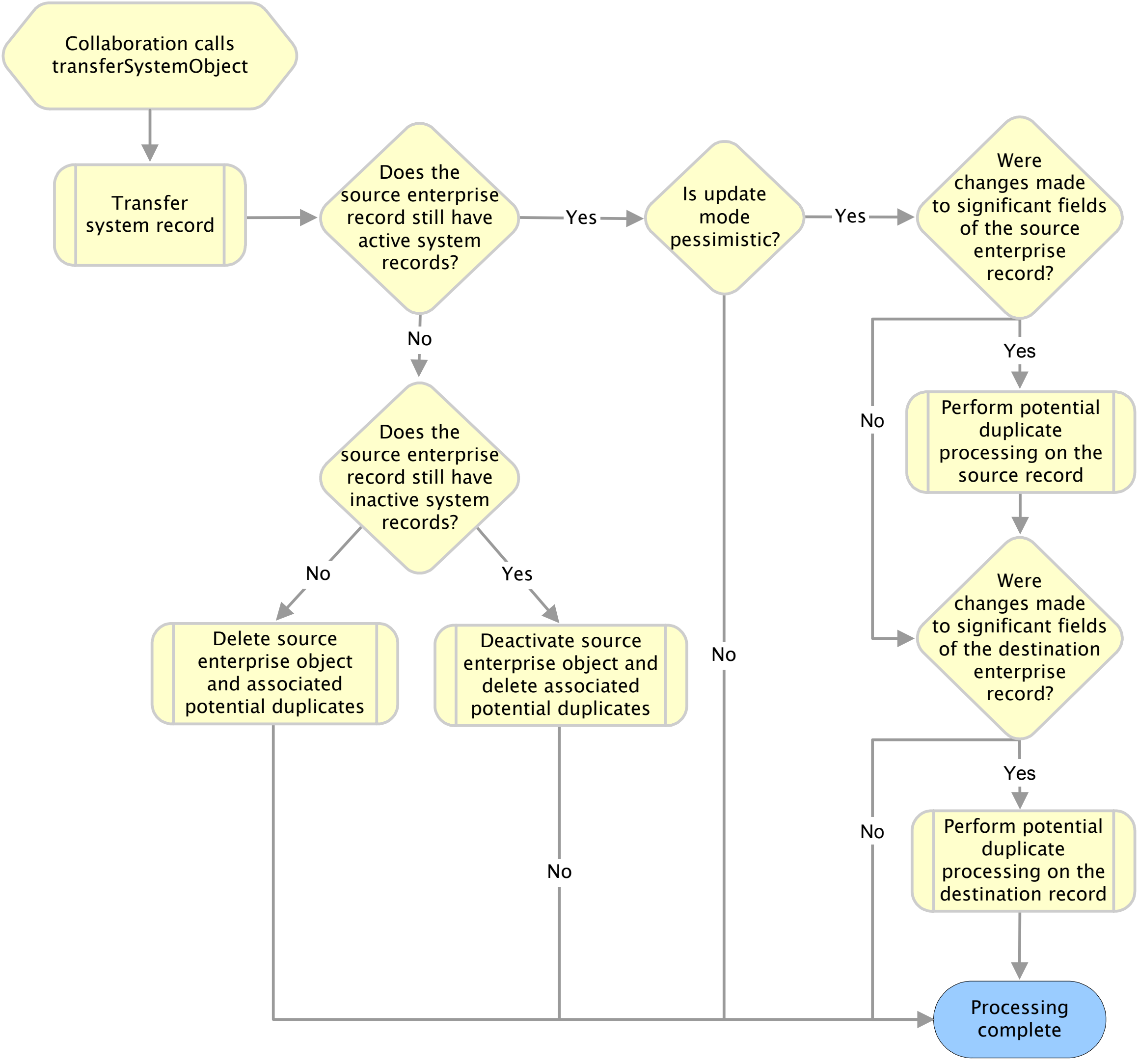
Figure 13 mergeSystemObject Processing


Collaboration or EDM calls mergeSystemObject
What is the value of calculateOnly?
False
True
Compute merge result record, but do not save to database
Have SBRs changed?
No
Yes
Compute merge and change status of unkept system record to "merged"
Were the system records from different enterprise records?
Yes
No
Is update mode pessimistic?
Yes
No
Were changes made to significant fields in the enterprise record?
Yes
No
Perform potential duplicate processing
Does the source enterprise record still have active system records?
Yes
No
Does the source enterprise record still have inactive system records?
No
Yes
Delete enterprise record and its potential duplicate listings
Deactivate enterprise record and delete its potential duplicate listings
Is update mode pessimistic?
Yes
Were changes made to significant fields of the source enterprise record?
Yes
No
Perform potential duplicate processing on the source enterprise record
Were changes made to significant fields of the destination enterprise record?
Yes
No
Perform potential duplicate processing on the destination enterprise record
Processing complete

2.5.10 transferSystemObject

This transfers a system record from one enterprise record to another. This method is not called from the EDM. 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 14 illustrates the processing steps.

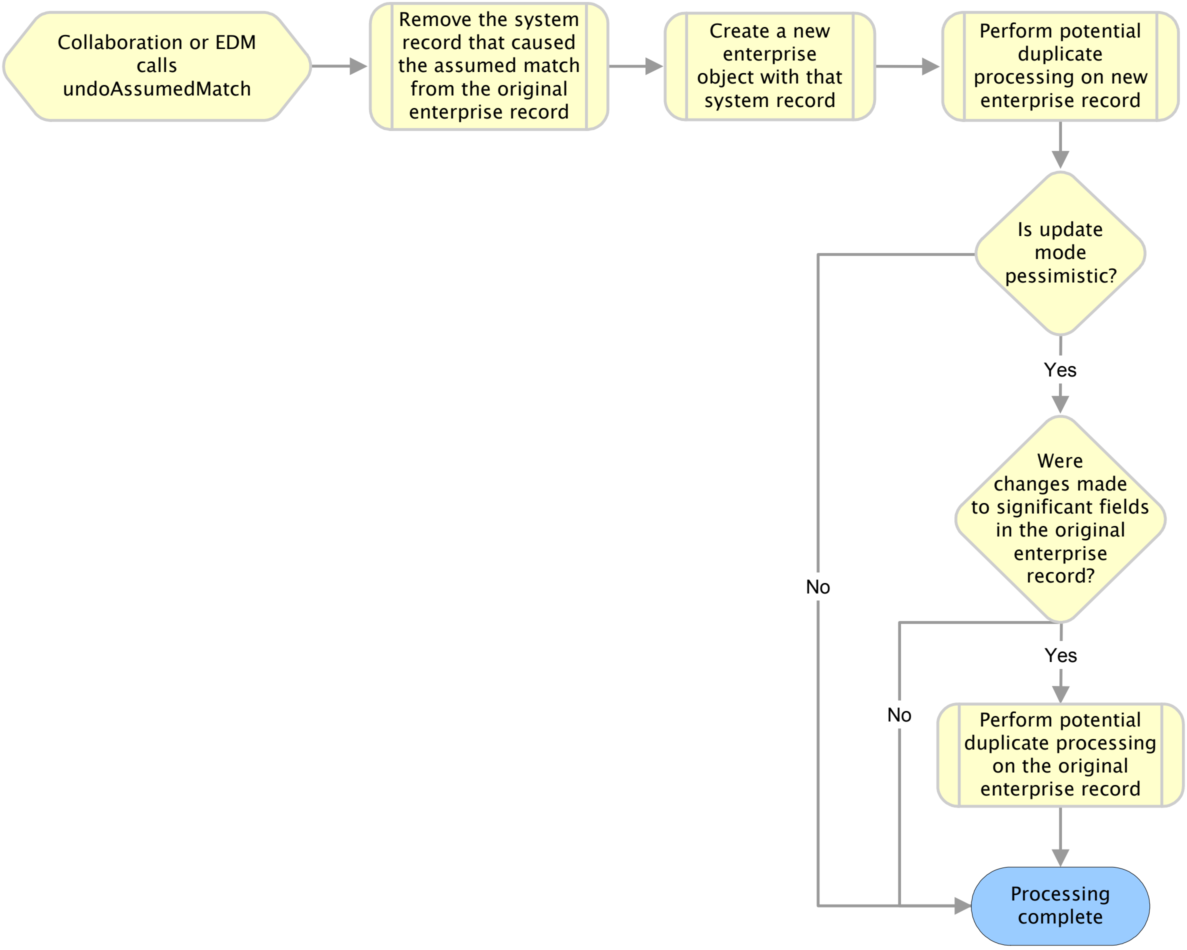
Figure 14 transferSystemObject Processing


Collaboration calls transferSystemObject
Transfer system record
Does the source enterprise record still have active system records?
Yes
Is update mode pessimistic?
Yes
Were changes made to significant fields of the source enterprise record?
No
Yes
Perform potential duplicate processing on the source record
No
Does the source enterprise record still have inactive system records?
No
Yes
Delete source enterprise object and associated potential duplicates
Deactivate source enterprise object and delete associated potential duplicates
No
Were changes made to significant fields of the destination enterprise record?
Yes
No
Perform potential duplicate processing on the destination record
No
Processing complete

2.5.11 undoAssumedMatch

This method reverses an assumed match made by the master index application, using the information from the system record that created the assumed match to create a new enterprise record. The EDM 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 15 illustrates the processing steps.

Figure 15 undoAssumedMatch Processing

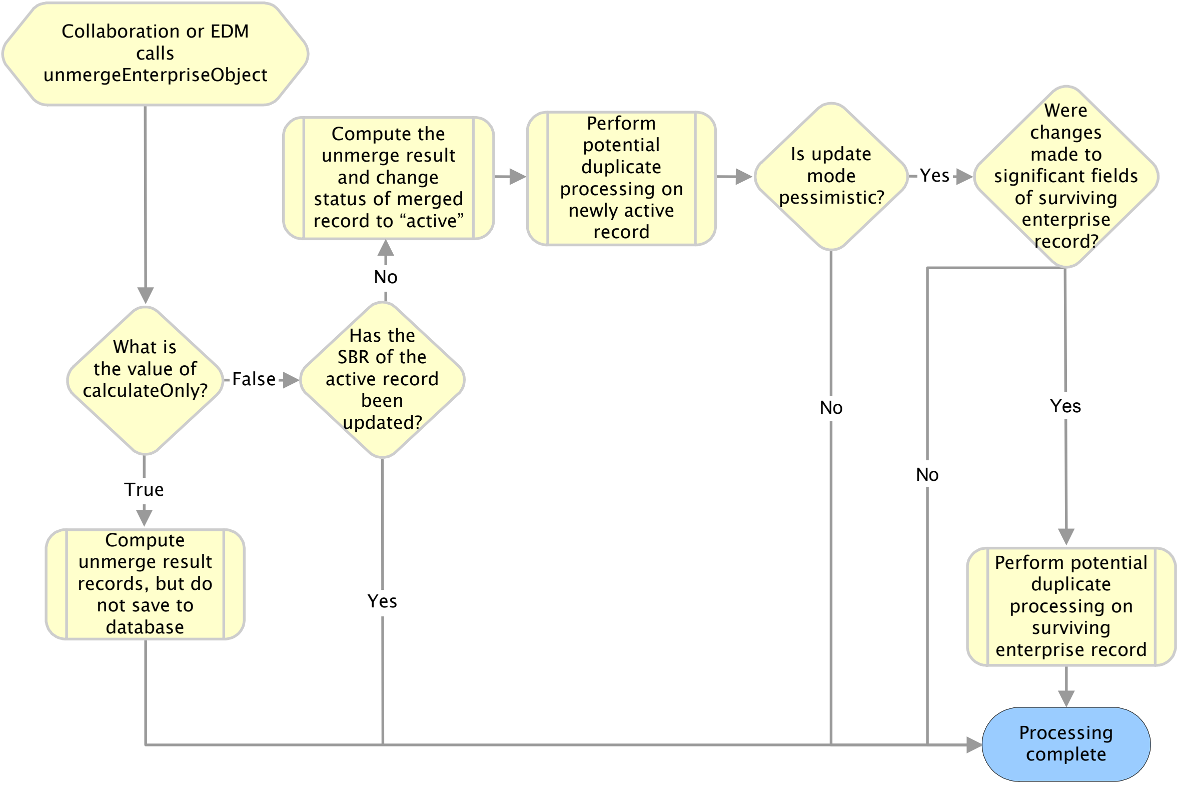

Collaboration or EDM calls undoAssumedMatch
Remove the system record that caused the assumed match from the original enterprise record
Create a new enterprise object with that system record
Perform potential duplicate processing on new enterprise record
Is update mode pessimistic?
Yes
No
Were changes made to significant fields in the original enterprise record?
Yes
No
Perform potential duplicate processing on the original enterprise record
Processing complete

2.5.12 unmergeEnterpriseObject

There are two methods that unmerge two enterprise records that were previously merged. One method unmerges the record without checking to make sure the SBR of the active record was not changed by another process before finalizing the merge and one method performs the SBR check (see the Javadocs provided with eIndex SPV for more information). The EDM 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 EDM 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 16 illustrates the processing steps and includes the check for SBR updates.

Figure 16 unmergeEnterpriseObject Processing


Collaboration or EDM calls unmergeEnterpriseObject
What is the value of calculateOnly?
False
Has the SBR of the active record been updated?
No
Compute the unmerge result and change status of merged record to "active"
Perform potential duplicate processing on newly active record
Is update mode pessimistic?
Yes
Were changes made to significant fields of surviving enterprise record?
No
Yes
No
True
Compute unmerge result records, but do not save to database
Yes
Perform potential duplicate processing on surviving enterprise record
Processing complete

2.5.13 unmergeSystemObject

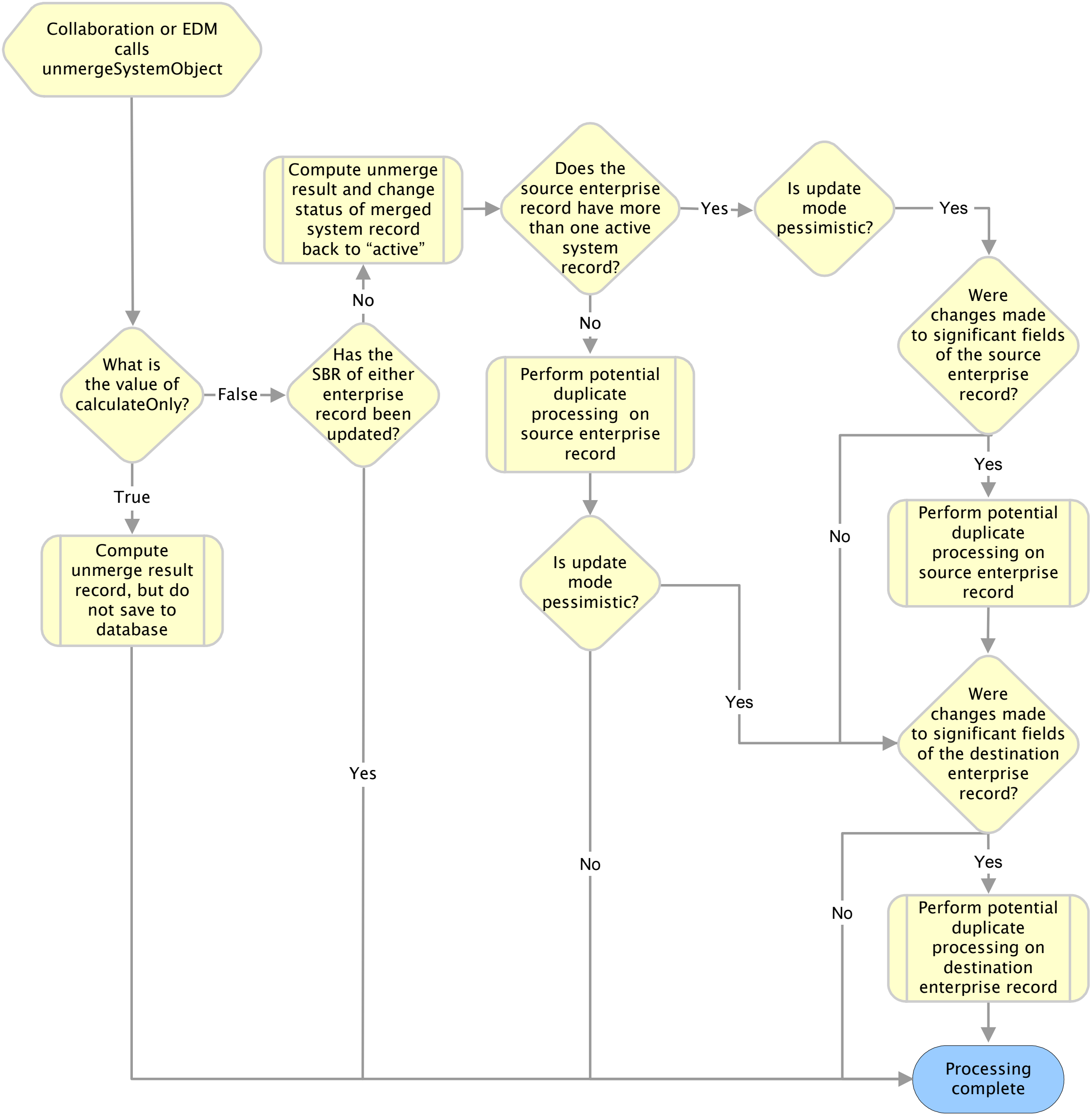
There are two methods that unmerge two system records that had previously been merged. One method unmerges the record without checking to make sure the SBR of the active record was not changed by another process before finalizing the merge and one method performs the SBR check (see the Javadocs provided with eIndex SPV for more information). The EDM 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 EDM 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 destination enterprise record. If key fields were updated, potential duplicates are recalculated for each enterprise record. Figure 17 illustrates the processing steps, assuming the system record unmerge involves two enterprise records and including the check for SBR updates.

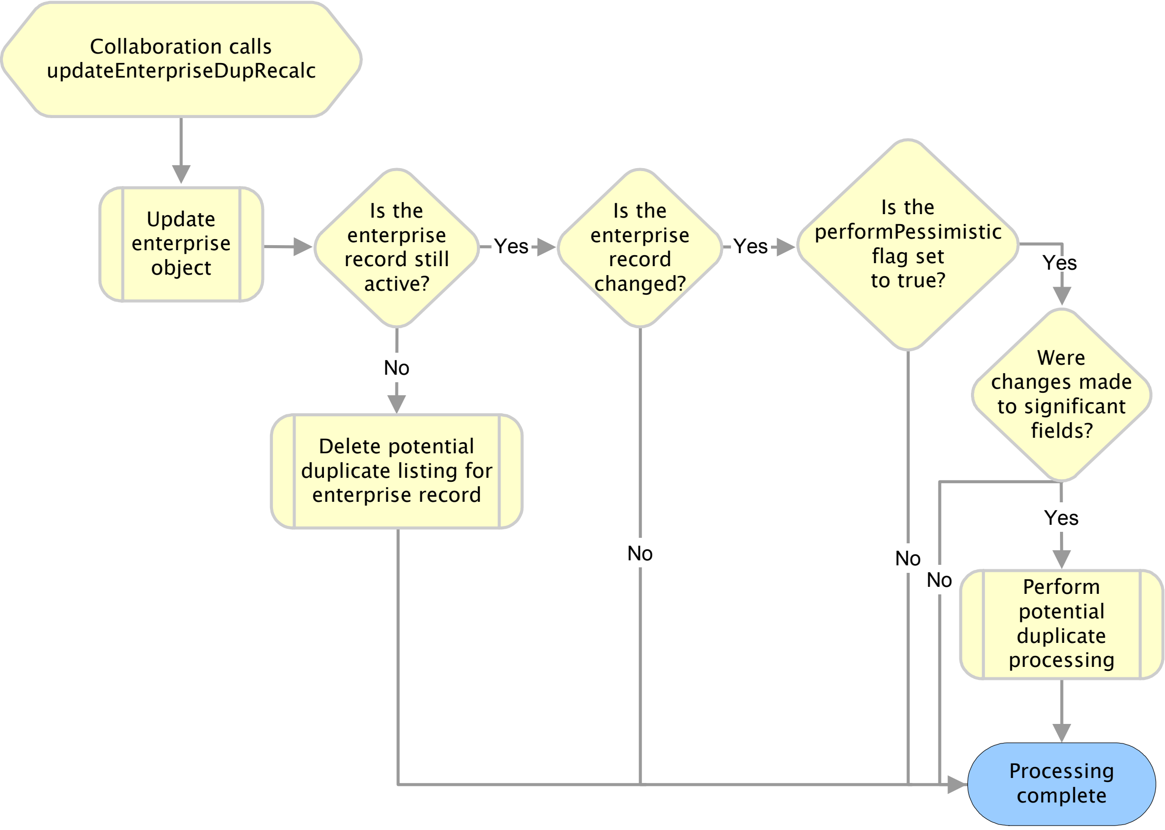
Figure 17 unmergeSystemObject Processing


Collaboration or EDM calls unmergeSystemObject
What is the value of calculateOnly?
False
True
Compute unmerge result record, but do not save to database
Has the SBR of either enterprise record been updated?
No
Yes
Compute unmerge result and change status of merged system record back to "active"
Does the source enterprise record have more than one active system record?
Yes
No
Is update mode pessimistic?
Yes
Perform potential duplicate processing on source enterprise record
Is update mode pessimistic?
Yes
No
Were changes made to significant fields of the source enterprise record?
Yes
No
Perform potential duplicate processing on source enterprise record
Were changes made to significant fields of the destination enterprise record?
Yes
No
Perform potential duplicate processing on destination enterprise record
Processing complete

2.5.14 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 EDM 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 18 illustrates the processing steps.

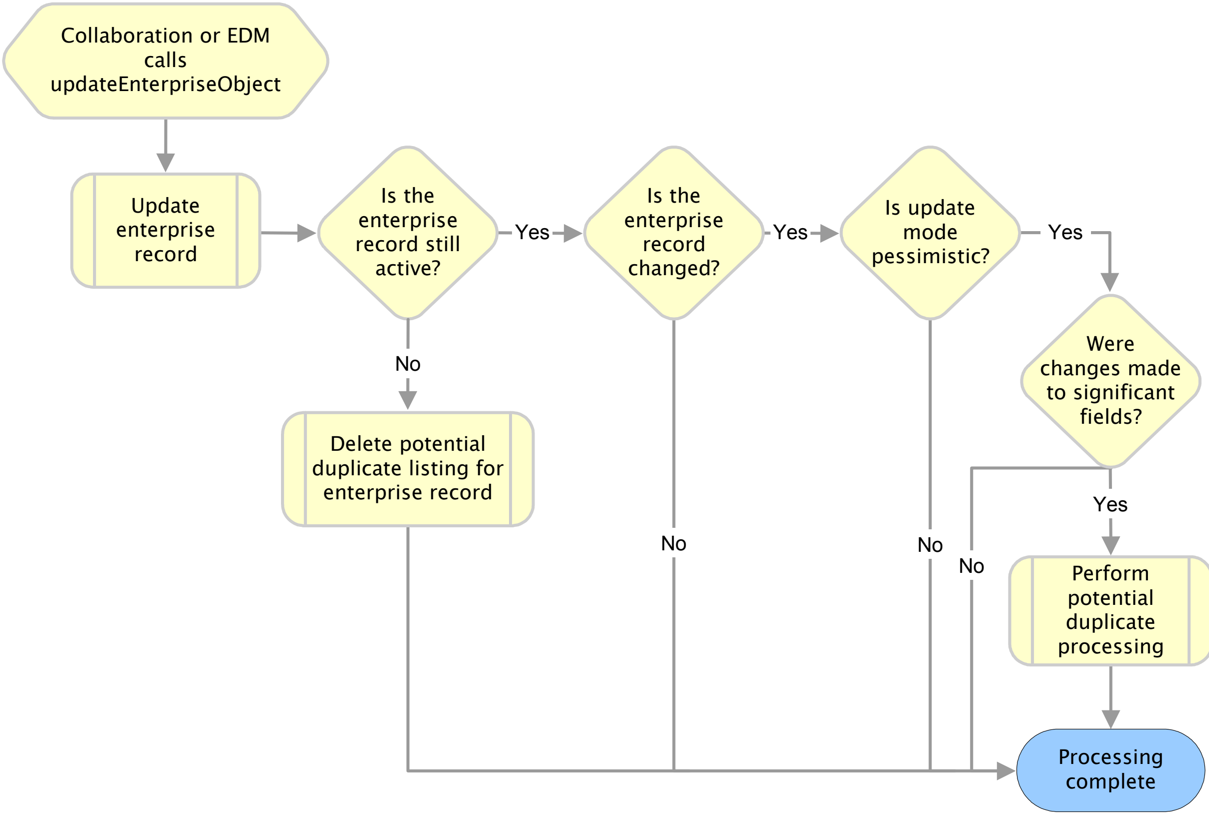
Figure 18 updateEnterpriseDupRecalc Processing


Collaboration calls updateEnterpriseDupRecalc
Update enterprise object
Is the enterprise record still active?
Yes
Is the enterprise record changed?
Yes
Is the performPessimistic flag set to true?
Yes
Were changes made to significant fields?
No
Delete potential duplicate listing for enterprise record
No
No
No
Yes
Perform potential duplicate processing
Processing complete

2.5.15 updateEnterpriseObject

This method updates the database to reflect new values for an enterprise record, and is called from the EDM when you commit changes to an existing record. If the enterprise record is deactivated during the update, potential duplicates are deleted for that record. If the enterprise record is still active, was changed during the transaction, and the update mode is set to “pessimistic”, the application checks whether any key fields were updated in the SBR of the enterprise record. If key fields were updated, potential duplicates are recalculated. Figure 19 illustrates the processing steps.

Figure 19 updateEnterpriseObject Processing

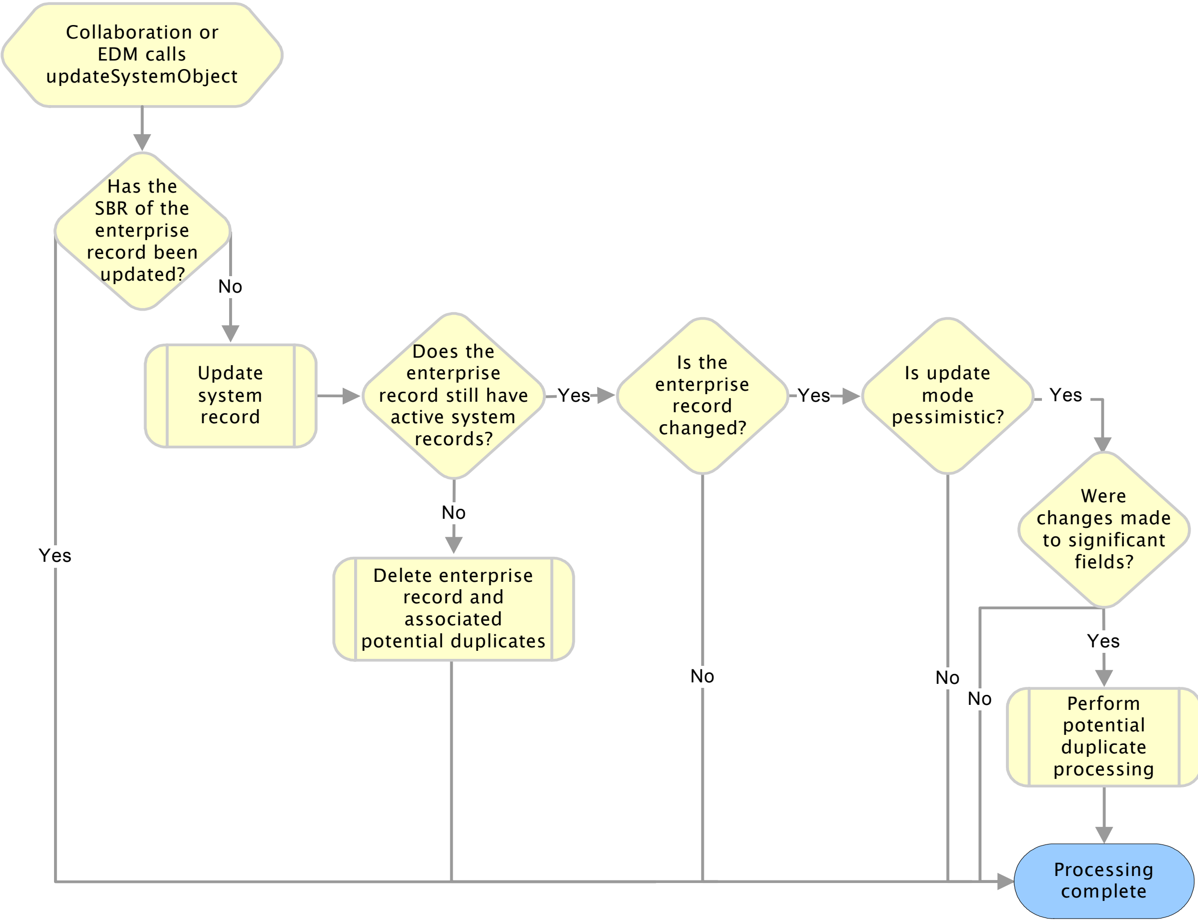

Collaboration or EDM calls updateEnterpriseObject
Update enterprise record
Is the enterprise record still active?
Yes
No
Delete potential duplicate listing for enterprise record
Is the enterprise record changed?
Yes
No
Is update mode pessimistic?
Yes
No
Were changes made to significant fields?
Yes
No
Perform potential duplicate processing
Processing complete

2.5.16 updateSystemObject

There are two methods that update the database to reflect new values for a system record. One method updates the record without checking that there were no concurrent changes to the record, and the other method compares the SBR of the associated enterprise object in the transaction with that in the database to be sure there were no concurrent changes (see the Javadocs provided with eIndex SPV for more information). The EDM 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 20 illustrates the processing steps and includes the check for SBR changes though it only occurs with one of the methods.

Figure 20 updateSystemObject Processing



Collaboration or EDM calls updateSystemObject
Has the SBR of the enterprise record been updated?
No
Yes
Update system record
Does the enterprise record still have active system records?
Yes
No
Delete enterprise record and associated potential duplicates
Is the enterprise record changed?
Yes
No
Is update mode pessimistic?
Yes
No
Were changes made to significant fields?
Yes
No
Perform potential duplicate processing
Processing complete

Chapter 3

The Database Structure

This chapter provides information about the master index 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 customizations made to the Object Definition and to the scripts used to create the master index database.

What's in This Chapter

3.1 About the Database

3.1.1 Overview

The master index database stores information about the patients being indexed. The database stores records from local systems in their original form and also stores a record for each patient that is considered to be the single best record (SBR).

The structure of the database tables that store patient information is dependent on the information specified in the Object Definition file. eIndex SPV includes a script to create the tables and fields in the database based on the information in the Object Definition file If you update the Object Definition file, generating the application updates the database scripts accordingly. This allows you to define the database as you define the object structure.

3.1.2 Database Table Overview

While most of the structures created in the database are based on information in the Object Definition file, 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 patients defined for the eIndex SPV application as well as tables that store common maintenance information, transactional information, and external system information. The database includes the tables listed in Table 5.

Table 5 Master Index Database Tables

Table Name	Description
SBYN_<OBJECT_NAME>	Stores information for the parent objects associated with local system records (by default, *Person* objects). This database table is named by the parent object name. Only one table stores parent object information for system records.
SBYN_<OBJECT_NAME>SBR	Stores information for the parent objects associated with single best records (by default, *Person* objects). This database table is named by the parent object name followed by "SBR". Only one table stores parent object information for SBRs.
SBYN_<CHILD_OBJECT>	Stores information for child objects associated with local system records. These database tables are named by their object name. For example, a table storing address objects is named sbyn_address; a table storing comment objects is named sbyn_comment. One database table is created for each child object defined in the object structure.
SBYN_<CHILD_OBJECT>SBR	Stores information for child objects associated with a single best record. These database tables are named by their object name followed by "SBR". For example, a table storing address objects is named sbyn_addresssbr; a table storing comment objects is named sbyn_commentsbr. One SBR database table is created for each child object defined in the object structure.
SBYN_APPL	Lists the applications with which each item in stc_common_header is associated. Currently the only item in this table is **eView**.
SBYN_ASSUMEDMATCH	Stores information about records that were automatically matched by the master index.
SBYN_AUDIT	Stores audit information about each time patient information is accessed from the EDM. ***Note:*** *If audit logging is enabled, this table can grow very large and might require periodic archiving.*
SBYN_COMMON_DETAIL	Contains all of the processing codes associated with the items listed in sbyn_common_header.

Table 5 Master Index Database Tables

Table Name	Description
SBYN_COMMON_HEADER	Contains a list of the different types of processing codes used by the master index. These types are also associated with the drop-down lists you can specify for the EDM.
SBYN_ENTERPRISE	Stores the local ID and system pairs, along with their associated EUID.
SBYN_MERGE	Stores information about all merge and unmerge transactions processed from either external systems or the EDM.
SBYN_OVERWRITE	Stores information about fields that are locked for updates in an SBR.
SBYN_POTENTIALDUPLICATES	Stores a list of potential duplicate records and flags potential duplicate pairs that have been resolved.
SBYN_SEQ_TABLE	Stores the sequential codes that are used in other tables in the database, such as EUIDs, transaction numbers, and so on.
SBYN_SYSTEMOBJECT	Stores information about the system objects in the database, including the local ID and system, create date and user, status, and so on.
SBYN_SYSTEMS	Stores a list of systems in your organization, along with defining information.
SBYN_SYSTEMSBR	Stores transaction information about an SBR, such as the create or update date, status, and so on.
SBYN_TRANSACTION	Stores a history of changes to each record stored in the database.
SBYN_USER_CODE	Like the sbyn_common_detail table, this table stores processing codes and drop-down list values. This table contains additional validation information that allows you to validate information in a dependent field (for example, to validate cities against the entered postal code).

3.2 Database Table Details

The tables on the following pages describe each column in the default database tables.

3.2.1 SBYN_<OBJECT_NAME>

This table stores the parent object in each system record received by the master index. By default, the table is named SBYN_PERSON. It is linked to the tables that store each child object in the system record by the **<object_name>id** column (where <object_name> is the name of the parent object). This table contains the columns listed below regardless of the design of the object structure, and also contains a column for each field you defined for the parent object in the Object Definition file.

3.2.2 SBYN_<OBJECT_NAME>SBR

This table stores the parent object of the SBR for each enterprise object in the master index database. By default, the table is named SBYN_PERSONSBR. 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 the Object Definition file.

3.2.3 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 master index. 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.

Table 6 SBYN_<CHILD_OBJECT> and SBYN_<CHILD_OBJECT>SBR Table Description

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

3.2.4 SBYN_<CHILD_OBJECT>SBR

The sbyn_<child_object> sbr 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 master index. 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 the object structure definition file, including any standardized or phonetic fields.

Table 7 SBYN_<CHILD_OBJECT> and SBYN_<CHILD_OBJECT>SBR Table Description

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

3.2.5 SBYN_APPL

This table stores information about the applications used in the eIndex SPV system. Currently, there is only one entry, “eView”.

Table 8 SBYN_APPL Table Description

Column Name	Data Type	Description
APPL_ID	NUMBER(10)	The unique sequence number code for the listed application.
CODE	VARCHAR2(8)	A unique code for the application.
DESCR	VARCHAR2(30)	A brief description of the application.
READ_ONLY	CHAR(1)	An indicator of whether the current entry can be modified. If the value of this column is “Y”, the entry cannot be modified.
CREATE_DATE	DATE	The date the application entry was created.
CREATE_USERID	VARCHAR2(20)	The logon ID of the user who created the application entry.

3.2.6 SBYN_ASSUMEDMATCH

This table maintains a record of each assumed match transaction that occurs in the master index, 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.

Table 9 SBYN_ASSUMEDMATCH Table Description

Column Name	Data Type	Description
ASSUMEDMATCHID	VARCHAR2(20)	The unique ID for the assumed match transaction.
EUID	VARCHAR2(20)	The EUID into which the incoming record was merged.

Table 9 SBYN_ASSUMEDMATCH Table Description

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

3.2.7 SBYN_AUDIT

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

Table 10 SBYN_AUDIT Table Description

Column Name	Data Type	Description
AUDIT_ID	VARCHAR2(20)	The unique identification code for the audit record. This column cannot be null.
PRIMARY_OBJECT_TYPE	VARCHAR2(20)	The name of the parent object as defined in the Object Definition file.
EUID	VARCHAR2(15)	The EUID whose information was accessed during an EDM transaction.
EUID_AUX	VARCHAR2(15)	The second EUID whose information was accessed during an EDM transaction. A second EUID appears when viewing information about merge and unmerge transactions, comparisons, and so on.
FUNCTION	VARCHAR2(32)	The type of transaction that caused the audit record to be written. This column cannot be null.
DETAIL	VARCHAR2(120)	A brief description of the transaction that caused the audit record to be written.
CREATE_DATE	DATE	The date the transaction that created the audit record was performed. This column cannot be null.

Table 10 SBYN_AUDIT Table Description

Column Name	Data Type	Description
CREATE_BY	VARCHAR2(20)	The user ID of the person who performed the transaction that caused the audit log. This column cannot be null.

3.2.8 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.

Table 11 SBYN_COMMON_DETAIL Table Description

Column Name	Data Type	Description
COMMON_DETAIL_ID	NUMBER(10)	The unique identification code of the common table data element.
COMMON_HEADER_ID	NUMBER(10)	The unique identification code of the common table data type associated with the data element (as stored in the common_header_id column of the sbyn_common_header table).
CODE	VARCHAR2(20)	The processing code for the common table data element.
DESCR	VARCHAR2(50)	A description of the common table data element.
READ_ONLY	CHAR(1)	An indicator of whether the common table data element can be modified.
CREATE_DATE	DATE	The date the data element record was created.
CREATE_USERID	VARCHAR2(20)	The user ID of the person who created the data element record.

3.2.9 SBYN_COMMON_HEADER

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

Table 12 SBYN_COMMON_HEADER Table Description

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

3.2.10 SBYN_ENTERPRISE

This table stores a list of all the system and local ID pairs assigned to the person records in the database, along with the associated EUID for each pair. This table is linked to sbyn_systemobject by the **systemcode** 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.

Table 13 SBYN_ENTERPRISE Table Description

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

3.2.11 SBYN_MERGE

This table maintains a record of each merge transaction that occurs in the master index, both through the EDM and the eGate Project. It also records any unmerges that occur.

Table 14 SBYN_MERGE Table Description

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

3.2.12 SBYN_OVERWRITE

This table stores information about the fields that are locked 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.

Table 15 SBYN_OVERWRITE Table Description

Column Name	Data Type	Description
EUID	VARCHAR2(20)	The EUID of an SBR containing fields for which the overwrite lock is set.
PATH	VARCHAR2(200)	The ePath to a field that is locked in an SBR from the EDM.
TYPE	VARCHAR2(20)	The data type of a field that is locked in an SBR.
INTEGERDATA	NUMBER(38)	The data that is locked for overwrite in an integer field.
BOOLEANDATA	NUMBER(38)	The data that is locked for overwrite in a boolean field.
STRINGDATA	VARCHAR2(200)	The data that is locked for overwrite in a string field.
BYTEDATA	CHAR(2)	The data that is locked for overwrite in a byte field.
LONGDATA	LONG	The data that is locked for overwrite in a long integer field.

Table 15 SBYN_OVERWRITE Table Description

Column Name	Data Type	Description
DATEDATA	DATE	The data that is locked for overwrite in a date field.
FLOATDATA	NUMBER(38,4)	The data that is locked for overwrite in a floating decimal field.
TIMESTAMPDATA	DATE	The data that is locked for overwrite in a timestamp field.

3.2.13 SBYN_POTENTIALDUPLICATES

This table maintains a list of all records that are potential duplicates of one another. It also maintains a record of whether a potential duplicate pair has been resolved or permanently resolved.

Table 16 SBYN_POTENTIALDUPLICATES Table Description

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

Table 16 SBYN_POTENTIALDUPLICATES Table Description

Column Name	Data Type	Description
EUID1	VARCHAR2(20)	The EUID of the first record in the potential duplicate pair.

3.2.14 SBYN_SEQ_TABLE

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

Table 17 SBYN_SEQ_TABLE Table Description

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

The default sequence numbers are listed in Table 18.

Table 18 Default Sequence Numbers

Sequence Name	Description
EUID	The sequence number that determines how EUIDs are assigned to new records. The chunk size for the EUID sequence number is configurable in the eIndex SPV Project Threshold file.
POTENTIALDUPLICATE	The sequence number assigned each potential duplicate transaction record in sbyn_potentialduplicates (column name “potentialduplicateid”).
TRANSACTIONNUMBER	The sequence number assigned to each transaction in the master index. This number is stored in sbyn_transaction (column name “transactionnumber”).
ASSUMEDMATCH	The sequence number assigned to each assumed match transaction record in sbyn_assumedmatch (column name “assumedmatchid”).
AUDIT	The sequence number assigned to each audit log record in sbyn_audit (column name “audit_id”).
MERGE	The sequence number assigned to each merge transaction in sbyn_merge (column name “merge_id”).

Table 18 Default Sequence Numbers

Sequence Name	Description
SBYN_APPL	The sequence number assigned to each application listed in sbyn_appl (column name "appl_id")
SBYN_COMMON_HEADER	The sequence number assigned to each common table data type listed in sbyn_common_header (column name "common_header_id").
SBYN_COMMON_DETAIL	The sequence number assigned to each common table data element listed in sbyn_common_detail (column name "common_detail_id").
<OBJECT_NAME>	Each parent and child object system record table is assigned a sequential ID. The column names are named after the object (for example, sbyn_address has a sequential column named "addressid"). The parent object ID is included in each child object table.
<OBJECT_NAME>SBR	Each parent and child object SBR table is assigned a sequential ID. The column names are named after the object (for example, sbyn_addresssbr has a sequential column named "addressid"). The parent object ID is included in each child object SBR table.

3.2.15 SBYN_SYSTEMOBJECT

This table stores information about the system records in the database, including their local ID and source system pairs. It also stores transactional information, such as the create or update date and function.

Table 19 SBYN_SYSTEMOBJECT Table Description

Column Name	Data Type	Description
SYSTEMCODE	VARCHAR2(20)	The processing code of the system associated with the local ID. This column cannot be null.
LID	VARCHAR2(25)	The local ID associated with the system and EUID (the associated EUID is found in sbyn_enterprise). This column cannot be null.
CHILDTYPE	VARCHAR2(20)	The type of object being processed (currently only the name of the parent object). This column is reserved for future use.
CREATEUSER	VARCHAR2(30)	The user ID of the person who created the system record.
CREATEFUNCTION	VARCHAR2(20)	The type of transaction that created the system record.
CREATEDATE	DATE	The date the system record was created.

Table 19 SBYN_SYSTEMOBJECT Table Description

Column Name	Data Type	Description
UPDATEUSER	VARCHAR2(30)	The user ID of the person who last updated the system record.
UPDATEFUNCTION	VARCHAR2(20)	The type of transaction that last updated the system record.
UPDATEDATE	DATE	The date the system record was last updated.
STATUS	VARCHAR2(15)	The status of the system record. The status can be one of these values: ▪ active ▪ inactive ▪ merged

3.2.16 SBYN_SYSTEMS

This table stores information about each system integrated into the eIndex SPV 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.

Table 20 SBYN_SYSTEMS Table Description

Column Name	Data Type	Description
SYSTEMCODE	VARCHAR2(20)	The unique processing code of the system.
DESCRIPTION	VARCHAR2(120)	A brief description of the system, or the system name. This is the value that appears in the tree view panes of the EDM for each system and local ID pair.
STATUS	CHAR(1)	The status of the system in the master index. "A" indicates active and "D" indicates deactivated.
ID_LENGTH	NUMBER	The length of the local identifiers assigned by the system. This length does not include any additional characters added by the input mask.

Table 20 SBYN_SYSTEMS Table Description

Column Name	Data Type	Description
FORMAT	VARCHAR2(60)	The required data pattern for the local IDs assigned by the system. For more information about possible values and using Java patterns, see "Patterns" in the class list for **java.util.regex** in the Javadocs provided with the Java™ 2 Platform, Standard Edition (J2SE™ platform). Note that the data pattern is also limited by the input mask described below. All regex patterns are supported if there is no input mask.
INPUT_MASK	VARCHAR2(60)	A mask used by the EDM to add punctuation to the local ID. For example, the input mask **DD-DDD-DDD** inserts a hyphen after the second and fifth characters in an 8-digit ID. These character types can be used. ▪ **D**—Numeric character ▪ **L**—Alphabetic character ▪ **A**—Alphanumeric character
VALUE_MASK	VARCHAR2(60)	A mask used to strip any extra characters that were added by the input mask for database storage. The value mask is the same as the input mask, but with an "x" in place of each punctuation mark. Using the input mask described above, the value mask is **DDxDDDxDDD**. This strips the hyphens before storing the ID.
CREATE_DATE	DATE	The date the system information was inserted into the database.
CREATE_USERID	VARCHAR2(20)	The logon ID of the user who inserted the system information into the database.
UPDATE_DATE	DATE	The most recent date the system's information was updated.
UPDATE_USERID	VARCHAR2(20)	The logon ID of the user who last updated the system's information.

3.2.17 SBYN_SYSTEMSBR

This table stores transactional information about the system records for the SBR, such as the create or update date and function. The sbyn_systemsbr table is indirectly linked to the sbyn_systemobjects table through sbyn_enterprise.

Table 21 SBYN_SYSTEMSBR Table Description

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

3.2.18 SBYN_TRANSACTION

This table stores a history of changes made to each record in the master index, allowing you to view a transaction history and to undo certain actions, such as merging two patient profiles.

Table 22 SBYN_TRANSACTION Table Description

Column Name	Data Type	Description
TRANSACTIONNUMBER	VARCHAR2(20)	The unique number of the transaction.
LID1	VARCHAR2(25)	This column is reserved for future use.
LID2	VARCHAR2(25)	The local ID of the second system record involved in the transaction.
EUID1	VARCHAR2(20)	This column is reserved for future use.
EUID2	VARCHAR2(20)	The EUID of the second patient profile involved in the transaction.
FUNCTION	VARCHAR2(20)	The type of transaction that occurred, such as update, add, merge, and so on.
SYSTEMUSER	VARCHAR2(30)	The logon ID of the user who performed the transaction.
TIMESTAMP	TIMESTAMP	The date and time the transaction occurred.
DELTA	BLOB	A list of the changes that occurred to system records as a result of the transaction.
SYSTEMCODE	VARCHAR2(20)	The processing code of the source system in which the transaction originated.
LID	VARCHAR2(25)	The local ID of the system record involved in the transaction.
EUID	VARCHAR2(20)	The EUID of the enterprise record involved in the transaction.

3.2.19 SBYN_USER_CODE

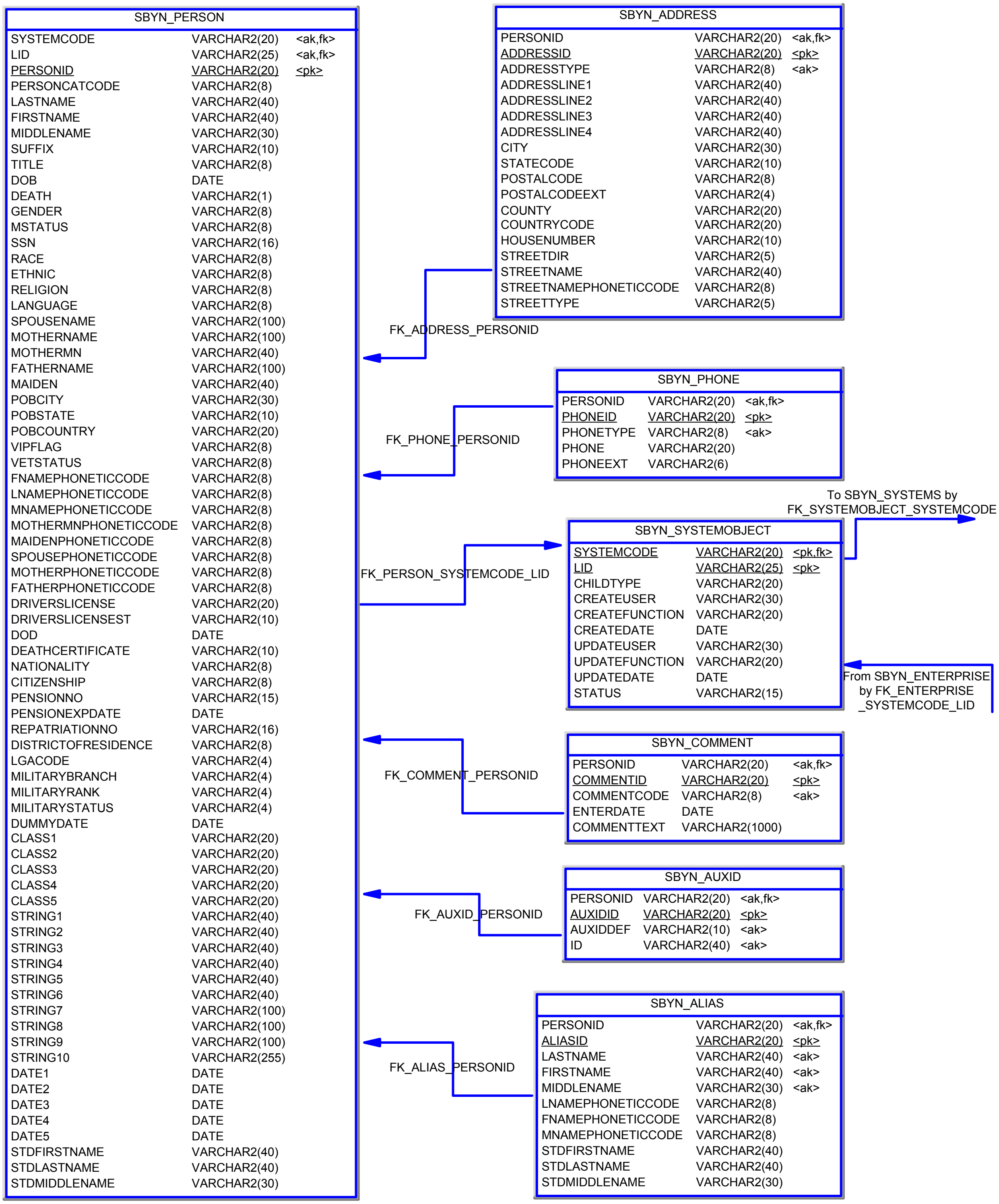
This table is similar to the sbyn_common_header and sbyn_common_detail tables in that it stores processing codes and drop-down list values. This table is used when the value of one field is dependent on the value of another. For example, if you store credit card information, you could list each credit 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.

Table 23 SBYN_USER_CODE Table Description

Column Name	Data Type	Description
CODE_LIST	VARCHAR2(20)	The code list name of the user code type (using the credit card example above, this might be similar to "CREDCARD"). This column links the values for each list.
CODE	VARCHAR2(20)	The processing code of each user code element.
DESCRIPTION	VARCHAR2(50)	A brief description or name for the user code. This is the value that appears in the drop-down list.
FORMAT	VARCHAR2(60)	The required data pattern for the field that is constrained by the user code. For more information about possible values and using Java patterns, see "Patterns" in the class list for **java.util.regex** in the Javadocs provided with the J2SE platform. Note that the data pattern is also limited by the input mask described below. All regex patterns are supported if there is no input mask.
INPUT_MASK	VARCHAR2(60)	A mask used by the EDM to add punctuation to the constrained field. For example, the input mask **DD-DDD-DDD** inserts a hyphen after the second and fifth characters in an 8-digit ID. These character types can be used. ▪ **D**—Numeric character ▪ **L**—Alphabetic character ▪ **A**—Alphanumeric character
VALUE_MASK	VARCHAR2(60)	A mask used to strip any extra characters that were added by the input mask for database storage. The value mask is the same as the input mask, but with an "x" in place of each punctuation mark. Using the input mask described above, the value mask is **DDxDDDxDDD**. This strips the hyphens before storing the ID.

3.3 Sample Database Model

The diagrams on the following pages illustrate the table structure and relationships for a sample eIndex SPV master index 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.



SBYN_PERSON		
SYSTEMCODE	VARCHAR2(20)	<ak,fk>
LID	VARCHAR2(25)	<ak,fk>
PERSONID	VARCHAR2(20)	<pk>
PERSONCATCODE	VARCHAR2(8)	
LASTNAME	VARCHAR2(40)	
FIRSTNAME	VARCHAR2(40)	
MIDDLENAME	VARCHAR2(30)	
SUFFIX	VARCHAR2(10)	
TITLE	VARCHAR2(8)	
DOB	DATE	
DEATH	VARCHAR2(1)	
GENDER	VARCHAR2(8)	
MSTATUS	VARCHAR2(8)	
SSN	VARCHAR2(16)	
RACE	VARCHAR2(8)	
ETHNIC	VARCHAR2(8)	
RELIGION	VARCHAR2(8)	
LANGUAGE	VARCHAR2(8)	
SPOUSENAME	VARCHAR2(100)	
MOTHERNAME	VARCHAR2(100)	
MOTHERMN	VARCHAR2(40)	
FATHERNAME	VARCHAR2(100)	
MAIDEN	VARCHAR2(40)	
POBCITY	VARCHAR2(30)	
POBSTATE	VARCHAR2(10)	
POBCOUNTRY	VARCHAR2(20)	
VIPFLAG	VARCHAR2(8)	
VETSTATUS	VARCHAR2(8)	
FNAMEPHONETICCODE	VARCHAR2(8)	
LNAMEPHONETICCODE	VARCHAR2(8)	
MNAMEPHONETICCODE	VARCHAR2(8)	
MOTHERMNPHONETICCODE	VARCHAR2(8)	
MAIDENPHONETICCODE	VARCHAR2(8)	
SPOUSEPHONETICCODE	VARCHAR2(8)	
MOTHERPHONETICCODE	VARCHAR2(8)	
FATHERPHONETICCODE	VARCHAR2(8)	
DRIVERSLICENSE	VARCHAR2(20)	
DRIVERSLICENSEST	VARCHAR2(10)	
DOD	DATE	
DEATHCERTIFICATE	VARCHAR2(10)	
NATIONALITY	VARCHAR2(8)	
CITIZENSHIP	VARCHAR2(8)	
PENSIONNO	VARCHAR2(15)	
PENSIONEXPDATE	DATE	
REPATRIATIONNO	VARCHAR2(16)	
DISTRICTOFRESIDENCE	VARCHAR2(8)	
LGACODE	VARCHAR2(4)	
MILITARYBRANCH	VARCHAR2(4)	
MILITARYRANK	VARCHAR2(4)	
MILITARYSTATUS	VARCHAR2(4)	
DUMMYDATE	DATE	
CLASS1	VARCHAR2(20)	
CLASS2	VARCHAR2(20)	
CLASS3	VARCHAR2(20)	
CLASS4	VARCHAR2(20)	
CLASS5	VARCHAR2(20)	
STRING1	VARCHAR2(40)	
STRING2	VARCHAR2(40)	
STRING3	VARCHAR2(40)	
STRING4	VARCHAR2(40)	
STRING5	VARCHAR2(40)	
STRING6	VARCHAR2(40)	
STRING7	VARCHAR2(100)	
STRING8	VARCHAR2(100)	
STRING9	VARCHAR2(100)	
STRING10	VARCHAR2(255)	
DATE1	DATE	
DATE2	DATE	
DATE3	DATE	
DATE4	DATE	
DATE5	DATE	
STDFIRSTNAME	VARCHAR2(40)	
STDLASTNAME	VARCHAR2(40)	
STDMIDDLENAME	VARCHAR2(30)	

SBYN_ADDRESS		
PERSONID	VARCHAR2(20)	<ak,fk>
ADDRESSID	VARCHAR2(20)	<pk>
ADDRESSTYPE	VARCHAR2(8)	<ak>
ADDRESSLINE1	VARCHAR2(40)	
ADDRESSLINE2	VARCHAR2(40)	
ADDRESSLINE3	VARCHAR2(40)	
ADDRESSLINE4	VARCHAR2(40)	
CITY	VARCHAR2(30)	
STATECODE	VARCHAR2(10)	
POSTALCODE	VARCHAR2(8)	
POSTALCODEEXT	VARCHAR2(4)	
COUNTY	VARCHAR2(20)	
COUNTRYCODE	VARCHAR2(20)	
HOUSENUMBER	VARCHAR2(10)	
STREETDIR	VARCHAR2(5)	
STREETNAME	VARCHAR2(40)	
STREETNAMEPHONETICCODE	VARCHAR2(8)	
STREETTYPE	VARCHAR2(5)	

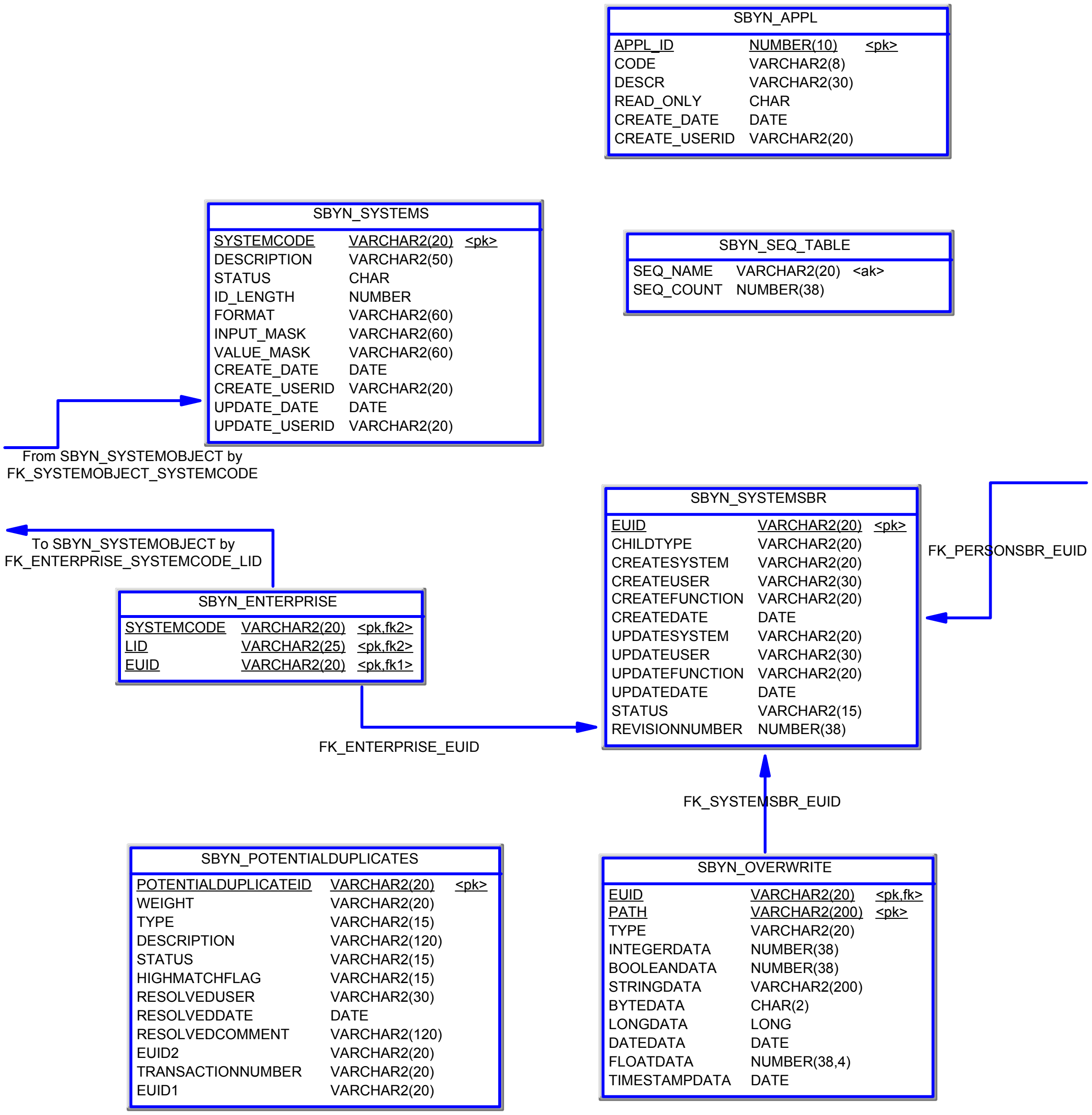
SBYN_PHONE		
PERSONID	VARCHAR2(20)	<ak,fk>
PHONEID	VARCHAR2(20)	<pk>
PHONETYPE	VARCHAR2(8)	<ak>
PHONE	VARCHAR2(20)	
PHONEEXT	VARCHAR2(6)	

SBYN_SYSTEMOBJECT		
SYSTEMCODE	VARCHAR2(20)	<pk,fk>
LID	VARCHAR2(25)	<pk>
CHILDTYPE	VARCHAR2(20)	
CREATEUSER	VARCHAR2(30)	
CREATEFUNCTION	VARCHAR2(20)	
CREATEDATE	DATE	
UPDATEUSER	VARCHAR2(30)	
UPDATEFUNCTION	VARCHAR2(20)	
UPDATEDATE	DATE	
STATUS	VARCHAR2(15)	

SBYN_COMMENT		
PERSONID	VARCHAR2(20)	<ak,fk>
COMMENTID	VARCHAR2(20)	<pk>
COMMENTCODE	VARCHAR2(8)	<ak>
ENTERDATE	DATE	
COMMENTTEXT	VARCHAR2(1000)	

SBYN_AUXID		
PERSONID	VARCHAR2(20)	<ak,fk>
AUXIDID	VARCHAR2(20)	<pk>
AUXIDDEF	VARCHAR2(10)	<ak>
ID	VARCHAR2(40)	<ak>

SBYN_ALIAS		
PERSONID	VARCHAR2(20)	<ak,fk>
ALIASID	VARCHAR2(20)	<pk>
LASTNAME	VARCHAR2(40)	<ak>
FIRSTNAME	VARCHAR2(40)	<ak>
MIDDLENAME	VARCHAR2(30)	<ak>
LNAMEPHONETICCODE	VARCHAR2(8)	
FNAMEPHONETICCODE	VARCHAR2(8)	
MNAMEPHONETICCODE	VARCHAR2(8)	
STDFIRSTNAME	VARCHAR2(40)	
STDLASTNAME	VARCHAR2(40)	
STDMIDDLENAME	VARCHAR2(30)	


From SBYN_SYSTEMOBJECT by
FK_SYSTEMOBJECT_SYSTEMCODE
To SBYN_SYSTEMOBJECT by
FK_ENTERPRISE_SYSTEMCODE_LID
FK_ENTERPRISE_EUID
FK_PERSONSBR_EUID
FK_SYSTEMSBR_EUID

SBYN_APPL

APPL_ID	NUMBER(10)	<pk>
CODE	VARCHAR2(8)	
DESCR	VARCHAR2(30)	
READ_ONLY	CHAR	
CREATE_DATE	DATE	
CREATE_USERID	VARCHAR2(20)	

SBYN_SYSTEMS

SYSTEMCODE	VARCHAR2(20)	<pk>
DESCRIPTION	VARCHAR2(50)	
STATUS	CHAR	
ID_LENGTH	NUMBER	
FORMAT	VARCHAR2(60)	
INPUT_MASK	VARCHAR2(60)	
VALUE_MASK	VARCHAR2(60)	
CREATE_DATE	DATE	
CREATE_USERID	VARCHAR2(20)	
UPDATE_DATE	DATE	
UPDATE_USERID	VARCHAR2(20)	

SBYN_SEQ_TABLE

SEQ_NAME	VARCHAR2(20)	<ak>
SEQ_COUNT	NUMBER(38)	

SBYN_SYSTEMSBR

EUID	VARCHAR2(20)	<pk>
CHILDTYPE	VARCHAR2(20)	
CREATESYSTEM	VARCHAR2(20)	
CREATEUSER	VARCHAR2(30)	
CREATEFUNCTION	VARCHAR2(20)	
CREATEDATE	DATE	
UPDATESYSTEM	VARCHAR2(20)	
UPDATEUSER	VARCHAR2(30)	
UPDATEFUNCTION	VARCHAR2(20)	
UPDATEDATE	DATE	
STATUS	VARCHAR2(15)	
REVISIONNUMBER	NUMBER(38)	

SBYN_ENTERPRISE

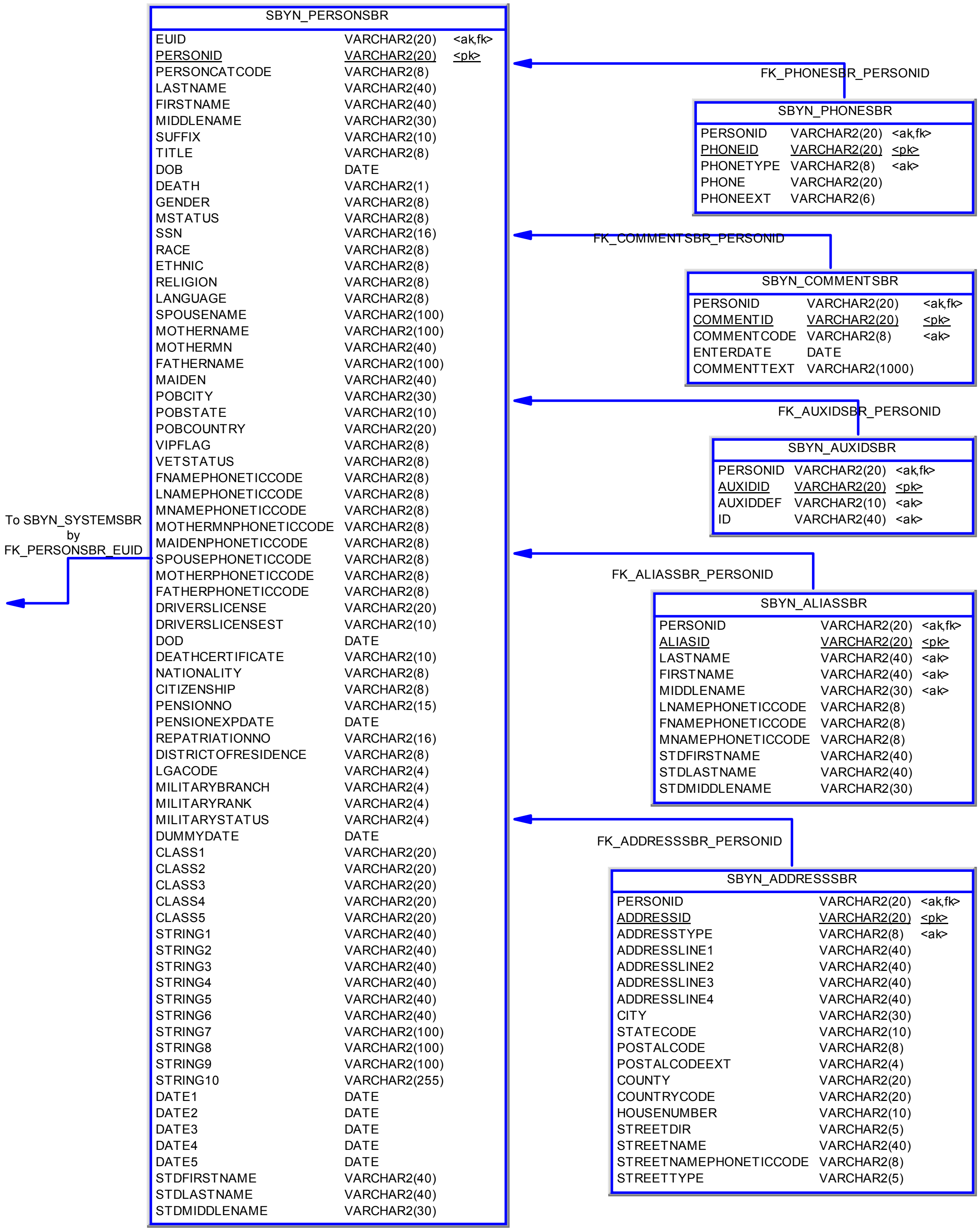
SYSTEMCODE	VARCHAR2(20)	<pk,fk2>
LID	VARCHAR2(25)	<pk,fk2>
EUID	VARCHAR2(20)	<pk,fk1>

SBYN_POTENTIALDUPLICATES

POTENTIALDUPLICATEID	VARCHAR2(20)	<pk>
WEIGHT	VARCHAR2(20)	
TYPE	VARCHAR2(15)	
DESCRIPTION	VARCHAR2(120)	
STATUS	VARCHAR2(15)	
HIGHMATCHFLAG	VARCHAR2(15)	
RESOLVEDUSER	VARCHAR2(30)	
RESOLVEDDATE	DATE	
RESOLVEDCOMMENT	VARCHAR2(120)	
EUID2	VARCHAR2(20)	
TRANSACTIONNUMBER	VARCHAR2(20)	
EUID1	VARCHAR2(20)	

SBYN_OVERWRITE

EUID	VARCHAR2(20)	<pk,fk>
PATH	VARCHAR2(200)	<pk>
TYPE	VARCHAR2(20)	
INTEGERDATA	NUMBER(38)	
BOOLEANDATA	NUMBER(38)	
STRINGDATA	VARCHAR2(200)	
BYTEDATA	CHAR(2)	
LONGDATA	LONG	
DATEDATA	DATE	
FLOATDATA	NUMBER(38,4)	
TIMESTAMPDATA	DATE	



SBYN_PERSONSBR		
EUID	VARCHAR2(20)	<ak,fk>
PERSONID	VARCHAR2(20)	<pk>
PERSONCATCODE	VARCHAR2(8)	
LASTNAME	VARCHAR2(40)	
FIRSTNAME	VARCHAR2(40)	
MIDDLENAME	VARCHAR2(30)	
SUFFIX	VARCHAR2(10)	
TITLE	VARCHAR2(8)	
DOB	DATE	
DEATH	VARCHAR2(1)	
GENDER	VARCHAR2(8)	
MSTATUS	VARCHAR2(8)	
SSN	VARCHAR2(16)	
RACE	VARCHAR2(8)	
ETHNIC	VARCHAR2(8)	
RELIGION	VARCHAR2(8)	
LANGUAGE	VARCHAR2(8)	
SPOUSENAME	VARCHAR2(100)	
MOTHERNAME	VARCHAR2(100)	
MOTHERMN	VARCHAR2(40)	
FATHERNAME	VARCHAR2(100)	
MAIDEN	VARCHAR2(40)	
POBCITY	VARCHAR2(30)	
POBSTATE	VARCHAR2(10)	
POBCOUNTRY	VARCHAR2(20)	
VIPFLAG	VARCHAR2(8)	
VETSTATUS	VARCHAR2(8)	
FNAMEPHONETICCODE	VARCHAR2(8)	
LNAMEPHONETICCODE	VARCHAR2(8)	
MNAMEPHONETICCODE	VARCHAR2(8)	
MOTHERMNPHONETICCODE	VARCHAR2(8)	
MAIDENPHONETICCODE	VARCHAR2(8)	
SPOUSEPHONETICCODE	VARCHAR2(8)	
MOTHERPHONETICCODE	VARCHAR2(8)	
FATHERPHONETICCODE	VARCHAR2(8)	
DRIVERSLICENSE	VARCHAR2(20)	
DRIVERSLICENSEST	VARCHAR2(10)	
DOD	DATE	
DEATHCERTIFICATE	VARCHAR2(10)	
NATIONALITY	VARCHAR2(8)	
CITIZENSHIP	VARCHAR2(8)	
PENSIONNO	VARCHAR2(15)	
PENSIONEXPDATE	DATE	
REPATRIATIONNO	VARCHAR2(16)	
DISTRICTOFRESIDENCE	VARCHAR2(8)	
LGACODE	VARCHAR2(4)	
MILITARYBRANCH	VARCHAR2(4)	
MILITARYRANK	VARCHAR2(4)	
MILITARYSTATUS	VARCHAR2(4)	
DUMMYDATE	DATE	
CLASS1	VARCHAR2(20)	
CLASS2	VARCHAR2(20)	
CLASS3	VARCHAR2(20)	
CLASS4	VARCHAR2(20)	
CLASS5	VARCHAR2(20)	
STRING1	VARCHAR2(40)	
STRING2	VARCHAR2(40)	
STRING3	VARCHAR2(40)	
STRING4	VARCHAR2(40)	
STRING5	VARCHAR2(40)	
STRING6	VARCHAR2(40)	
STRING7	VARCHAR2(100)	
STRING8	VARCHAR2(100)	
STRING9	VARCHAR2(100)	
STRING10	VARCHAR2(255)	
DATE1	DATE	
DATE2	DATE	
DATE3	DATE	
DATE4	DATE	
DATE5	DATE	
STDFIRSTNAME	VARCHAR2(40)	
STDLASTNAME	VARCHAR2(40)	
STDMIDDLENAME	VARCHAR2(30)	

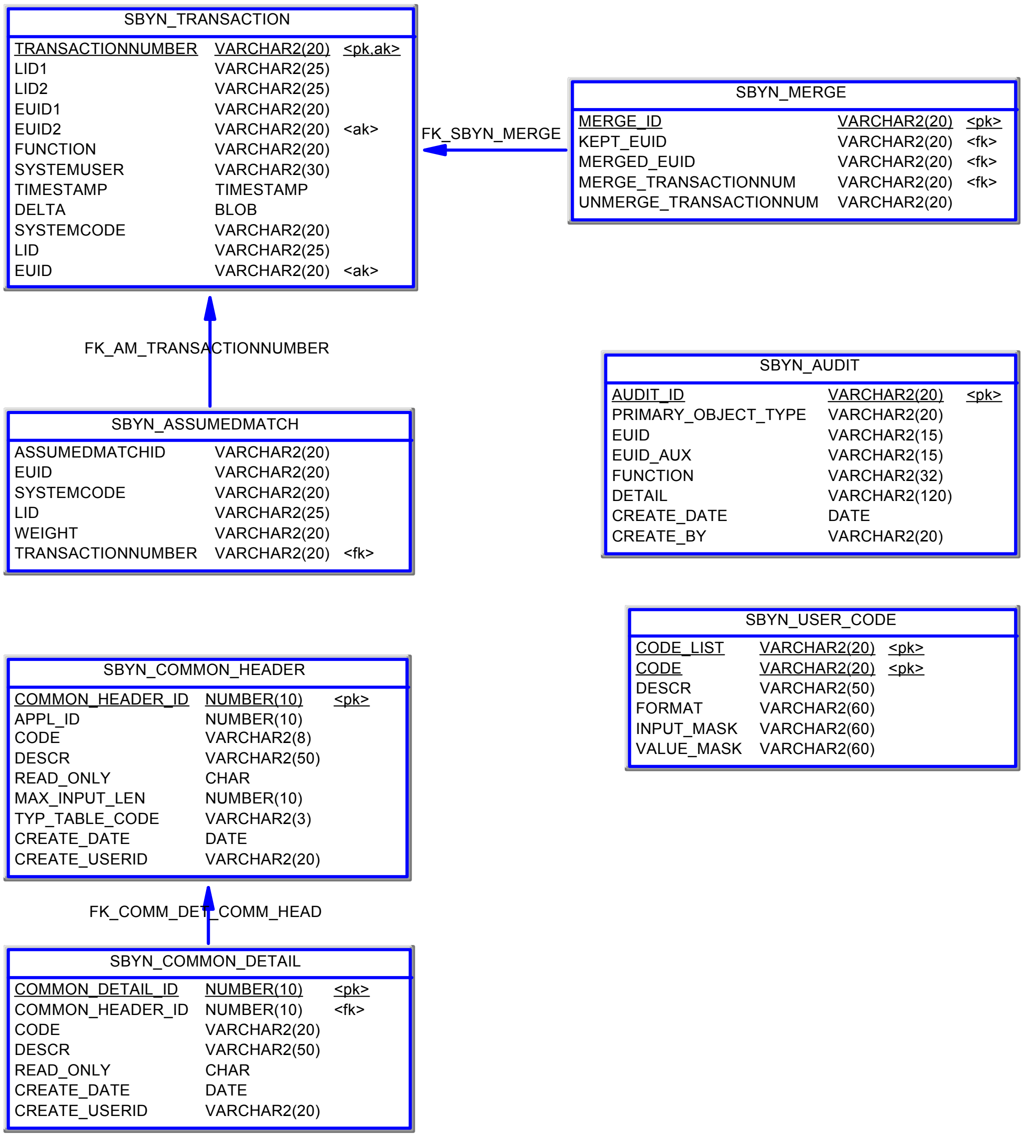
SBYN_PHONESBR		
PERSONID	VARCHAR2(20)	<ak,fk>
PHONEID	VARCHAR2(20)	<pk>
PHONETYPE	VARCHAR2(8)	<ak>
PHONE	VARCHAR2(20)	
PHONEEXT	VARCHAR2(6)	

SBYN_COMMENTSBR		
PERSONID	VARCHAR2(20)	<ak,fk>
COMMENTID	VARCHAR2(20)	<pk>
COMMENTCODE	VARCHAR2(8)	<ak>
ENTERDATE	DATE	
COMMENTTEXT	VARCHAR2(1000)	

SBYN_AUXIDSBR		
PERSONID	VARCHAR2(20)	<ak,fk>
AUXIDID	VARCHAR2(20)	<pk>
AUXIDDEF	VARCHAR2(10)	<ak>
ID	VARCHAR2(40)	<ak>

SBYN_ALIASSBR		
PERSONID	VARCHAR2(20)	<ak,fk>
ALIASID	VARCHAR2(20)	<pk>
LASTNAME	VARCHAR2(40)	<ak>
FIRSTNAME	VARCHAR2(40)	<ak>
MIDDLENAME	VARCHAR2(30)	<ak>
LNAMEPHONETICCODE	VARCHAR2(8)	
FNAMEPHONETICCODE	VARCHAR2(8)	
MNAMEPHONETICCODE	VARCHAR2(8)	
STDFIRSTNAME	VARCHAR2(40)	
STDLASTNAME	VARCHAR2(40)	
STDMIDDLENAME	VARCHAR2(30)	

SBYN_ADDRESSSBR		
PERSONID	VARCHAR2(20)	<ak,fk>
ADDRESSID	VARCHAR2(20)	<pk>
ADDRESSTYPE	VARCHAR2(8)	<ak>
ADDRESSLINE1	VARCHAR2(40)	
ADDRESSLINE2	VARCHAR2(40)	
ADDRESSLINE3	VARCHAR2(40)	
ADDRESSLINE4	VARCHAR2(40)	
CITY	VARCHAR2(30)	
STATECODE	VARCHAR2(10)	
POSTALCODE	VARCHAR2(8)	
POSTALCODEEXT	VARCHAR2(4)	
COUNTY	VARCHAR2(20)	
COUNTRYCODE	VARCHAR2(20)	
HOUSENUMBER	VARCHAR2(10)	
STREETDIR	VARCHAR2(5)	
STREETNAME	VARCHAR2(40)	
STREETNAMEPHONETICCODE	VARCHAR2(8)	
STREETTYPE	VARCHAR2(5)	



SBYN_TRANSACTION

TRANSACTIONNUMBER	VARCHAR2(20)	<pk,ak>
LID1	VARCHAR2(25)	
LID2	VARCHAR2(25)	
EUID1	VARCHAR2(20)	
EUID2	VARCHAR2(20)	<ak>
FUNCTION	VARCHAR2(20)	
SYSTEMUSER	VARCHAR2(30)	
TIMESTAMP	TIMESTAMP	
DELTA	BLOB	
SYSTEMCODE	VARCHAR2(20)	
LID	VARCHAR2(25)	
EUID	VARCHAR2(20)	<ak>

SBYN_MERGE

MERGE_ID	VARCHAR2(20)	<pk>
KEPT_EUID	VARCHAR2(20)	<fk>
MERGED_EUID	VARCHAR2(20)	<fk>
MERGE_TRANSACTIONNUM	VARCHAR2(20)	<fk>
UNMERGE_TRANSACTIONNUM	VARCHAR2(20)	

SBYN_ASSUMEDMATCH

ASSUMEDMATCHID	VARCHAR2(20)	
EUID	VARCHAR2(20)	
SYSTEMCODE	VARCHAR2(20)	
LID	VARCHAR2(25)	
WEIGHT	VARCHAR2(20)	
TRANSACTIONNUMBER	VARCHAR2(20)	<fk>

SBYN_AUDIT

AUDIT_ID	VARCHAR2(20)	<pk>
PRIMARY_OBJECT_TYPE	VARCHAR2(20)	
EUID	VARCHAR2(15)	
EUID_AUX	VARCHAR2(15)	
FUNCTION	VARCHAR2(32)	
DETAIL	VARCHAR2(120)	
CREATE_DATE	DATE	
CREATE_BY	VARCHAR2(20)	

SBYN_USER_CODE

CODE_LIST	VARCHAR2(20)	<pk>
CODE	VARCHAR2(20)	<pk>
DESCR	VARCHAR2(50)	
FORMAT	VARCHAR2(60)	
INPUT_MASK	VARCHAR2(60)	
VALUE_MASK	VARCHAR2(60)	

SBYN_COMMON_HEADER

COMMON_HEADER_ID	NUMBER(10)	<pk>
APPL_ID	NUMBER(10)	
CODE	VARCHAR2(8)	
DESCR	VARCHAR2(50)	
READ_ONLY	CHAR	
MAX_INPUT_LEN	NUMBER(10)	
TYP_TABLE_CODE	VARCHAR2(3)	
CREATE_DATE	DATE	
CREATE_USERID	VARCHAR2(20)	

SBYN_COMMON_DETAIL

COMMON_DETAIL_ID	NUMBER(10)	<pk>
COMMON_HEADER_ID	NUMBER(10)	<fk>
CODE	VARCHAR2(20)	
DESCR	VARCHAR2(50)	
READ_ONLY	CHAR	
CREATE_DATE	DATE	
CREATE_USERID	VARCHAR2(20)	

Chapter 4

Working with the Java API

eIndex SPV provides several Java classes and methods to use in the Collaborations for an eIndex SPV Project. The eIndex SPV 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.

What's in This Chapter

4.1 Overview

This chapter provides an overview of the Java API for eIndex SPV, and describes the dynamic classes and methods that are generated based on the object structure of the master index. For detailed information about the static classes and methods, refer to the eIndex SPV Javadocs, provided as a download through the Enterprise Manager. 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.

4.2 Java Class Types

eIndex SPV provides a set of static API classes that can be used with any object structure. eIndex SPV also generates several dynamic API classes that are specific to the object structure. The dynamic classes contain similar methods, but the number and names of methods change depending on the object structure. In addition, several methods are generated in an OTD for use in external system Collaborations and another set of methods is generated for use within an eInsight Business Process.

Static Classes

Static classes provide the methods you need to perform basic data cleansing 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** method. Several classes support the MasterController class by defining additional objects and functions. Documentation for the static methods is provided in Javadoc format. The static classes are listed and described in the Javadocs provided with eIndex SPV.

Dynamic Object Classes

The eIndex SPV Project provides several dynamic methods that are specific to the default object structure. If the object structure is modified, regenerating the Project updates the dynamic methods for the new structure. This includes classes that define each object in the object structure and that allow you to work with the data in each object.

Dynamic OTD Methods

The eIndex SPV Project provides a method OTD that contains Java methods to help you define how records will be processed into the database from external systems. Like the dynamic classes, these methods are based on the object structure. Regenerating a Project updates these methods to reflect any changes to the object structure. These methods rely on the dynamic object classes to create objects in eIndex SPV and to define and retrieve field values for those objects.

Dynamic Business Process Methods

The eIndex SPV Project includes several methods under the method OTD folder that are designed for use within an eInsight Business Process. These methods are a subset of the eIndex SPV API and can be used to query eIndex SPV using a web-based interface. These methods are also based on the defined object structure. Regenerating a Project updates these methods to reflect any changes to the object structure.

4.3 Dynamic Object Classes

Several dynamic classes are included in each eIndex SPV Project for use in Collaborations. One class is created for each parent and child object defined in the Object Structure.

4.3.1 Parent Object Classes

A Java class is created to represent the parent object defined in the object definition of the master index. The methods in these classes provide the ability to create a parent object and to set or retrieve the field values for that object.

The name of the parent object class is the same as the name of each parent object, with the word "Object" appended (by default, **PersonObject**). The methods in this class include a constructor method for the parent object, and get and set methods for each field defined for the parent object. 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 following methods described for the parent object, <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

```
class <ObjectName>Object
```

Methods

- **<ObjectName>Object** on page 71
- **add<Child>** on page 72
- **addSecondaryObject** on page 72
- **copy** on page 73
- **dropSecondaryObject** on page 73
- **get<ObjectName>Id** on page 74
- **get<Child>** on page 74
- **get<Field>** on page 74
- **getChildTags** on page 75
- **getMetaData** on page 75
- **getSecondaryObject** on page 76
- **isAdded** on page 76
- **isRemoved** on page 76
- **isUpdated** on page 77
- **set<ObjectName>Id** on page 77
- **set<Field>** on page 78
- **setAddFlag** on page 78
- **setRemoveFlag** on page 79
- **setUpdateFlag** on page 79
- **structCopy** on page 80

<ObjectName>Object

Description

<ObjectName>Object is the user-defined object name class. You can instantiate this class to create a new instance of the parent object class.

Syntax

new <ObjectName>Object()

Parameters

None.

Returns

An instance of the parent object.

Throws

ObjectException

add<Child>

Description

add<Child> associates a new child object with the parent object. The new child object is of the type specified in the method name. For example, to associate a new address object with a parent object, call “addAddress”.

Syntax

```
void add<Child>(<Child>Object <child>)
```

Note: *The type of object passed as a parameter depends on the child object to associate with the parent object. For example, the syntax for associating an address object is as follows:* `void addAddress(AddressObject address)`*.*

Parameters

Name	Type	Description
<child>	<Child>Object	A child object to associate with the parent object. The name and type of the parameter is specified by the child object name.

Returns

None.

Throws

ObjectException

addSecondaryObject

Description

addSecondaryObject associates a new child object with the parent object. The object node passed as the parameter defines the child object type.

Syntax

```
void addSecondaryObject(ObjectNode obj)
```

Parameters

Name	Type	Description
obj	ObjectNode	An ObjectNode representing the child object to associate with the parent object.

Returns

None.

Throws

SystemObjectException

copy

Description

copy copies the structure and field values of the specified object node.

Syntax

```
ObjectNode copy()
```

Parameters

None.

Returns

A copy of the object node.

Throws

ObjectException

dropSecondaryObject

Description

dropSecondaryObject removes a child object associated with the parent object (in the memory copy of the object). The object node passed in as the parameter defines the child object type. Use this method to remove a child object before it has been committed to the database. This method is similar to ObjectNode.removeChild. Use ObjectNode.deleteChild to remove the child object permanently from the database.

Syntax

```
void dropSecondaryObject(ObjectNode obj)
```

Parameters

Name	Type	Description
obj	ObjectNode	An ObjectNode representing the child object to drop from the parent object.

Returns

None.

Throws

SystemObjectException

get<ObjectName>Id

Description

get<ObjectName>Id retrieves the unique identification code (primary key) of the object, as assigned by the master index.

Syntax

```
String get<ObjectName>Id()
```

Parameters

None.

Returns

A string containing the unique ID of the parent object.

Throws

ObjectException

get<Child>

Description

get<Child> retrieves all child objects associated with the parent object that are of the type specified in the method name. For example, to retrieve all address objects associated with a parent object, call “getAddress”.

Syntax

```
Collection get<Child>()
```

Parameters

None.

Returns

A collection of child objects of the type specified in the method name.

Throws

None.

get<Field>

Description

get<Field> 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 get<Field>()
```

Note: *The syntax for the getter methods depends of the type of data specified for the field in the object structure. For example, the getter method for a date field would have the following syntax:* `Date get<Field>`*.*

Parameters

None.

Returns

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

Throws

ObjectException

getChildTags

Description

getChildTags retrieves a list of the names of all child object types defined for the object structure.

Syntax

```
ArrayList getChildTags()
```

Parameters

None.

Returns

An array of child object names.

Throws

None

getMetaData

Description

getMetaData retrieves the metadata for the parent object.

Syntax

```
AttributeMetaData getMetaData()
```

Parameters

None.

Returns

An AttributeMetaData object containing the parent object's metadata.

Throws

None.

getSecondaryObject

Description

getSecondaryObject retrieves all child objects that are associated with the parent object and are of the specified type.

Syntax

```
Collection getSecondaryObject(String type)
```

Parameters

Name	Type	Description
type	String	The child type of the objects to retrieve.

Returns

A collection of child objects of the specified type.

Throws

SystemObjectException

isAdded

Description

isAdded retrieves the value of the “add flag” for the parent object. The add flag indicates whether the object will be added.

Syntax

```
String isAdded()
```

Parameters

None.

Returns

A Boolean value indicating whether the add flag is set to true or false.

Throws

ObjectException

isRemoved

Description

isRemoved retrieves the value of the “remove flag” for the parent object. The remove flag indicates whether the object will be removed.

Syntax

```
String isRemoved()
```

Parameters

None.

Returns

A Boolean value indicating whether the remove flag is set to true or false.

Throws

ObjectException

isUpdated

Description

isUpdated retrieves the value of the "update flag" for the parent object. The updated flag indicates whether the object will be updated.

Syntax

```
String isUpdated()
```

Parameters

None.

Returns

A Boolean value indicating whether the update flag is set to true or false.

Throws

ObjectException

set<ObjectName>Id

Description

set<ObjectName>Id sets the value of the **<ObjectName>Id** field in the parent object.

Syntax

```
void set<ObjectName>Id(Object value)
```

Parameters

Name	Type	Description
value	Object	An object containing the value of the **<ObjectName>Id** field.

Returns

None.

Throws

ObjectException

set<Field>

Description

set<Field> 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

```
void set<Field>(Object value)
```

Parameters

Name	Type	Description
value	Object	An object containing the value of the field specified by the method name.

Returns

None.

Throws

ObjectException

setAddFlag

Description

setAddFlag sets the "add flag" of the parent object. The add flag indicates whether the object will be added.

Syntax

```
void setAddFlag(boolean flag)
```

Parameters

Name	Type	Description
flag	Boolean	An indicator of whether the add flag is set to true or false.

Returns

None.

Throws

None.

setRemoveFlag

Description

setRemoveFlag sets the "remove flag" of the parent object. The remove flag indicates whether the object will be removed.

Syntax

```
void setRemoveFlag(boolean e)
```

Parameters

Name	Type	Description
e	Boolean	An indicator of whether the remove flag is set to true or false.

Returns

None.

Throws

None.

setUpdateFlag

Description

setUpdateFlag sets the "update flag" of the parent object. The update flag indicates whether the object will be updated.

Syntax

```
void setUpdateFlag(boolean flag)
```

Parameters

Name	Type	Description
flag	Boolean	An indicator of whether the update flag is set to true or false.

Returns

None.

Throws

None.

structCopy

Description

structCopy copies the structure of the specified object node.

Syntax

```
ObjectNode structCopy()
```

Parameters

None.

Returns

A copy of the structure of the object node.

Throws

ObjectException

4.3.2 Child Object Classes

One Java class is created for each child object defined in the object definition of the master index. If the object definition contains three child objects, three child object classes are created. The methods in these classes provide the ability to create the child objects and to set or retrieve the field values for those objects.

The name of each child object class is the same as the name of the child object, with the word "Object" appended. For example, if a child object in your object structure is named "Address", the name of the corresponding child class is "AddressObject". The methods in these classes include a constructor method for the child object, 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 methods listed below, <Child> indicates the name of the child object and <Field> indicates the name of each field defined for that object.

Definition

```
class <Child>Object
```

Methods

- **<Child>Object** on page 81
- **copy** on page 81
- **get<Child>Id** on page 81
- **get<Field>** on page 82
- **getMetaData** on page 82
- **getParentTag** on page 83
- **set<Child>Id** on page 83
- **set<Field>** on page 84
- **structCopy** on page 84

<Child>Object

Description

<Child>Object is the child object class. This class can be instantiated to create a new instance of a child object class.

Syntax

```
new <Child>Object()
```

Parameters

None.

Returns

An instance of the child object.

Throws

ObjectException

copy

Description

copy copies the structure and field values of the specified object node.

Syntax

```
ObjectNode copy()
```

Parameters

None.

Returns

A copy of the object node.

Throws

ObjectException

get<Child>Id

Description

get<Child>Id retrieves the unique identification code (primary key) of the object, as assigned by the master index.

Syntax

```
String get<Child>Id()
```

Parameters

None.

Returns

A string containing the unique ID of the child object.

Throws

ObjectException

get<Field>

Description

get<Field> 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 "TelephoneNumber", the getter method for this field is named "getTelephoneNumber". A getter method is created for each field in the object, including fields that store standardized or phonetic data.

Syntax

```
String get<Field>()
```

Note: *The syntax for the getter methods depends on the type of data specified for the field in the object structure. For example, the getter method for a date field would have the following syntax:* `Date` *get<Field>.*

Parameters

None.

Returns

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

Throws

ObjectException

getMetaData

Description

getMetaData retrieves the metadata for the child object.

Syntax

```
AttributeMetaData getMetaData()
```

Parameters

None.

Returns

An AttributeMetaData object containing the child object's metadata.

Throws

None.

getParentTag

Description

getParentTag retrieves the name of the parent object of the child object.

Syntax

```
String getParentTag()
```

Parameters

None.

Returns

A string containing the name of the parent object.

Throws

None.

set<Child>Id

Description

set<Child>Id sets the value of the **<Child>Id** field in the child object.

Syntax

```
void set<Child>Id(Object value)
```

Parameters

Name	Type	Description
value	Object	An object containing the value of the **<Child>Id** field.

Returns

None.

Throws

ObjectException

set<Field>

Description

set<Field> 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".

Syntax

```
void set<Field>(Object value)
```

Parameters

Name	Type	Description
value	Object	An object containing the value of the field specified by the method name.

Returns

None.

Throws

ObjectException

structCopy

Description

structCopy copies the structure of the specified object node.

Syntax

```
ObjectNode structCopy()
```

Parameters

None.

Returns

A copy of the structure of the object node.

Throws

ObjectException

4.4 Dynamic OTD Methods

A set of Java methods are created in an OTD for use in the eIndex SPV Collaborations. These methods wrap static Java API methods, allowing them to work with the dynamic object classes. Many OTD methods return objects of the dynamic object type, or they use these objects as parameters. In the following methods described for the OTD methods, <ObjectName> indicates the name of the parent object.

- **activateEnterpriseRecord** on page 85
- **activateSystemRecord** on page 86
- **addSystemRecord** on page 86
- **deactivateEnterpriseRecord** on page 87
- **deactivateSystemRecord** on page 87
- **executeMatch** on page 88
- **executeMatchUpdate** on page 89
- **findMasterController** on page 90
- **getEnterpriseRecordByEUID** on page 90
- **getEnterpriseRecordByLID** on page 91
- **getEUID** on page 91
- **getLIDs** on page 92
- **getLIDsByStatus** on page 92
- **getSBR** on page 93
- **getSystemRecord** on page 94
- **getSystemRecordsByEUID** on page 94
- **getSystemRecordsByEUIDStatus** on page 95
- **lookupLIDs** on page 95
- **mergeEnterpriseRecord** on page 96
- **mergeSystemRecord** on page 97
- **searchBlock** on page 97
- **searchExact** on page 98
- **searchPhonetic** on page 99
- **transferSystemRecord** on page 99
- **updateEnterpriseRecord** on page 100
- **updateSystemRecord** on page 100

activateEnterpriseRecord

Description

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

Syntax

```
void activateEnterpriseRecord(String euid)
```

Parameters

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

Returns

None.

Throws

RemoteException

ProcessingException

UserException

activateSystemRecord

Description

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

Syntax

```
void activateSystemRecord(String systemCode, String localId)
```

Parameters

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

Returns

None.

Throws

RemoteException

ProcessingException

UserException

addSystemRecord

Description

addSystemRecord adds the system object to the enterprise object associated with the specified EUID.

Syntax

```
void addSystemRecord(String euid, SystemObjectBean systemObject)
```

Parameters

Name	Type	Description
euid	String	The EUID of the enterprise object to which you want to add the system object.
systemObject	SystemObjectBean	The Bean for the system object to be added to the enterprise object.

Returns

None.

Throws

RemoteException

ProcessingException

UserException

deactivateEnterpriseRecord

Description

deactivateEnterpriseRecord changes the status of an active enterprise object to inactive.

Syntax

```
void deactivateEnterpriseRecord(String euid)
```

Parameters

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

Returns

None.

Throws

RemoteException

ProcessingException

UserException

deactivateSystemRecord

Description

deactivateSystemRecord changes the status of an active system object to inactive.

Syntax

```
void deactivateSystemRecord(String systemCode, String localId)
```

Parameters

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

Returns

None.

Throws

RemoteException

ProcessingException

UserException

executeMatch

executeMatch is one of two methods you can call to process an incoming system object based on the configuration defined for the eIndex SPV Manager Service and associated runtime components (the second method is **executeMatchUpdate** on page 89). This process searches for possible matches in the database and contains the logic to add a new record or update existing records in the database. One of the two execute match methods should be used for inserting or updating a record in the database.

The following runtime components configure **executeMatch**.

- The Query Builder defines the blocking queries used for matching.
- The Threshold file 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.

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

Syntax

```
MatchColResult executeMatch(SystemObjectBean systemObject)
```

Parameters

Name	Type	Description
systemObject	SystemObjectBean	The Bean for the system object to be added to or updated in the enterprise object.

Returns

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

Throws

RemoteException

ProcessingException

UserException

executeMatchUpdate

Like **executeMatch** on page 88, **executeMatchUpdate** processes the system object based on the configuration defined for the eIndex SPV Manager Service and associated runtime components. It is configured by the same runtime components as **executeMatch**. One of the 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(SystemObjectbean systemObject)
```

Parameters

Name	Type	Description
systemObject	SystemObjectBean	The Bean for the system object to be added to or updated in the enterprise object.

Returns

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

Throws

RemoteException

ProcessingException

UserException

findMasterController

findMasterController obtains a handle to the MasterController class, providing access to all of the methods of that class. For more information about the available methods in this class, see the Javadoc provided with eIndex SPV.

Syntax

```
MasterController findMasterController()
```

Parameters

None.

Returns

A handle to the **com.stc.eindex.ejb.master.MasterController** class.

Throws

None.

getEnterpriseRecordByEUID

Description

getEnterpriseRecordByEUID returns the enterprise object associated with the specified EUID.

Syntax

```
Enterprise<ObjectName> getEnterpriseRecordByEUID(String euid)
```

Parameters

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

Returns

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

Throws

RemoteException

ProcessingException

UserException

getEnterpriseRecordByLID

Description

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

Syntax

```
Enterprise<ObjectName> getEnterpriseRecordByLID(String system, String
localid)
```

Parameters

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

Returns

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

Throws

RemoteException

ProcessingException

UserException

getEUID

Description

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

Syntax

```
String getEUID(String system, String localid)
```

Parameters

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

Returns

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

Throws

RemoteException

ProcessingException

UserException

getLIDs

Description

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

Syntax

```
System<ObjectName>PK[] getLIDs(String euid)
```

Parameters

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

Returns

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

Throws

RemoteException

ProcessingException

UserException

getLIDsByStatus

Description

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

Syntax

```
System<ObjectName>PK[] getLIDsByStatus(String euid, String status)
```

Parameters

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

Returns

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

Throws

RemoteException

ProcessingException

UserException

getSBR

Description

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

Syntax

```
SBR<ObjectName> getSBR(String euid)
```

Parameters

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

Returns

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

Throws

RemoteException

ProcessingException

UserException

getSystemRecord

Description

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

Syntax

```
System<ObjectName> getSystemRecord(String system, String localid)
```

Parameters

Name	Type	Description
system	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

RemoteException

ProcessingException

UserException

getSystemRecordsByEUID

Description

getSystemRecordsByEUID returns the active system objects associated with the specified EUID.

Syntax

```
System<ObjectName>[] getSystemRecordsByEUID(String euid)
```

Parameters

Name	Type	Description
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

RemoteException

ProcessingException

UserException

getSystemRecordsByEUIDStatus

Description

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

Syntax

```
System<ObjectName>[] getSystemRecordsByEUIDStatus(String euid, String
status)
```

Parameters

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

Returns

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

Throws

RemoteException

ProcessingException

UserException

lookupLIDs

Description

lookupLIDs 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 that both systems must be of the specified status or an error will occur.

Syntax

```
System<ObjectName>PK[] lookupLIDs(String sourceSystem, String
sourceLID, String destSystem, String status)
```

Parameters

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

Returns

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

Throws

RemoteException

ProcessingException

UserException

mergeEnterpriseRecord

Description

mergeEnterpriseRecord merges two enterprise objects, specified by their EUIDs.

Syntax

```
Merge<ObjectName>Result mergeEnterpriseRecord(String fromEUID, String
toEUID, boolean calculateOnly)
```

Parameters

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

Returns

A merge result object containing the results of the merge.

Throws

RemoteException

ProcessingException

UserException

mergeSystemRecord

Description

mergeSystemRecord 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

```
Merge<ObjectName>Result mergeSystemRecord(String sourceSystem, String
sourceLID, String destLID, boolean calculateOnly)
```

Parameters

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

Returns

A merge result object containing the results of the merge.

Throws

RemoteException

ProcessingException

UserException

searchBlock

Description

searchBlock performs a blocking query against the database using the blocking query specified in the Threshold file and the criteria contained in the specified object bean.

Syntax

```
Search<ObjectName>Result searchBlock(<ObjectName>Bean searchCriteria)
```

Parameters

Name	Type	Description
searchCriteria	<ObjectName>Bean	The search criteria for the blocking query.

Returns

The results of the search.

Throws

RemoteException

ProcessingException

UserException

searchExact

Description

searchExact 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

```
Search<ObjectName>Result searchExact(<ObjectName>Bean searchCriteria)
```

Parameters

Name	Type	Description
searchCriteria	<ObjectName>Bean	The search criteria for the exact match search.

Returns

The results of the search stored in a Search<ObjectName>Result object.

Throws

RemoteException

ProcessingException

UserException

searchPhonetic

Description

searchPhonetic 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

```
Search<ObjectName>Result searchPhonetic(<ObjectName>Bean
searchCriteria)
```

Parameters

Name	Type	Description
searchCriteria	<ObjectName>Bean	The search criteria for the phonetic search.

Returns

The results of the search.

Throws

RemoteException

ProcessingException

UserException

transferSystemRecord

Description

transferSystemRecord transfers a system record from one enterprise record to another enterprise record.

Syntax

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

Parameters

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

Returns

None.

Throws

RemoteException

ProcessingException

UserException

updateEnterpriseRecord

Description

updateEnterpriseRecord 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

```
void updateEnterpriseRecord(Enterprise<ObjectName> enterpriseObject)
```

Parameters

Name	Type	Description
enterpriseObject	Enterprise<ObjectName>	The enterprise object containing the values that will update the existing enterprise object.

Returns

None.

Throws

RemoteException

ProcessingException

UserException

updateSystemRecord

Description

updateSystemRecord updates the existing system object in the database with the given system object.

Syntax

```
void updateSystemRecord(System<ObjectName> systemObject)
```

Parameters

Name	Type	Description
systemObject	System<ObjectName>	The system object to be updated to the enterprise object. **Note:** *In the method OTD, "Object" in the parameter name is changed to the name of the parent object. For example, if the parent object is "Person", the name of this parameter will appear as "systemPerson".*

Returns

None.

Throws

RemoteException

ProcessingException

UserException

4.5 Dynamic Business Process Methods

A set of Java methods are included in the eIndex SPV Project for use in eInsight Business Processes. These methods include a subset of the dynamic OTD methods, which are documented above. Many of these methods return objects of the dynamic object type, or they use these objects as parameters. In the descriptions for these methods, <ObjectName> indicates the name of the parent object.

The following methods are available for Business Processes. They are described in the previous section, **"Dynamic OTD Methods"**.

- **executeMatch** on page 88
- **executeMatchUpdate** on page 89
- **getEnterpriseRecordByEUID** on page 90
- **getEnterpriseRecordByLID** on page 91
- **getEUID** on page 91
- **getLIDs** on page 92
- **getLIDsByStatus** on page 92
- **getSBR** on page 93
- **getSystemRecordsByEUID** on page 94
- **getSystemRecordsByEUIDStatus** on page 95
- **lookupLIDs** on page 95
- **searchBlock** on page 97
- **searchExact** on page 98
- **searchPhonetic** on page 99

4.6 Helper Classes

Helper classes include objects that can be passed as parameters to an OTD method or a Business Process method. They also include the methods that you can access through the system<ObjectName> variable in the eIndex SPV Collaboration (where <ObjectName> is the name of a parent object. The helper classes include:

- **System<ObjectName>** on page 102
- **Parent Beans** on page 106
- **Child Beans** on page 113
- **DestinationEO** on page 117
- **Search<ObjectName>Result** on page 117
- **SourceEO** on page 119
- **System<ObjectName>PK** on page 119

4.6.1 System<ObjectName>

In order to run **executeMatch** in a Java Collaboration, you must define a variable of the class type System<ObjectName>, where <ObjectName> is the name of a parent object. This class is passed as a parameter to **executeMatch**. 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 **"ClearFieldIndicator Field" on page 103**). 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

```
class System<ObjectName>
```

Fields

- **ClearFieldIndicator Field** on page 103

Methods

- **System<ObjectName>** on page 103
- **getClearFieldIndicator** on page 103
- **get<Field>** on page 104
- **get<ObjectName>** on page 104
- **setClearFieldIndicator** on page 105
- **set<Field>** on page 105
- **set<ObjectName>** on page 106

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)

ClearFieldIndicator Field

The **ClearFieldIndicator** field 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 master index, 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 master index by entering an indicator in that field. This indicator is specified by the **ClearFieldIndicator** field. By default, the **ClearFieldIndicator** field is set to double-quotes (“”), so if a field is set to double-quotes, that field will be blanked out. If you do not want to use this feature, set the clear field indicator to null.

System<ObjectName>

Description

System<ObjectName> 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 System<ObjectName>()
```

Parameters

None.

Returns

An instance of the System<ObjectName> class.

Throws

ObjectException

getClearFieldIndicator

Description

getClearFieldIndicator retrieves the value of the **ClearFieldIndicator** field.

Syntax

```
Object getClearFieldIndicator()
```

Parameters

None.

Returns

An object containing the value of the **ClearFieldIndicator** field.

Throws

None.

get<Field>

Description

get<Field> retrieves the value of the specified system field. There are getter methods for the following fields: LocalId, SystemCode, Status, CreateDateTime, CreateFunction, and CreateUser.

Syntax

```
String get<Field>()
```

or

```
Date get<Field>()
```

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

get<ObjectName>

Description

get<ObjectName> retrieves the parent object Java Bean for the system record (where <ObjectName> is the name of the parent object).

Syntax

```
<ObjectName>Bean get<ObjectName>()
```

Parameters

None.

Returns

A Java Bean containing the parent object.

Throws

None.

setClearFieldIndicator

Description

setClearFieldIndicator sets the value of the clear field character (in the **ClearFieldIndicator** field). By default, this is set to double quotes (“”).

Syntax

```
void setClearFieldIndicator(String value)
```

Parameters

Name	Type	Description
value	String	The value that should be entered into a field to indicate that any existing values should be replaced with null.

Returns

None.

Throws

None.

set<Field>

Description

set<Field> sets the value of the specified system field. There are setter methods for the following fields: LocalId, SystemCode, Status, CreateDateTime, CreateFunction, and CreateUser.

Syntax

```
void set<Field>(value)
```

Parameters

Name	Type	Description
value	varies	The value to set in the specified field. The type of value depends on the field into which the value is being set.

Returns

None.

Throws

ObjectException

set<ObjectName>

Description

set<ObjectName> sets the parent object Java Bean for the system record (where <ObjectName> is the name of the parent object).

Syntax

```
void set<ObjectName>(<ObjectName>Bean object)
```

Parameters

Name	Type	Description
object	<ObjectName>Bean	The Java Bean for the parent object.

Returns

None.

Throws

ObjectException

4.6.2 Parent Beans

A Java Bean is created to represent each parent object defined in the object definition of the master index. The methods in these classes provide the ability to create a parent object Bean and to set or retrieve the field values for that object Bean.

The name of each parent object Bean class is the same as the name of each parent object, with the word “Bean” appended (by default, **PersonBean**). The methods in this class include a constructor method for the parent object Bean, and get and set methods for each field defined for the parent object. 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 <ObjectName>Bean
```

Methods

- **<ObjectName>Bean** on page 107
- **count<Child>** on page 107
- **countChildren** on page 108
- **countChildren** on page 108
- **delete<Child>** on page 109
- **get<Child>** on page 109
- **get<Child>** on page 110
- **get<Field>** on page 110
- **get<ObjectName>Id** on page 111
- **set<Child>** on page 111
- **set<Child>** on page 112
- **set<Field>** on page 112
- **set<ObjectName>Id** on page 113

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)

<ObjectName>Bean

Description

<ObjectName>Bean 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 <ObjectName>Bean()
```

Parameters

None.

Returns

An instance of the parent object Bean.

Throws

ObjectException

count<Child>

Description

count<Child> 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 Phone or Address).

Syntax

```
int count<Child>()
```

Parameters

None.

Returns

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

Throws

None.

countChildren

Description

countChildren 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

None.

countChildren

Description

countChildren 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

Name	Type	Description
type	String	The type of child object to count, such as Phone or Address.

Returns

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

Throws

None.

delete<Child>

Description

delete<Child> 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 index.

Syntax

```
void delete<Child>(String <Child>Id)
```

Parameters

Name	Type	Description
<Child>Id	String	The unique identification code of the child object to delete.

Returns

None.

Throws

ObjectException

get<Child>

Description

get<Child> retrieves an array of child object Beans. 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

```
<Child>Bean[] get<Child>()
```

Parameters

None.

Returns

An array of Java Beans containing the type of child objects specified by the method name.

Throws

None.

get<Child>

Description

get<Child> retrieves a child object Bean 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

```
<Child>Bean get<Child>(int i)
```

Parameters

Name	Type	Description
i	int	The index of the child object to retrieve from a list of child objects.

Returns

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

Throws

ObjectException

get<Field>

Description

get<Field> 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 get<Field>()
```

Note: *The syntax for the getter methods depends of the type of data specified for the field in the object structure. For example, the getter method for a date field would have the following syntax:* `Date get<Field>`*.*

Parameters

None.

Returns

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

Throws

ObjectException

get<ObjectName>Id

Description

get<ObjectName>Id retrieves the unique identification code (primary key) of the object, as assigned by the master index.

Syntax

```
String get<ObjectName>Id()
```

Parameters

None.

Returns

A string containing the unique ID of the parent object.

Throws

ObjectException

set<Child>

Description

set<Child> adds a child object to the system object.

Syntax

```
void set<Child>(int index, <Child>Bean child)
```

Parameters

Name	Type	Description
index	integer	The index number for the new child object.
child	<Child>Bean	The Java Bean containing the child object to add.

Returns

None.

Throws

None.

set<Child>

Description

set<Child> adds an array of child objects of one type to the system object.

Syntax

```
void set<Child>(<Child>Bean[] children)
```

Parameters

Name	Type	Description
children	<Child>Bean[]	The array of child objects to add.

Returns

None.

Throws

None.

set<Field>

Description

set<Field> 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

```
void set<Field>(value)
```

Parameters

Name	Type	Description
value	varies	The value of the field specified by the method name. The type of value depends on the field being populated.

Returns

None.

Throws

ObjectException

set<ObjectName>Id

Description

set<ObjectName>Id sets the value of the **<ObjectName>Id** field in the parent object.

Note: *This ID is set internally by the master index. Do not set this field manually.*

Syntax

```
void set<ObjectName>Id(String value)
```

Parameters

Name	Type	Description
value	String	The value of the **<ObjectName>Id** field.

Returns

None.

Throws

ObjectException

4.6.3 Child Beans

A Java Bean is created to represent each child object defined in the object definition of the master index. 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 Bean 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 Bean, 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

```
final class <Child>Bean
```

Methods

- **<Child>Bean** on page 114
- **delete** on page 114
- **get<Field>** on page 115
- **get<Child>Id** on page 115
- **set<Field>** on page 116
- **set<Child>Id** on page 116

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)

<Child>Bean

Description

<Child>Bean is the user-defined object Bean class. You can instantiate this class to create a new instance of the child object Bean class.

Syntax

```
new <Child>Bean()
```

Parameters

None.

Returns

An instance of the child object Bean.

Throws

ObjectException

delete

Description

delete removes the child object from the eIndex SPV object. This is used with the **executeMatchUpdate** function to update a system object by deleting one of the child objects from the eIndex SPV object.

Syntax

```
void delete()
```

Parameters

None.

Returns

None.

Throws

ObjectException

get<Field>

Description

get<Field> 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

```
String get<Field>()
```

Note: *The syntax for the getter methods depends of the type of data specified for the field in the object structure. For example, the getter method for a date field would have the following syntax:* `Date get<Field>`.

Parameters

None.

Returns

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

Throws

ObjectException

get<Child>Id

Description

get<Child>Id retrieves the unique identification code (primary key) of the object, as assigned by the master index.

Syntax

```
String get<Child>Id()
```

Parameters

None.

Returns

A string containing the unique ID of the child object.

Throws

ObjectException

set<Field>

Description

set<Field> 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 set<Field>(value)
```

Parameters

Name	Type	Description
value	varies	The value of the field specified by the method name. The type of value depends on the data type of the field being populated.

Returns

None.

Throws

ObjectException

set<Child>Id

Description

set<Child>Id sets the value of the **<Child>Id** field in the child object.

Note: *This ID is set internally by the master index. Do not set this field manually.*

Syntax

```
void set<Child>Id(String value)
```

Parameters

Name	Type	Description
value	String	The value of the **<Child>Id** field.

Returns

None.

Throws

ObjectException

4.6.4 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

- **getEnterprise<ObjectName>** on page 117

getEnterprise<ObjectName>

Description

getEnterprise<ObjectName> (where <ObjectName> is the name of the parent object) retrieves the surviving enterprise object from a merge transaction in order to allow the records to be unmerged.

Syntax

```
Enterprise<ObjectName> getEnterprise<ObjectName>()
```

where <ObjectName> is the name of the parent object.

Parameters

None.

Returns

The surviving enterprise object from a merge transaction.

Throws

ObjectException

4.6.5 Search<ObjectName>Result

This class represents the results of a search. A **Search<ObjectName>Result** object (where <ObjectName> is the name of the parent object) is returned as a result of a call to **"searchBlock"**, **"searchExact"**, or **"searchPhonetic"**.

Definition

```
class Search<ObjectName>Result
```

Methods

- **getEUID** on page 118
- **getComparisonScore** on page 118
- **get<ObjectName>** on page 118

getEUID

Description

getEUID retrieves the EUID of a search result record.

Syntax

```
String getEUID()
```

Parameters

None.

Returns

A string containing an EUID.

Throws

None.

getComparisonScore

Description

getComparisonScore 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.

get<ObjectName>

Description

get<ObjectName> retrieves an object bean for a search result record.

Syntax

```
<ObjectName>Bean get<ObjectName>()
```

where <ObjectName> is the name of the parent object.

Parameters

None.

Returns

An object bean.

Throws

None.

4.6.6 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

- **getEnterprise<ObjectName>** on page 119

getEnterprise<ObjectName>

Description

getEnterprise<ObjectName> (where <ObjectName> is the name of the parent object) retrieves the non-surviving enterprise object from a merge transaction in order to allow the records to be unmerged.

Syntax

```
Enterprise<ObjectName> getEnterprise<ObjectName>()
```

where <ObjectName> is the name of the parent object.

Parameters

None.

Returns

The non-surviving enterprise object from a merge transaction.

Throws

None.

4.6.7 System<ObjectName>PK

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

```
class System<ObjectName>PK
```

where <ObjectName> is the name of the parent object.

Methods

- **System<ObjectName>PK** on page 120
- **getLocalId** on page 120
- **getSystemCode** on page 121

System<ObjectName>PK

Description

System<ObjectName>PK 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

```
new System<ObjectName>PK()
```

where <ObjectName> is the name of the parent object.

Parameters

None.

Returns

An instance of the system primary key object.

Throws

None.

getLocalId

Description

getLocalID retrieves the local identifier from a system primary key object.

Syntax

```
String getLocalId()
```

Parameters

None.

Returns

A string containing a local identifier.

Throws

None.

getSystemCode

Description

getSystemCode retrieves the system's processing code from a system primary key object.

Syntax

```
String getSystemCode()
```

Parameters

None.

Returns

A string containing the processing code for a system.

Throws

None.

Appendix A

Inbound Message Processing with Custom Logic

You can customize the way the execute match methods process inbound messages by defining custom plug-ins that include decision-point methods. This appendix describes the standard inbound processing logic as described in **"Inbound Message Processing Logic" on page 26**, but also includes how the decision-point methods alter the process.

What's in This Chapter

A.1 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 26**, but include descriptions of the decision points, which are listed in italic font. If no custom logic is defined, the decision points default to "false", and processing is identical to that described in **"Inbound Message Processing Logic"**.

For more information about the methods and plug-ins, see "Customizing Match Processing Logic" in the *Sun SeeBeyond eIndex Single Patient View User's Guide*. For detailed information about the methods, see the Javadocs provided with eIndex SPV. The methods are contained in the **ExecuteMatchLogics** class in the package **com.stc.eindex.master**.

1 When a message is received by the master index, 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 patient. Using the EUID of the existing record, the master index 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 the update does not make any changes to the patient's information, no further processing is required and the existing EUID is returned.
- If there are changes to the patient'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 re-evaluated 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 master index 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 master index 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 patient. Using the EUID of the existing record, the master index 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 patient and an assumed match occurs. Using the EUID of the existing record, the master index 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 the *Sun SeeBeyond eIndex Single Patient View Configuration Guide*).

Glossary

alphanumeric search

A type of search that looks for records that precisely match the specified criteria. This type of search does not allow for misspellings or data entry errors, but does allow the use of wildcard characters.

assumed match

When the matching weight between two records is at or above a weight you specify and the records are from two different systems, (depending on the configuration of matching parameters) the objects are considered an assumed match and are automatically combined.

Blocking Query

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

Candidate Select file

The eIndex SPV configuration file that defines the queries you can perform from the Enterprise Data Manager (EDM) and the queries that are performed for matching.

candidate selection

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

candidate selection pool

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

checksum

A value added to the end of an EUID for validation purposes. The checksum for each EUID is derived from a specific mathematical formula.

code list

A list of values in the sbyn_common_detail database table that is used to populate values in the drop-down lists of the EDM.

code list type

A category of code list values, such as states or country codes. These are defined in the sbyn_common_header database table.

duplicate threshold

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

EDM

See *Enterprise Data Manager*.

Enterprise Data Manager

Also known as the EDM, this is the web-based interface that allows monitoring and manual control of the master index database. The configuration of the EDM is stored in the Enterprise Data Manager file in the eIndex Project.

enterprise object

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

ePath

A definition of the location of a field in an eIndex SPV object. Also known as the *element path*.

EUID

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

eIndex SPV Manager Service

An eIndex SPV component that provides an interface to all eIndex SPV components and includes the primary functions of eIndex. This component is configured by the Threshold file.

field IDs

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

Field Validator

An eIndex SPV component that specifies the Java classes containing field validation logic for incoming data. This component is configured by the Field Validation file.

Field Validation file

The eIndex SPV configuration file that specifies any default or custom Java classes that perform field validations when data is processed.

LID

See *local ID*.

local ID

A unique identification code assigned to an member in a specific local system. A member profile may have several local IDs in different systems. The combination of a local ID and system constitutes a unique identifier for a system record. The name of the local ID field is configurable on the EDM, and might have been modified for your implementation.

master person index
A database application that stores and cross-references information about the members in a business organization, regardless of the computer system from which the information originates.

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

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

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

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

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

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

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

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

member
Any person who participates within your business enterprise. A member could be a customer, employee, patient, and so on.

member profile

A set of information that describes characteristics of one member. A profile includes demographic and identification information about a member and contains a single best record and one or more system records.

merge

To join two member profiles or system records that represent the same person into one member profile.

merged profile

See *non-surviving profile*.

non-surviving profile

A member profile that is no longer active because it has been merged into another member profile. Also called a *merged profile*.

normalization

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

object

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

parsing

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

phonetic encoding

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

phonetic search

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

potential duplicates

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

probabilistic weighting

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

probability weight

See *matching probability weight*.

Query Builder

An eIndex SPV component that defines how queries are processed. The user-configured logic for this component is contained in the Candidate Select file.

SBR

See *single best record*.

single best record

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

standardization

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

survivor calculator

The logic that determines which field values or child objects from the available source systems are used to populate the SBR. This logic is a combination of Java classes and user-configured logic contained in the Best Record file.

survivorship

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

system

A computer application within your company where information is entered about the members in eIndex and that shares this information with eIndex (such as a registration system). Also known as a source system, local system, or external system.

system object

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

tab

A heading on an application window that, when clicked, displays a different type of information. For example, click the Create System Record tab to display the Create System Record page.

Threshold file

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

transaction history

A stored history of an enterprise object. This history displays changes made to the object's information as well as merges, unmerges, and so on.

Update Manager

The component of the master index that contains the Java classes and logic that determines how records are updated and how the SBR is populated. The user-configured logic for this component is contained in the Best Record file.

Index