SeeBeyond ICAN Suite

eIndex Single Patient Identifier Composite Application Reference Guide

Release 5.0.5

SEEBEYOND®

SeeBeyond Proprietary and Confidential

The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

SeeBeyond, e*Gate, e*Way, and e*Xchange are the registered trademarks of SeeBeyond Technology Corporation in the United States and/or select foreign countries. The SeeBeyond logo, SeeBeyond Integrated Composite Application Network Suite, eGate, eWay, eInsight, eVision, eXchange, eView, eIndex, eTL, ePortal, eBAM, and e*Insight are trademarks of SeeBeyond Technology Corporation. The absence of a trademark from this list does not constitute a waiver of SeeBeyond Technology Corporation's intellectual property rights concerning that trademark. This document may contain references to other company, brand, and product names. These company, brand, and product names are used herein for identification purposes only and may be the trademarks of their respective owners.

© 2004 by SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20041213151251.

Contents

List of Tables 8

Chapter 1

Introduction 9

About eIndex Single Patient Identifier Composite Application 9
- Overview 9
- Features and Functions 10

What's New in This Release 10

About This Document 11
- What's in This Document 11
- Scope 11
- Intended Audience 12
- Document Conventions 12
- Screenshots 12

Related Documents 12

SeeBeyond Web Site 13

SeeBeyond Documentation Feedback 13

Chapter 2

eIndex Overview 14

eIndex Repository Components 14
- Editors 14
- Project Components 15
  - Configuration Files 16
  - Database Scripts 17
  - Custom Plug-ins 17
  - Match Engine Configuration Files 18
  - Outbound Object Type Definition (OTD) 18
  - Dynamic Java API 18
  - Connectivity Components 20
  - Deployment Profile 21
- Environment Components 21

The Runtime Environment 21
- Functions of the Runtime Environment 21
- Runtime Environment Components 22

Matching Service 23
eIndex Manager Service 23
Query Builder 23
Query Manager 24
Update Manager 24
Object Persistence Service (OPS) 24
Database 24
Enterprise Data Manager 24

Chapter 3

Understanding Operational Processes 25

Learning About Message Processing 25

Inbound Message Processing 26
About Inbound Messages 27
The Default Inbound OTD 28
Formatting Guidelines 28
Transaction Details 28

Outbound Message Processing 34
About Outbound Messages 34
Outbound OTD Structure 35
Outbound Message Trigger Events 36
Sample Outbound Message 36

Inbound Message Processing Logic 37

Chapter 4

The Database Structure 43

Overview of the eIndex Database 43

eIndex Database Description 43
Database Table Overview 44
Database Table Details 45
SBYN_<*OBJECT_NAME*> *45*
SBYN_<*OBJECT_NAME*>SBR 46
SBYN_<*CHILD_OBJECT*> and SBYN_<*CHILD_OBJECT*>SBR 46
SBYN_APPL 47
SBYN_ASSUMEDMATCH 47
SBYN_AUDIT 48
SBYN_COMMON_DETAIL 49
SBYN_COMMON_HEADER 49
SBYN_ENTERPRISE 50
SBYN_MERGE 51
SBYN_OVERWRITE 51
SBYN_POTENTIALDUPLICATES 52
SBYN_SEQ_TABLE 53
SBYN_SYSTEMOBJECT 54
SBYN_SYSTEMS 55

SBYN_SYSTEMSBR 56
SBYN_TRANSACTION 57
SBYN_USER_CODE 58

Sample Database Model 59

Chapter 5

Working with the Java API 64

Overview 64
Java Class Types 64
Static Classes 65
Dynamic Object Classes 65
Dynamic OTD Methods 65
Dynamic eInsight Integration Methods 65

Dynamic Object Classes 65
Parent Object Classes 65
<*ObjectName*>Object 66
add<*Child*> *67*
addSecondaryObject 67
copy 68
dropSecondaryObject 68
get<*ObjectName*>Id 69
get<*Field*> *69*
get<*Child*> *70*
getChildTags 70
getMetaData 70
getSecondaryObject 71
getStatus 71
set<*ObjectName*>Id 72
set<*Field*> *72*
setStatus 73
structCopy 73
Child Object Classes 73
<*Child*>Object 74
copy 74
get<*Child*>Id 75
get<*Field*> *75*
getMetaData 76
getParentTag 76
set<*Child*>Id 76
set<*Field*> *77*
structCopy 77

Dynamic OTD Methods 78
activateEnterpriseRecord 78
activateSystemRecord 79
addSystemRecord 79
deactivateEnterpriseRecord 80
deactivateSystemRecord 81
executeMatch 81
executeMatchUpdate 82
findMasterController 83
getEnterpriseRecordByEUID 83
getEnterpriseRecordByLID 84
getEUID 85
getLIDs 85
getLIDsByStatus 86

getSBR 86
getSystemRecord 87
getSystemRecordsByEUID 87
getSystemRecordsByEUIDStatus 88
lookupLIDs 89
mergeEnterpriseRecord 89
mergeSystemRecord 90
searchBlock 91
searchExact 91
searchPhonetic 92
transferSystemRecord 92
updateEnterpriseRecord 93
updateSystemRecord 94

Dynamic eInsight Integration Methods 94

Helper Classes 95

System<*ObjectName*> **95**
ClearFieldIndicator Field 96
System<*ObjectName*> *96*
getClearFieldIndicator 96
get<*Field*> 97
get<*ObjectName*> 97
setClearFieldIndicator 98
set<*Field*> 98
set<*ObjectName*> 99

Parent Beans **99**
<*ObjectName*>Bean 100
count<*Child*> 100
countChildren 101
countChildren 101
delete<*Child*> 102
get<*Child*> 102
get<*Child*> 103
get<*Field*> *103*
get<*ObjectName*>Id 104
set<*Child*> 104
set<*Child*> 105
set<*Field*> *105*
set<*ObjectName*>Id 106

Child Beans **106**
<*Child*>Bean 107
delete 107
get<*Field*> *108*
get<*Child*>Id 108
set<*Field*> *109*
set<*Child*>Id 109

DestinationEO **110**
getEnterprise<*ObjectName*> 110

Search<*ObjectName*>Result **110**
getEUID 111
getComparisonScore 111
get<*ObjectName*> 111

SourceEO **112**
getEnterprise<*ObjectName*> 112

System<*ObjectName*>PK **113**
System<*ObjectName*>PK 113
getLocalId 113
getSystemCode 114

Glossary 115

Index 120

List of Tables

Table 1	Document Conventions	12
Table 2	Default Inbound Message Structure - Transaction Information	28
Table 3	Default Inbound Message Structure - Record Information	29
Table 4	Outbound OTD SBR Node	35
Table 5	Master Index Database Tables	44
Table 6	SBYN_<*OBJECT_NAME*> Table Description	46
Table 7	SBYN_<*OBJECT_NAME*>SBR Table Description	46
Table 8	SBYN_<*CHILD_OBJECT*> and SBYN_<*CHILD_OBJECT*>SBR Table Description	46
Table 9	SBYN_APPL Table Description	47
Table 10	SBYN_ASSUMEDMATCH Table Description	47
Table 11	SBYN_AUDIT Table Description	48
Table 12	SBYN_COMMON_DETAIL Table Description	49
Table 13	SBYN_COMMON_HEADER Table Description	49
Table 14	SBYN_ENTERPRISE Table Description	50
Table 15	SBYN_MERGE Table Description	51
Table 16	SBYN_OVERWRITE Table Description	51
Table 17	SBYN_POTENTIALDUPLICATES Table Description	52
Table 18	SBYN_SEQ_TABLE Table Description	53
Table 19	Default Sequence Numbers	53
Table 20	SBYN_SYSTEMOBJECT Table Description	54
Table 21	SBYN_SYSTEMS Table Description	55
Table 22	SBYN_SYSTEMSBR Table Description	56
Table 23	SBYN_TRANSACTION Table Description	57
Table 24	SBYN_USER_CODE Table Description	58

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

- **About eIndex Single Patient Identifier Composite Application** on page 9
- **What's New in This Release** on page 10
- **About This Document** on page 11
- **Related Documents** on page 12
- **SeeBeyond Web Site** on page 13
- **SeeBeyond Documentation Feedback** on page 13

1.1 About eIndex Single Patient Identifier Composite Application

1.1.1 Overview

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

In eIndex, 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.

1.1.2 Features and Functions

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

- **Rapid Development**—eIndex 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 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 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 implement a custom survivor strategy.
- **Flexible Architecture**—eIndex 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 provides support for both the SeeBeyond Match Engine and the Ascential™ INTEGRITY™ matching algorithm. In addition, you can plug in a custom matching algorithm.
- **Custom Java API**—eIndex 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 inbound to eIndex.

1.2 What's New in This Release

This release provides enhancements to the Enterprise Data Manager (EDM) and Query Manager. The following changes are included in this release.

- The ability to search by local ID and system was added to the History Search page of the EDM.
- The ability to search by local IDs on the Potential Duplicate Search page of the EDM was enhanced so existing local IDs are found (previously only the local IDs of the record that caused the potential duplicate were found).
- The local ID field, column, and screen area labels on the EDM are now configurable, and can be changed to a name more appropriate for your organization.
- The Query Manager was enhanced to allow custom queries to be written using combinations of nested AND and OR operations. Previously, only AND operations were supported.

1.3 About This Document

This guide provides comprehensive information about the database structure, the Java API, and message processing for the SeeBeyond® eIndex™ Single Patient Identifier Composite Application (eIndex). As a component of SeeBeyond's Integrated Composite Application Network (ICAN) Suite, eIndex 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 may differ from some of the descriptions contained in this guide. This guide is intended to be used with the *eIndex Single Patient Identifier Composite Application Configuration Guide* and the *eIndex Single Patient Identifier Composite Application User's Guide*.

1.3.1 What's in This Document

This guide is divided into five chapters 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 "eIndex Overview"** gives an overview of eIndex, and discusses the architecture, integration servers, and the eIndex Project.
- **Chapter 3 "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 4 "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 5 "Working with the Java API"** gives implementation information about the eIndex Java API, and provides a reference of the dynamic methods created for the method OTD and eInsight integration.

1.3.2 Scope

This guide provides information about message processing in eIndex and about the eIndex Java API. The API is designed to help you transform data and transfer the information into and out of the eIndex database using eGate Collaborations, Services, and eWays. This guide also provides an overview of the data processing flow, based on the 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). This guide compliments the *eIndex Single Patient Identifier Composite Application User's Guide*, the *eIndex Single Patient Identifier Composite Application Configuration Guide*, and the eIndex Javadocs. Once you understand the default processing, you can configure eIndex for your custom data and processing requirements.

This guide does not explain how to install eIndex, or how to implement an eIndex 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 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 eIndex, the operating system(s) on which eGate and the eIndex database run, and current business processes and information system (IS) setup.

1.3.4 Document Conventions

The following conventions are observed throughout this document.

Table 1 Document Conventions

Text	Convention	Example
Button, file, icon, parameter, variable, method, menu, and object names.	**Bold** text	▪ Click **OK** to save and close. ▪ From the **File** menu, select **Exit**. ▪ Select the **logicalhost.exe** file. ▪ Enter the **timeout** value. ▪ Use the **getClassName()** method. ▪ Configure the **Inbound** File eWay.
Command line arguments and code samples	`Fixed` font. Variables are shown in ***`bold italic`***.	`bootstrap -p` ***`password`***
Hypertext links	**Blue** text	**http://www.seebeyond.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.

1.4 Related Documents

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

- *eIndex Single Patient Identifier Composite Application User's Guide*
- *eIndex Single Patient Identifier Composite Application Configuration Guide*
- *Implementing the SeeBeyond Match Engine with eIndex*
- *Implementing Ascential INTEGRITY with eIndex*
- *eGate Integrator User's Guide*
- *eGate Integrator System Administration Guide*
- *SeeBeyond ICAN Suite Deployment Guide*

1.5 SeeBeyond Web Site

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

http://www.seebeyond.com

1.6 SeeBeyond Documentation Feedback

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

docfeedback@seebeyond.com

Chapter 2

eIndex Overview

eIndex is a highly configurable master person index that allows you to define the data structure and processing logic for the records stored in the eIndex database. This chapter provides overview information about eIndex and how the Project defines the master person index.

What's in This Chapter

- **eIndex Repository Components** on page 14
- **The Runtime Environment** on page 21

2.1 eIndex Repository Components

The Repository components of eIndex are designed to work within the eGate Enterprise Designer to create and configure eIndex, and to define connectivity between external systems and eIndex. This section describes the Repository components; the runtime components are described in **"Runtime Environment Components" on page 22**.

The primary Repository components of eIndex are:

- Editors
- Project Components
- Environment Components

2.1.1 Editors

eIndex provides the following editors to help you customize the files in the eIndex Project.

- **XML Editor**—allows you to review and customize the XML configuration files. This editor provides verification services for XML syntax (schema validation is provided through eIndex). The XML editor is automatically launched when you open an eIndex configuration file.
- **Text Editor**—allows you to review and customize the database scripts. This editor is very similar to the XML editor but without the verification services. The text editor is automatically launched when you open an eIndex database script.

- **Java Source Editor**—allows you to create and customize custom plug-in classes for eIndex. This editor is a simple text editor, similar to the Java Source Editor in the Java Collaboration Editor. The Java source editor is automatically launched when you open a custom plug-in file.

2.1.2 Project Components

eIndex is implemented within a Project in Enterprise Designer. The eIndex Project includes configuration files, database scripts, and custom plug-ins that you can customize. When you generate the Project, additional components are updated, including a method OTD, an outbound OTD, eInsight web page methods, and the necessary **.jar** files. To complete the Project, you create a Connectivity Map and Deployment Profile.

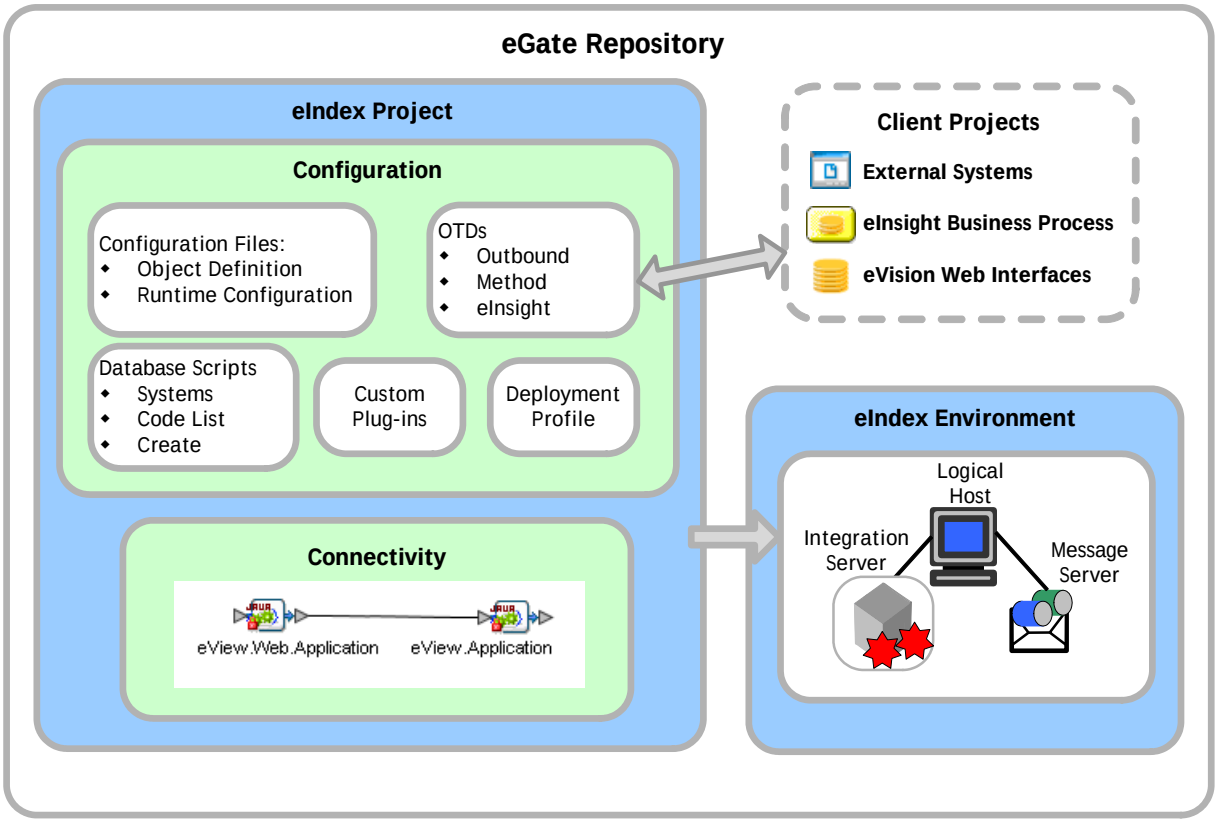
Additional eGate components must be added to the client Projects that share data with eIndex, including Services, Collaborations, OTDs, Web Connectors, eWays, JMS Queues, JMS Topics, and so on. You can use the standard Enterprise Designer editors, such as the OTD or Collaboration editors, to create these components.

Following is a list of eIndex Project components.

- Configuration Files
- Database Scripts
- Custom Plug-ins
- Match Engine Configuration Files
- Object Type Definitions
- Dynamic Java Methods
- Connectivity Components
- Deployment Profile

Figure 1 on page 16 illustrates the Project and Environment components of eIndex Studio.

Figure 1 eIndex Project and Environment Components



Configuration Files

Several XML files together determine certain characteristics of eIndex, such as how data is processed, queried, and matched. These files configure runtime components of eIndex, which are listed in **"Runtime Environment Components" on page 22**.

- **Object Definition**—Defines the data structure of eIndex.
- **Enterprise Data Manager**—Configures the search functions and appearance of the EDM, along with debug information and security information.
- **Candidate Select**—Configures the Query Builder component of eIndex, and defines the available queries.
- **Match Field**—Configures the Matching Service, and defines the fields to be standardized and the fields to use for matching. It also specifies the match and standardization engines to use.
- **Threshold**—Configures the eIndex Manager Service, and defines certain system parameters, such as match thresholds, EUID attributes, and update modes. It also specifies the query from the Query Builder to use for matching queries.
- **Best Record**—Configures the Update Manager and defines the strategies used by the survivor calculator to determine the field values for the SBR.

- **Field Validation**—Defines rules for validating field values. Rules are predefined for validating the local ID field, and you can create custom validation rules to plug in to this file.
- **Security—**This file is a placeholder to be used in future versions.

Database Scripts

Several database scripts are included in the eIndex Project to allow you to create the eIndex table structure, indexes, and custom start-up data. Several additional scripts are provided for testing purposes that drop the tables and indexes you created.

- **Systems**—Contains a sample SQL insert statement for adding information about an external system to the database. You can modify this script to define you own systems.
- **Code List**—Contains the SQL statements to insert processing codes and drop-down list values into the database. You might need to customize this file to match the processing codes used by your external systems.
- **Create Person database**—Defines the structure of the eIndex database based on the Object Definition file, and defines indexes against standard database tables. You can customize this script, and then run it against an Oracle database to create a customized database.
- **Create User Indexes**—Defines indexes against the fields that are defined for the blocking query in the Candidate Select file. You can define additional indexes if needed.
- **Create User Code Data**—Provides a sample "insert" script for adding data to the sbyn_user_code table.
- **Drop Person database**—Used primarily in testing, when you need to drop existing database tables and create new ones. The delete script removes all tables related to eIndex so you can recreate a fresh database for your Project.
- **Drop User Indexes**—Used primarily in testing, when you need to drop existing indexes, or for loading large batches of data, when indexes can slow down the process. This script removes all indexes defined in the **Create User Indexes** script.

You can also create custom scripts to store in the eIndex Project and run against the database.

Custom Plug-ins

eIndex provides a method by which you can create custom processing logic for the master index. To do this, you need to define and name a custom plug-in, which is a Java class that performs the required functions. Once you create a custom plug-in, you incorporate it into eIndex by adding it to the appropriate configuration file. You can create custom update procedures and field validations. Update procedures must be referenced in the update policies of the Best Record file, and field validations must be referenced in the Field Validation file. Custom plug-ins can also be used to create custom eIndex components, such as a custom query builder or block picker.

Match Engine Configuration Files

If you are using the SeeBeyond Match Engine, several configuration files for the engine are stored in the eIndex Project. The configuration files under the Match Engine node define certain weighting characteristics and constants for the match engine. The configuration files under the Standardization Engine node define how to standardize names, business names, and address fields. You can customize any of these fields as necessary. For more information, refer to *Implementing the SeeBeyond Match Engine with eIndex*.

Outbound Object Type Definition (OTD)

eIndex includes an outbound OTD based on the object structure defined in the Object Definition file. This OTD is used for distributing information that has been added or updated in eIndex to the external systems that share data with eIndex. It includes the objects and fields defined in the Object Definition file plus additional SBR information (such as the create date and create user) and additional system object information (such as the local ID and system code). If you plan to use this OTD to make eIndex data available to external systems, you must define a JMS Topic in the eIndex Connectivity Map to which eIndex can publish transactions.

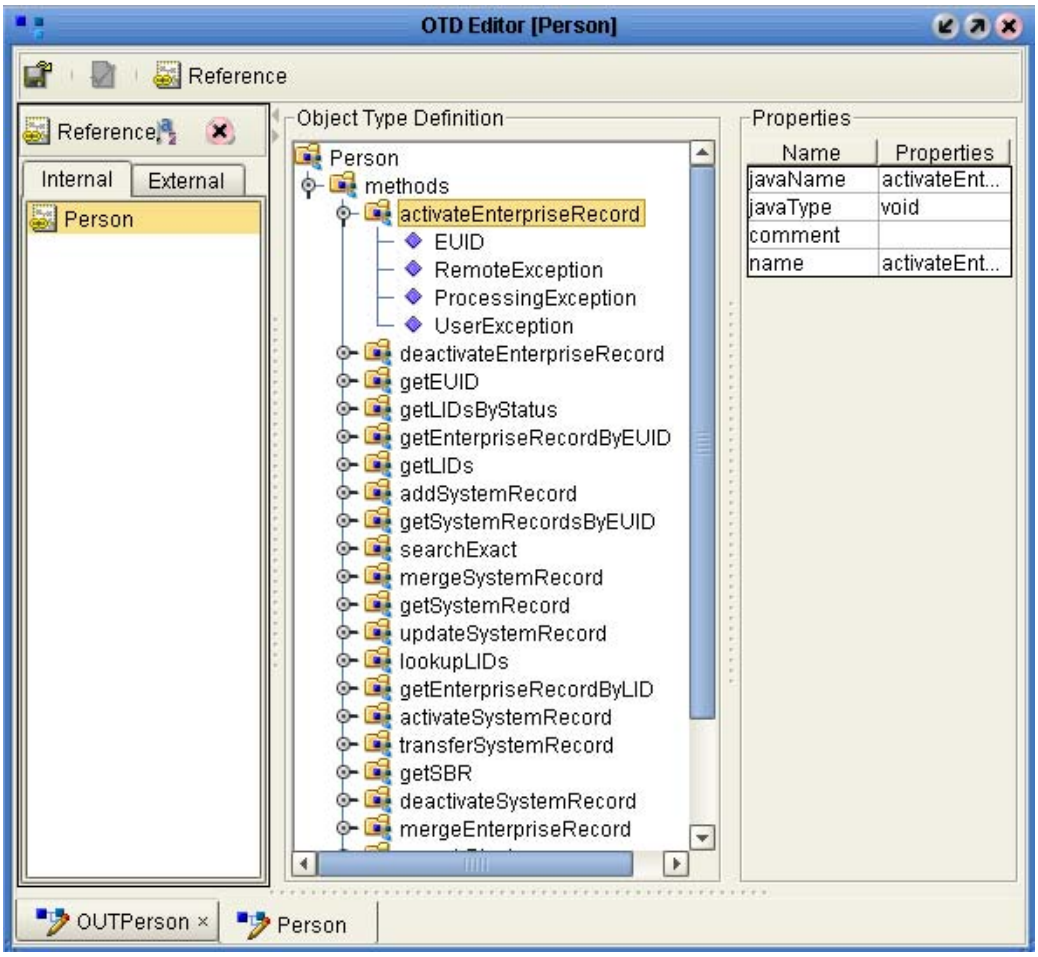
Dynamic Java API

Due to the flexibility of the object structure, eIndex includes several dynamic Java methods for use in Collaborations and in Web services. One set is provided in a method OTD for use in Collaborations and one set is provided for Web services. The names, parameter types, and return types of these methods vary based on whether you modify the object structure in the Object Definition file. These methods are described in **Chapter 5**, **"Working with the Java API"**.

Method OTD

The method OTD contains Java functions you can use to define data processing rules in Collaborations for external systems. These functions allow you to define how messages received from external systems are processed by the Service. You can define rules for inserting new records, retrieving record information, updating existing records, performing match processing on incoming records, and so on.

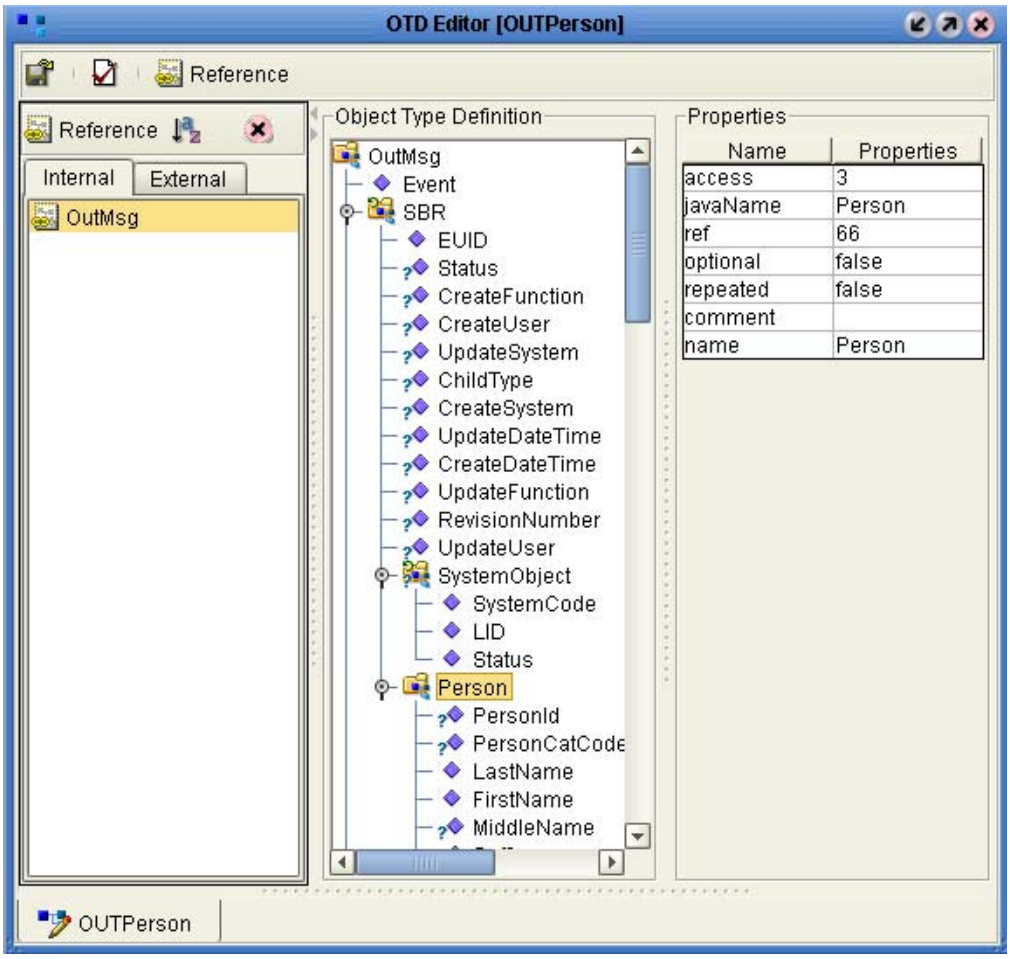
Figure 2 eIndex Method OTD



Web Services Java Methods

In addition to the method OTD, which can be used in Collaborations, eIndex includes a set of Java methods that can be incorporated into an eInsight Business Process for eVision Web services. These methods are a subset of those defined for the method OTD, providing the ability to view, retrieve, and match information in the eIndex database from eInsight Web pages.

Figure 3 Outbound OTD



Connectivity Components

The eIndex Project Connectivity Map only consists of two components: the Web application file and the application file. You can also include a JMS Topic for broadcasting messages or the Oracle eWay for database connectivity. In client Projects you can use any of the standard Project components to define connectivity and data flow for eIndex. Client Projects include those created for the external systems sharing data with the index and those created for eVision Web pages.

For the client Projects, you can use connectivity components from the eIndex server Project and create any standard eGate connectivity components, such as OTDs, Services, Collaborations, JMS Queues and Topics, and eWays. Client Project components transform and route incoming data into the eIndex database according to the rules contained in the Collaborations. They can also route the processed data back to the appropriate local systems through eWays.

Deployment Profile

The Deployment Profile defines information about the production environment of eIndex. It contains information about the assignment of Services and message destinations to integration servers and JMS IQ Managers within the eIndex system. Each eIndex Project must have at least one Deployment Profile, and can have several, depending on the Project requirements and the number of Environments used. You must activate the deployment before you can use the eIndex runtime environment.

2.1.3 Environment Components

The eIndex Environments define the configuration of the deployment environment of the runtime environment, including the Logical Host and application server. If eIndex client Projects use the same Environment, it may also include a JMS IQ Manager, constants, Web Connectors, and External Systems. Each Environment represents a unit of software that implements eIndex. You must define and configure at least one Environment for eIndex before you can deploy the application. The integration server hosting eIndex is configured within the Environment in the Enterprise Designer. Security is defined through the Environment configuration.

For more information about Environments, see the *eGate Integrator User's Guide*.

2.2 The Runtime Environment

The eIndex runtime environment includes the eIndex database, Enterprise Data Manager (EDM), and various services that process data. In the runtime environment, data is automatically processed from external systems using the configuration you set up using the Repository components. In addition, you can monitor and manage data manually using the EDM. This section describes the runtime environment features and components.

2.2.1 Functions of the Runtime Environment

eIndex in the runtime environment provides the following functions to help you monitor and maintain the data shared throughout the index system.

- **Transaction History**—The system provides a complete history of each member by recording all changes to each member's data. This history is maintained for both the local system records and the SBR.
- **Data Maintenance**—The web-based user interface supports all the necessary features for maintaining data records. It allows you to add new records; view, update, deactivate, or reactivate existing records; and compare records for similarities and differences. You can perform these functions against each local system record or SBR associated with a member record.
- **Search**—The information contained in each SBR or system record can be obtained from the database using a variety of search criteria. You can perform searches

against the database for a specific member or a set of members. For certain searches, the results are assigned a matching weight that indicates the probability of a match.

- **Potential Duplicate Detection and Handling**—One of the most important features of the eIndex system is its ability to match records and identify possible duplicates. Using matching algorithm logic, eIndex identifies potential duplicate records, and provides the functionality to correct the duplication. Potential duplicate records are easily corrected by either merging the records in question or marking the records as "resolved".
- **Merge and Unmerge**—You can compare potential duplicate records and then merge the records (at either the EUID or system-record level) if you find them to be actual duplicates of one another. At the EUID level, you can determine which record to retain as the active record. At the system level, you can determine which record to retain, and which information from each record to preserve in the resulting record.

2.2.2 Runtime Environment Components

The eIndex runtime environment is made up of several components that work together to form a complete indexing system. The primary components of the runtime environment are:

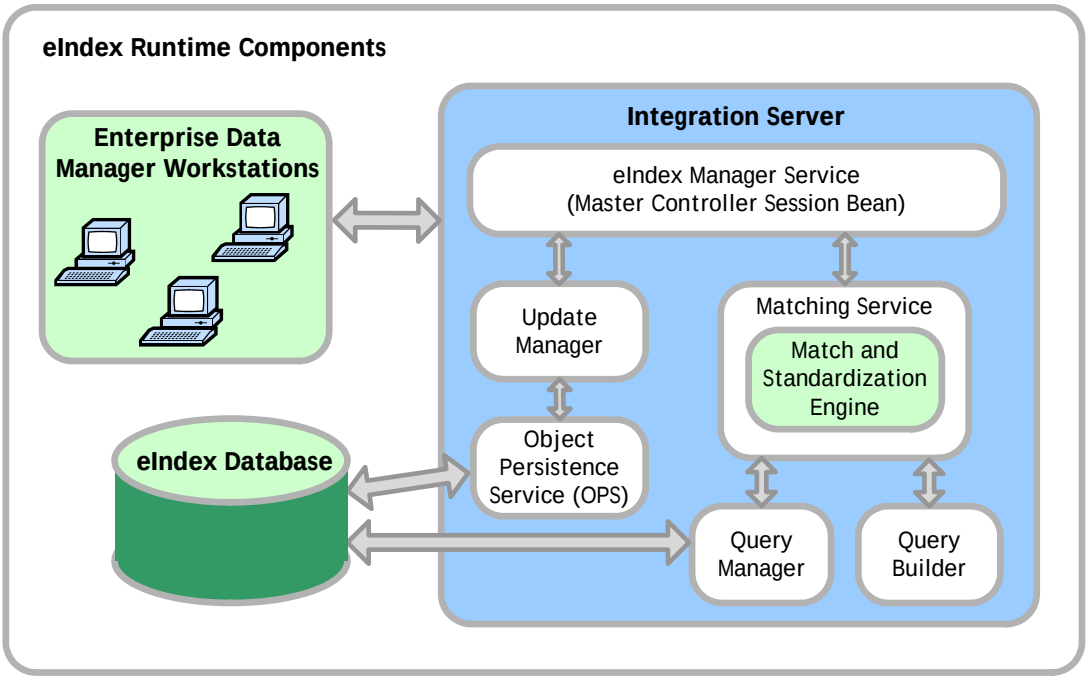
- eIndex Manager Service
- Matching Service
- Query Builder
- Query Manager
- Update Manager
- Object Persistence Service
- Database
- Enterprise Data Manager

In addition, eIndex uses the connectivity components defined in the eIndex server and client Projects to route data between external systems and the eIndex database.

The eGate Repository stores information about the configuration and structure of the runtime environment. Because eIndex is deployed within eGate, it can be implemented in a distributed environment. The eIndex system requires the SeeBeyond Integration Server to enable Web service connectivity.

The components of the eIndex runtime environment are illustrated in Figure 4.

Figure 4 eIndex Runtime Environment Architecture



2.2.3 Matching Service

The Matching Service stores the logic for standardization (which includes data parsing and normalization), phonetic encoding, and matching. It includes the specified standardization and match engines, along with the configuration you defined for each. The Matching Service also contains the data standardization tables and configuration files for the match engine, such as the configuration files for the SeeBeyond Match Engine or the rule set files for INTEGRITY. The configuration of the Matching Service is defined in the *Match Field* file.

2.2.4 eIndex Manager Service

The eIndex Manager Service provides a session bean to all components of the runtime environment, such as the Enterprise Data Manager, Query Builder, and Update Manager. The service also provides connectivity to the database. The configuration of the eIndex Manager Service specifies the query to use for matching, and defines system parameters that control EUID generation, matching thresholds, and update modes. The configuration of the eIndex Manager Service is defined in the *Threshold* file.

2.2.5 Query Builder

The Query Builder defines all queries available to eIndex. This includes the queries performed automatically by eIndex when searching for possible matches to an incoming record. It also includes the queries performed manually through the Enterprise Data Manager (EDM). The EDM queries can be either alphanumeric or

phonetic, and have the option of using wildcard characters. The configuration of the Query Builder is defined in the *Candidate Select* file.

2.2.6 Query Manager

The Query Manager is a service that performs queries against the eIndex database and returns a list of objects that match or closely match the query criteria. The Query Manager uses classes specified in the *Match Field* file to determine how to perform a query for match processing. All queries performed in the eIndex system are executed through the Query Manager.

2.2.7 Update Manager

The Update Manager controls how updates are made to a member's SBR by defining a survivor strategy for each field. The survivor calculator in the Update Manager uses these strategies to determine the relative reliability of the data from external systems and to determine which value for each field is populated into the SBR. The Update Manager also manages certain update policies, allowing you to define additional processing to be performed against incoming data. The configuration of the Update Manager is defined in the *Best Record* file.

2.2.8 Object Persistence Service (OPS)

OPS is a database service that translates high-level and descriptive object requests into actual JDBC calls. The service provides mapping from the Java object to the database and from the database to the Java object.

2.2.9 Database

eIndex uses an Oracle database to store the types of information you specify for member records. The database stores local system records, the SBR for each member record, and certain administrative information, such as drop-down menu lists, processing codes, and information about the systems from which data originates. The script used to create the eIndex database structure is based on the object structure defined in the Object Definition file.

2.2.10 Enterprise Data Manager

The Enterprise Data Manager (EDM) is a web-based interface that allows you to monitor and maintain the data in the eIndex database. The configurable attributes of the EDM are defined in the Enterprise Data Manager file, which you can modify after you generate the eIndex application. The EDM provides the ability to manually search for records; update, add, deactivate, and reactivate records; merge and unmerge records; view potential duplicates; and view comparisons of member records.

Chapter 3

Understanding Operational Processes

eIndex uses a custom Java API library and the eGate Integrator to transform and route data into and out of the eIndex database. In order to customize the way the Java methods transform the data, it is helpful to understand the logic of the primary processing function (executeMatch) and how messages are typically processed through the eIndex system.

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

What's in This Chapter

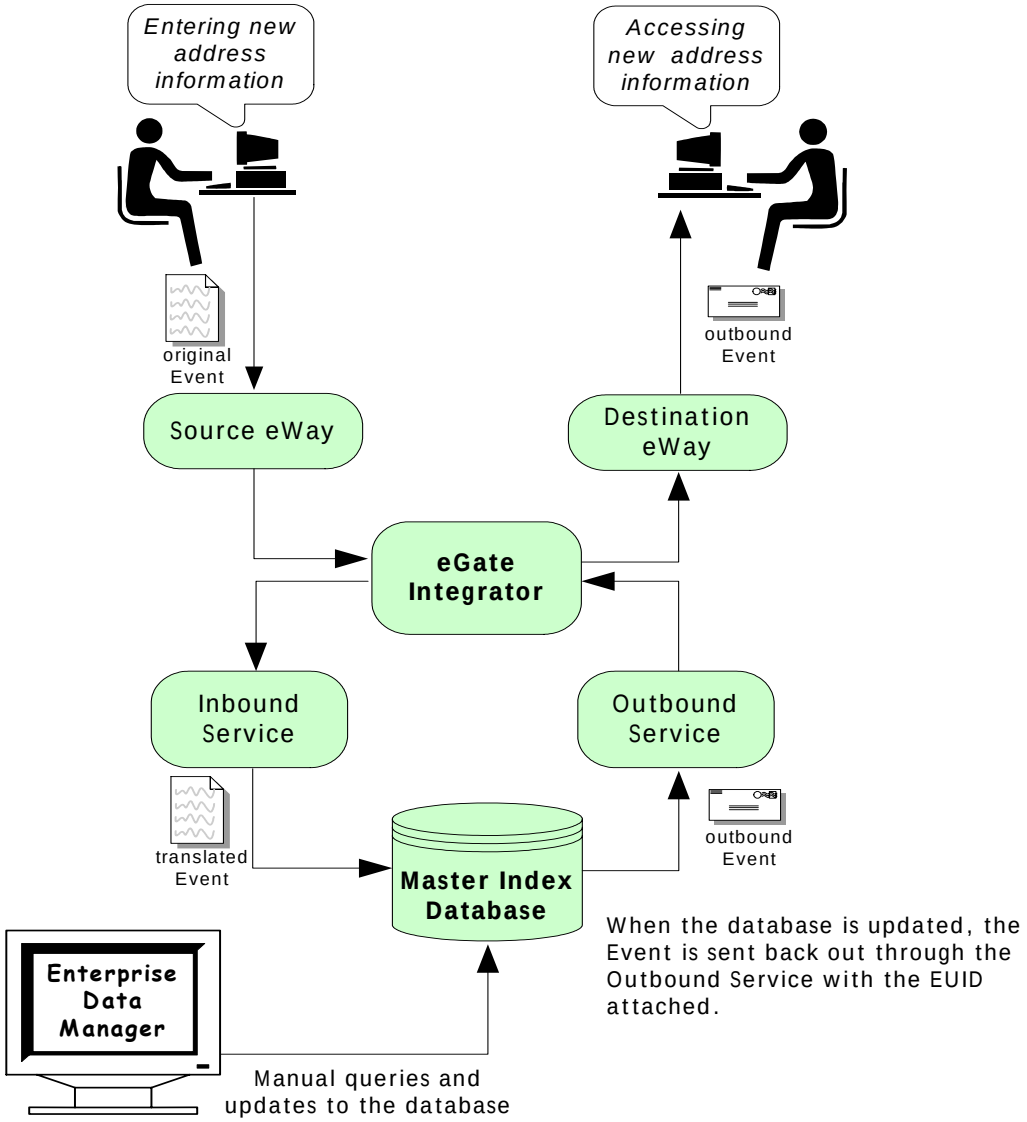
- **Learning About Message Processing** on page 25
- **Inbound Message Processing** on page 26
- **Outbound Message Processing** on page 34
- **Inbound Message Processing Logic** on page 37

3.1 Learning About Message Processing

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

Figure 5 on page 26 illustrates the flow of information through eIndex.

Figure 5 Master Index Processing Flow



3.2 Inbound Message Processing

An inbound message refers to the transmission of data from external systems to the eGate Integrator and then to the eIndex database. These messages may be sent into the database via a number of Services. Inbound messages are stored 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.

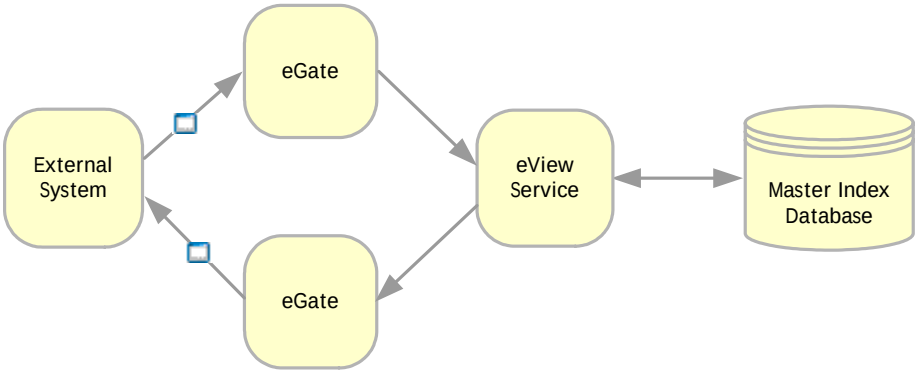
3 The message is modified into the appropriate format for the eIndex database, and validations are performed against the data elements of the message to ensure accurate delivery. The message is validated using the Java code in the Service's Collaboration and other information stored in the eIndex configuration files.

4 If the message was successfully transmitted to the database, the appropriate changes to the database are processed.

5 After eIndex processes the message, an 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 34** next).

Figure 6 below illustrates the flow of a message inbound to eIndex.

Figure 6 Inbound Message Processing Data Flow



3.2.1 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, or they can be mapped to the correct format in the Collaboration. The required format depends on how the object structure of eIndex is defined (in the Object Definition file of the eIndex Project).

In addition to the objects and fields defined in the Object Definition file, you can include standard eIndex 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 sample client Project includes system and local ID fields as well as transactional information fields. If incoming messages do not contain transactional information, eIndex 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 processes the transaction).

3.2.2 The Default Inbound OTD

This section describes the default format of the data to be inserted into the eIndex database. This format follows the format of the default object structure defined in the Object Definition file of the eIndex 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 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 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 Project.

Table 2 Default Inbound Message Structure - Transaction Information

Field	Description	Repeating?	Required?
SegmentId	"EVNT"	No	Yes
MessageId	Always leave this field blank. eIndex determines the message ID.	No	Yes
EventTypeCode	Always leave this field blank. eIndex 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 determines the EUID after it processes the message.	No	Yes
LocalId	The person'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)
SegmentId	**"DEMO"**	**No**	**Yes**

Table 3 Default Inbound Message Structure - Record Information

Field	Description	Repeating?	Required?
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 Event

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|1960-05-14|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~~~<>

3.3 Outbound Message Processing

An outbound message refers to the transmission of data from the eIndex database to any external system. Messages can be transmitted from eIndex 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 26**, and is only used for messages received from external systems.

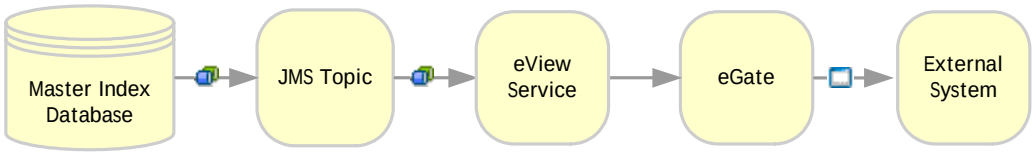
The second way is by publishing messages to a JMS Topic and publishing complete, updated records to any system subscribing to that topic. Outbound messages are generated in the format of the outbound OTD when updates are made to the database from either external systems or the Enterprise Data Manager. This section describes how the second type of outbound message is processed.

1. When a message is received and processed by eIndex, an XML message is generated and sent to a JMS Topic, which is configured to publish messages from eIndex.
2. Messages published by the JMS Topic are processed through a Service whose Collaboration uses the eIndex 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 log files.*

Figure 7 below illustrates the flow of data for a message outbound from eIndex.

Figure 7 Outbound Message Processing Data Flow



3.3.1 About Outbound Messages

When you customize the object definition and generate the eIndex 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 eIndex 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 eIndex through a JMS Topic using the outbound OTD.

Outbound OTD Structure

The outbound OTD is named after the application name of eIndex (for example, OUTCompany or OUTPerson). The outbound OTD for eIndex is named "OUTPerson". This OTD contains eight primary nodes: Event, ID, SBR, and the standard Java methods **unmarshalFromString**, **reset**, **marshalToString**, **marshal**, and **unmarshal**. 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 Node

Node	Descriptions
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 fields in this node contain local ID and system information.
SystemCode	The processing code of the system that created the new record or caused an existing record to be updated.
LID	The local ID associated with the above system for the published record.
Status	The status of the system record.
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 and contains all fields and child objects defined in the structure. This section varies depending on your customizations.

Outbound Message Trigger Events

When outbound messaging is enabled, the following transactions automatically generate an outbound message that is sent to the JMS Topic.

- 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 based on the default configuration. Your outbound messages may 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>
```

3.4 Inbound Message Processing Logic

When records are transmitted to eIndex, **executeMatch** is called and a series of processes are performed to ensure that accurate and current data is maintained in the database. 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 **executeMatch** are outlined below, and the diagrams on the following pages illustrate the message processing flow. The processing steps performed in your environment may vary from this depending on how you customize the Collaboration and Connectivity Map.

The following steps refer to the following parameters and element in the eIndex Threshold file (these are described in the *eIndex Single Patient Identifier Composite Application Configuration Guide*):

- OneExactMatch parameter
- SameSystemMatch parameter
- MatchThreshold parameter
- DuplicateThreshold parameter

- **update-mode** element

1 When a message is received by eIndex, 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 person. Using the EUID of the existing record, eIndex performs an update of the record's information in the database.

- If the update does not make any changes to the person's information, no further processing is required and the existing EUID is returned.
- If there are changes to the person'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, eIndex 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* ***eIndex Single Patient Identifier Composite Application Configuration Guide****.*

6 When records are checked for same system entries, eIndex 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 person. Using the EUID of the existing record, eIndex 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 person and an assumed match occurs. Using the EUID of the existing record, eIndex 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 *eIndex Single Patient Identifier Composite Application Configuration Guide*).

The flow charts on the following pages provide a visual representation of the processes performed by the default sample Project. Figures 8 and 9 represent the primary flow of information. Figure 10 expands on update procedures illustrated in Figures 8 and 9.

Figure 8 Inbound Message Processing in the Sample Project

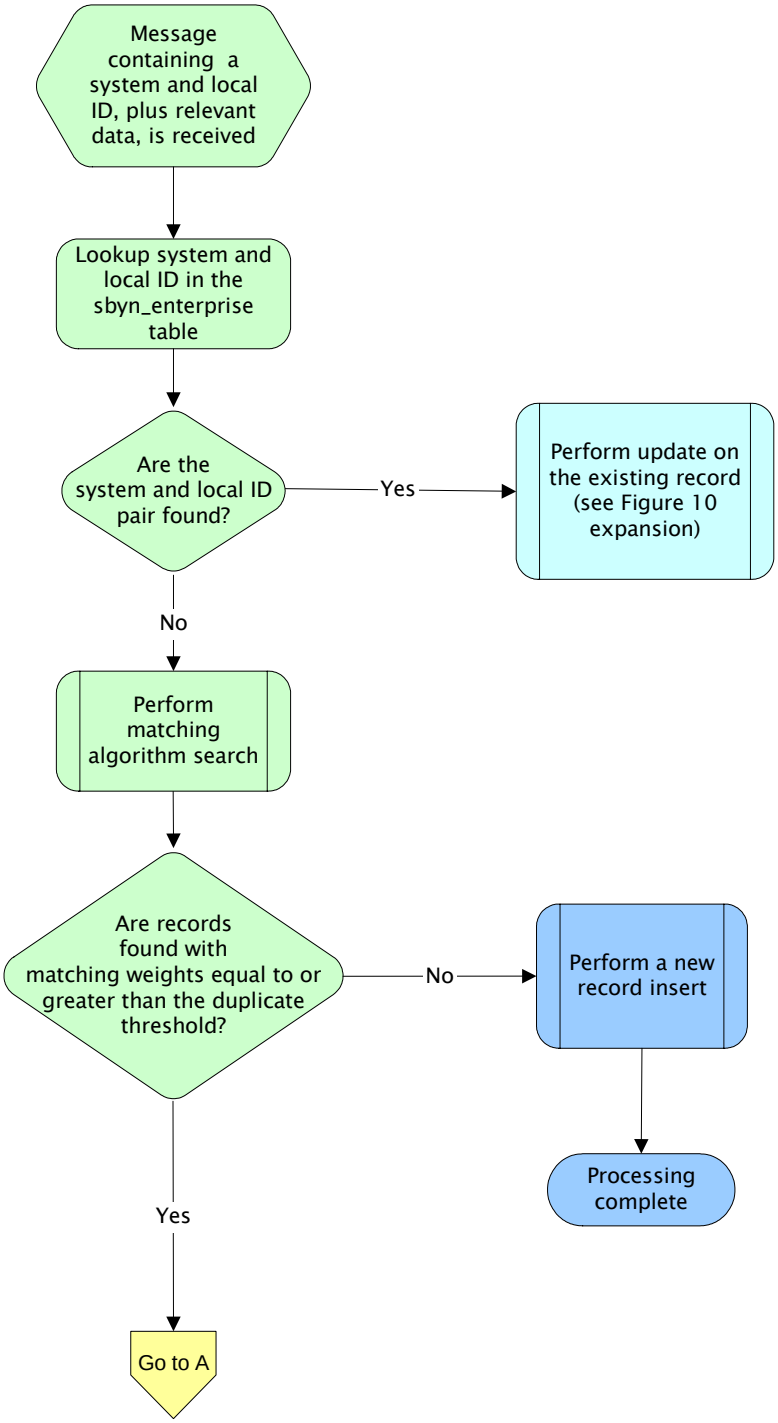


Figure 9 Inbound Message Processing (cont'd)

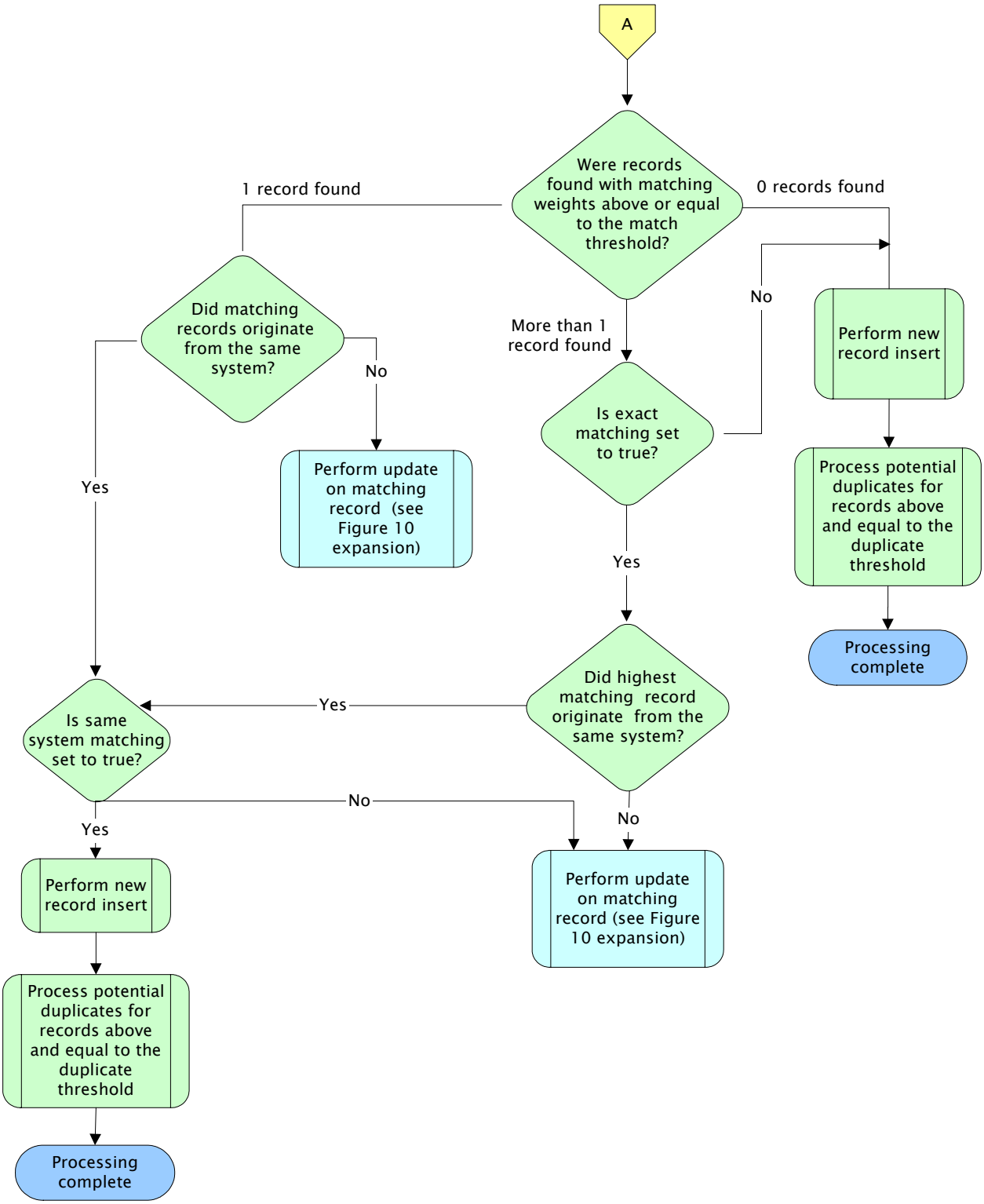
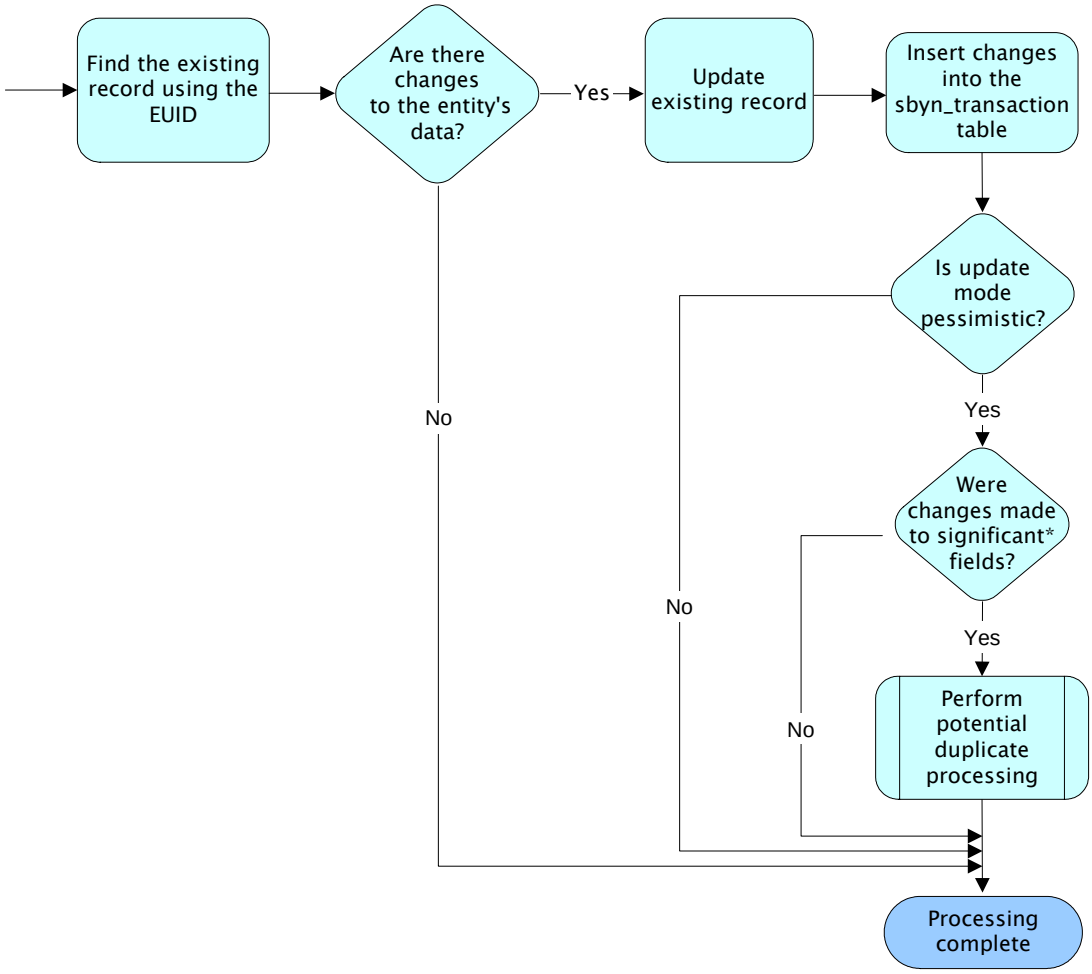


Figure 10 Record Update Expansion



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

Chapter 4

The Database Structure

This chapter provides information about the eIndex 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 may vary depending on the customizations made to the Object Definition and to the scripts used to create the eIndex database.

What's in This Chapter

- **Overview of the eIndex Database** on page 43
- **eIndex Database Description** on page 43
- **Sample Database Model** on page 59

4.1 Overview of the eIndex Database

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

The structure of the database tables that store person information is dependent on the information specified in the Object Definition file. eIndex includes a script to create the tables and fields in the eIndex 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.

4.2 eIndex Database Description

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. This section describes both types of tables and the fields contained in each table.

4.2.1 Database Table Overview

The eIndex database includes tables that store common maintenance information, transactional information, external system information, and information about the objects stored in the database. The database includes the tables listed in Table 5 on the following page.

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. There may be several tables storing child object information for system records.
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. There may be several tables storing child object information for SBRs.
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 merged by eIndex.
SBYN_AUDIT	Stores audit information about each time person information is accessed in the eIndex database.
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 eIndex. 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 eIndex 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).

4.2.2 Database Table Details

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

SBYN_<*OBJECT_NAME*>

This table stores the parent object in each system record received by eIndex. 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.

Table 6 SBYN_<*OBJECT_NAME*> Table Description

Column Name	Data Type	Column Description
SYSTEMCODE	VARCHAR2(20)	The system code for the system that produced the EUID record.
LID	VARCHAR2(25)	A local identification code assigned by the specified system.
<*OBJECT_NAME*>ID	Varies	A unique ID for the parent object in a system record. This is named according to the parent object. For example, "personid".

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.

Table 7 SBYN_<*OBJECT_NAME*>SBR Table Description

Column Name	Data Type	Column Description
EUID	VARCHAR2(20)	The enterprise unique identifier assigned by eIndex.
<*OBJECT_NAME*>ID	VARCHAR2(20)	A unique ID for the parent object in a system record. This is named according to the parent object. For example, "personid".

SBYN_<*CHILD_OBJECT*> and SBYN_<*CHILD_OBJECT*>SBR

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 eIndex. The sbyn_<*child_object*>sbr tables store information about the child objects associated with an SBR. All tables storing child object information 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 8 SBYN_<*CHILD_OBJECT*> and SBYN_<*CHILD_OBJECT*>SBR Table Description

Column Name	Data Type	Column Description
<*OBJECT_NAME*>ID	VARCHAR2(20)	The unique identification code for the parent object associated with the child object.

Table 8 SBYN_<*CHILD_OBJECT*> and SBYN_<*CHILD_OBJECT*>SBR Table Description

Column Name	Data Type	Column Description
<*CHILD_OBJECT*>ID	VARCHAR2(20)	The unique identification code for each record in the child object table. This column cannot be null.

SBYN_APPL

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

Table 9 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.

SBYN_ASSUMEDMATCH

This table maintains a record of each assumed match transaction that occurs in eIndex, 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 10 SBYN_ASSUMEDMATCH Table Description

Column Name	Data Type	Description
ASSUMEDMATCHID	VARCHAR2(20)	The unique ID for the assumed match transaction.
EUID	VARCHAR2(20)	The EUID into which the incoming record was merged.
SYSTEMCODE	VARCHAR2(20)	The processing code of the system from which the incoming record originated.
LID	VARCHAR2(25)	The local ID of the record in the source system.

Table 10 SBYN_ASSUMEDMATCH Table Description

Column Name	Data Type	Description
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 transaction.

SBYN_AUDIT

This table maintains a log of each instance in which any of the eIndex tables are accessed in the eIndex 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.

Table 11 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.
CREATE_BY	VARCHAR2(20)	The user ID of the person who performed the transaction that caused the audit log. This column cannot be null.

SBYN_COMMON_DETAIL

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

Table 12 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.

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 13 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.

Table 13 SBYN_COMMON_HEADER Table Description

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

SBYN_ENTERPRISE

This table stores a list of all the system and local ID pairs assigned to the person records in the eIndex 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 14 SBYN_ENTERPRISE Table Description

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

SBYN_MERGE

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

Table 15 SBYN_MERGE Table Description

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

SBYN_OVERWRITE

This table stores information about the fields that are locked 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 16 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 16 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 integer field.
TIMESTAMPDATA	DATE	The data that is locked for overwrite in a timestamp field.

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 17 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.
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 17 SBYN_POTENTIALDUPLICATES Table Description

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

SBYN_SEQ_TABLE

This table controls and maintains a record of the sequential identification numbers used in various tables in the database, ensuring that each number is unique and assigned in order. Several of the ID numbers maintained in this table are determined by the object structure. The numbers are assigned sequentially, but are allocated 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 18 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 19.

Table 19 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 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 eIndex. 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 19 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.

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 20 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 20 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

SBYN_SYSTEMS

This table stores information about each system integrated into the eIndex 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 21 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.
STATUS	CHAR(1)	The status of the system in eIndex. "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.
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 Java 2Software Development Kit (SDK).

Table 21 SBYN_SYSTEMS Table Description

Column Name	Data Type	Description
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.

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 22 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.

Table 22 SBYN_SYSTEMSBR Table Description

Column Name	Data Type	Description
CHILDTYPE	VARCHAR2(20)	The type of object being processed (currently only the name of the parent object). This column is reserved for future use.
CREATESYSTEM	VARCHAR2(20)	The system in which the system record was created.
CREATEUSER	VARCHAR2(30)	The user ID of the person who created the system record.
CREATEFUNCTION	VARCHAR2(20)	The type of transaction that created the system record.
CREATEDATE	DATE	The date the system object was created.
UPDATEUSER	VARCHAR2(30)	The user ID of the person who last updated the system record.
UPDATEFUNCTION	VARCHAR2(20)	The type of transaction that last updated the system record.
UPDATEDATE	DATE	The date the system object was last updated.
STATUS	VARCHAR2(15)	The status of the enterprise record. The status can be one of these values: ▪ active ▪ inactive ▪ merged
REVISIONNUMBER	NUMBER(38)	The revision number of the SBR. This is used for version control.

SBYN_TRANSACTION

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

Table 23 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.

Table 23 SBYN_TRANSACTION Table Description

Column Name	Data Type	Description
EUID2	VARCHAR2(20)	The EUID of the second person 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	DATE	The date and time the transaction occurred.
DELTA	BLOB	A list of the changes that occurred to system records as a result of the transaction.
SYSTEMCODE	VARCHAR2(20)	The processing code of the source system in which the transaction originated.
LID	VARCHAR2(25)	The local ID of the system record involved in the transaction.
EUID	VARCHAR2(20)	The EUID of the enterprise record involved in the transaction.

SBYN_USER_CODE

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

Table 24 SBYN_USER_CODE Table Description

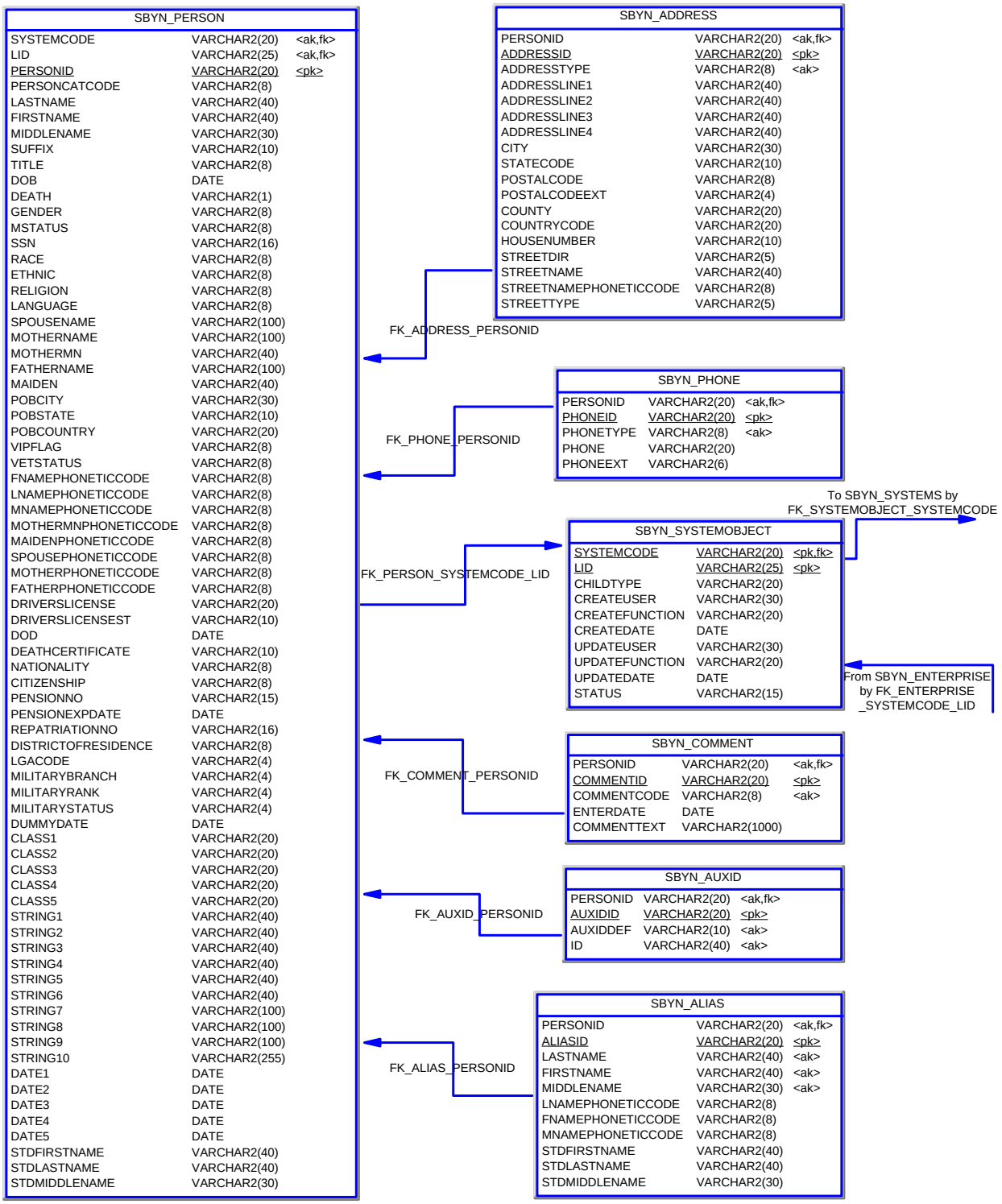
Column Name	Data Type	Description
CODE_LIST	VARCHAR2(20)	The code list name of the user code type (using the credit card example above, this might be similar to "CREDCARD"). This column links the values for each list.
CODE	VARCHAR2(20)	The processing code of each user code element.
DESCRIPTION	VARCHAR2(50)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

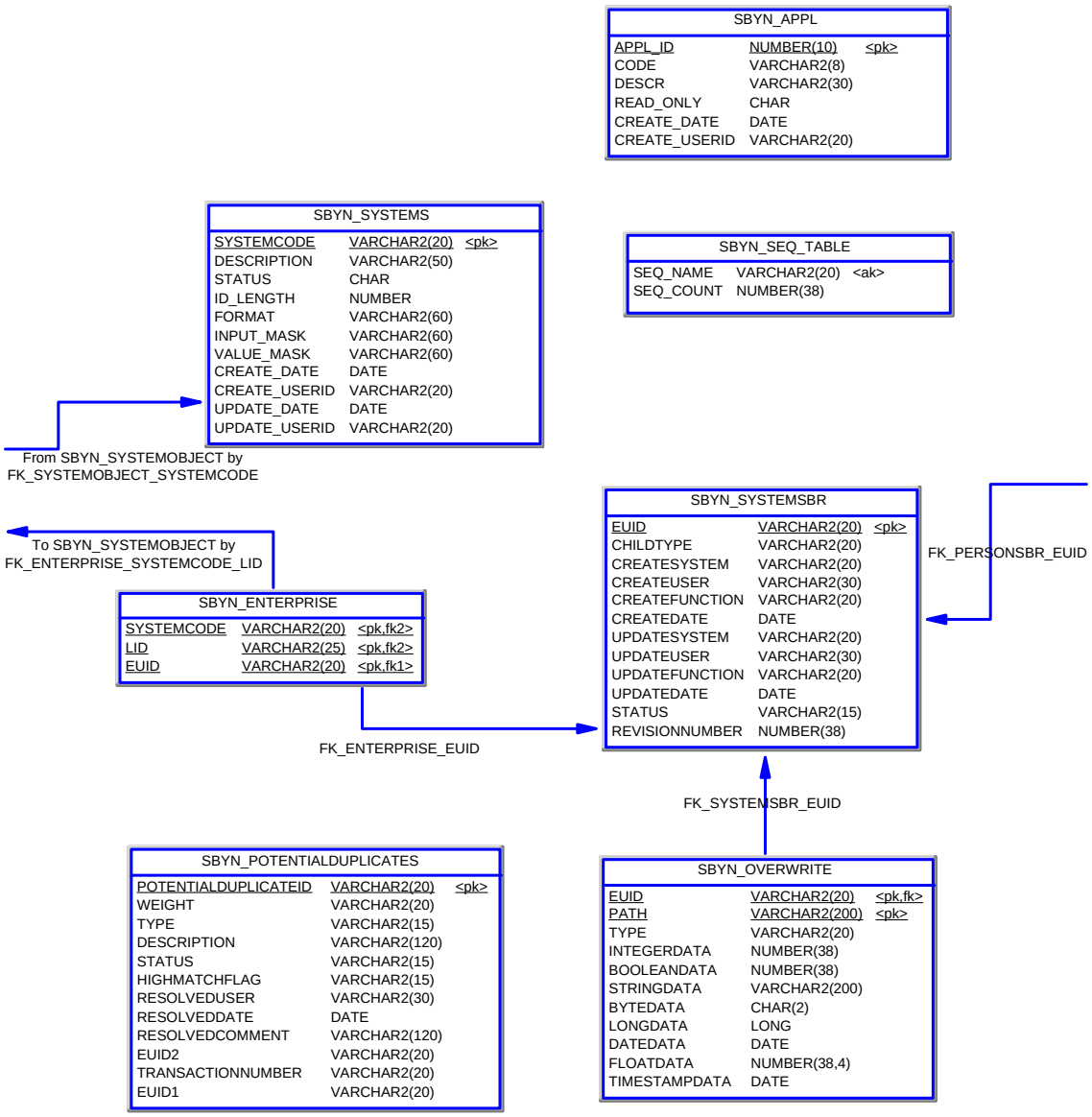
Table 24 SBYN_USER_CODE Table Description

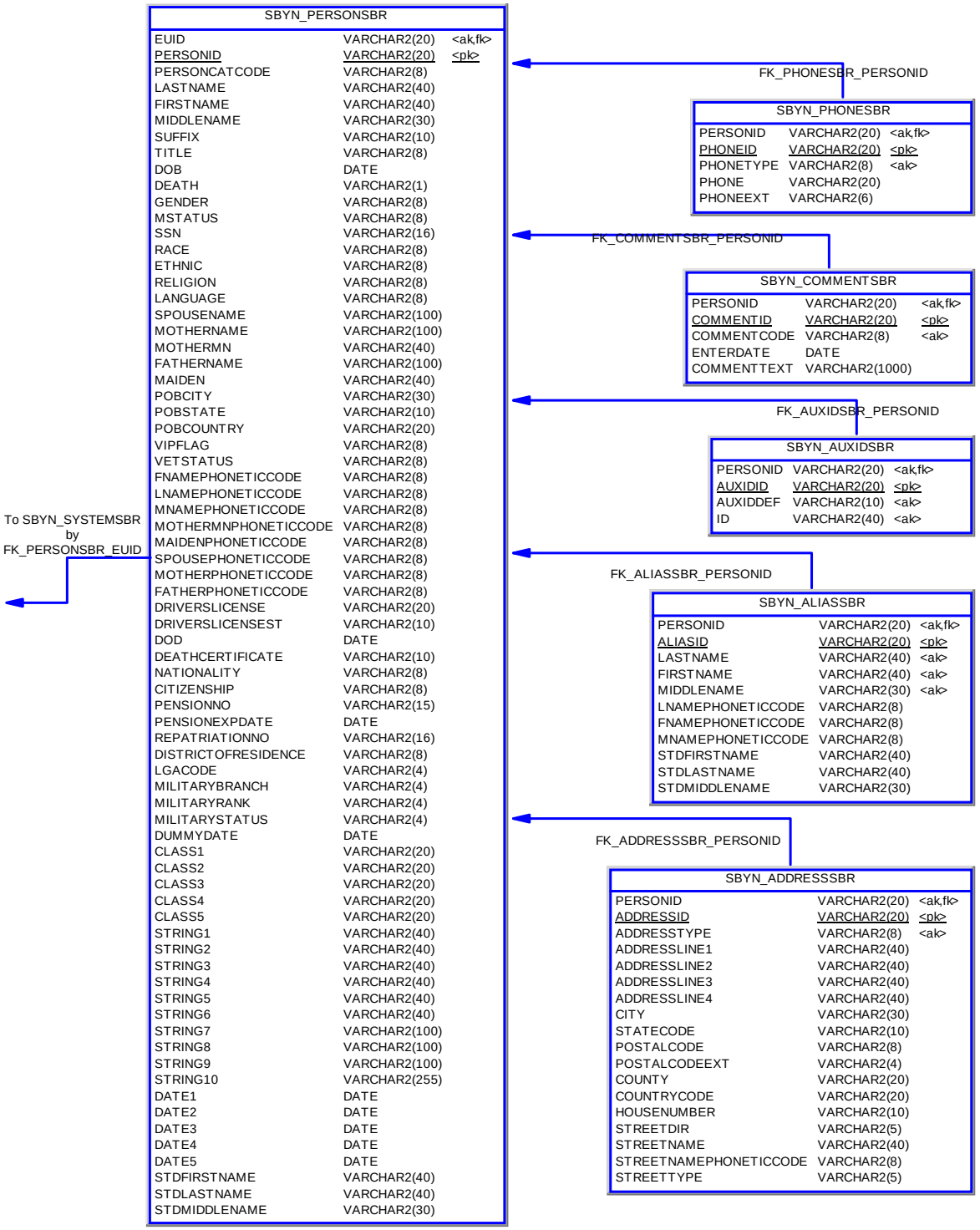
Column Name	Data Type	Description
FORMAT	VARCHAR2(60)	The required data pattern for the field that is constrained by the user code. For more information about possible values and using Java patterns, see "Patterns" in the class list for **java.util.regex** in the Javadocs provided with Java 2Software Development Kit (SDK).
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.

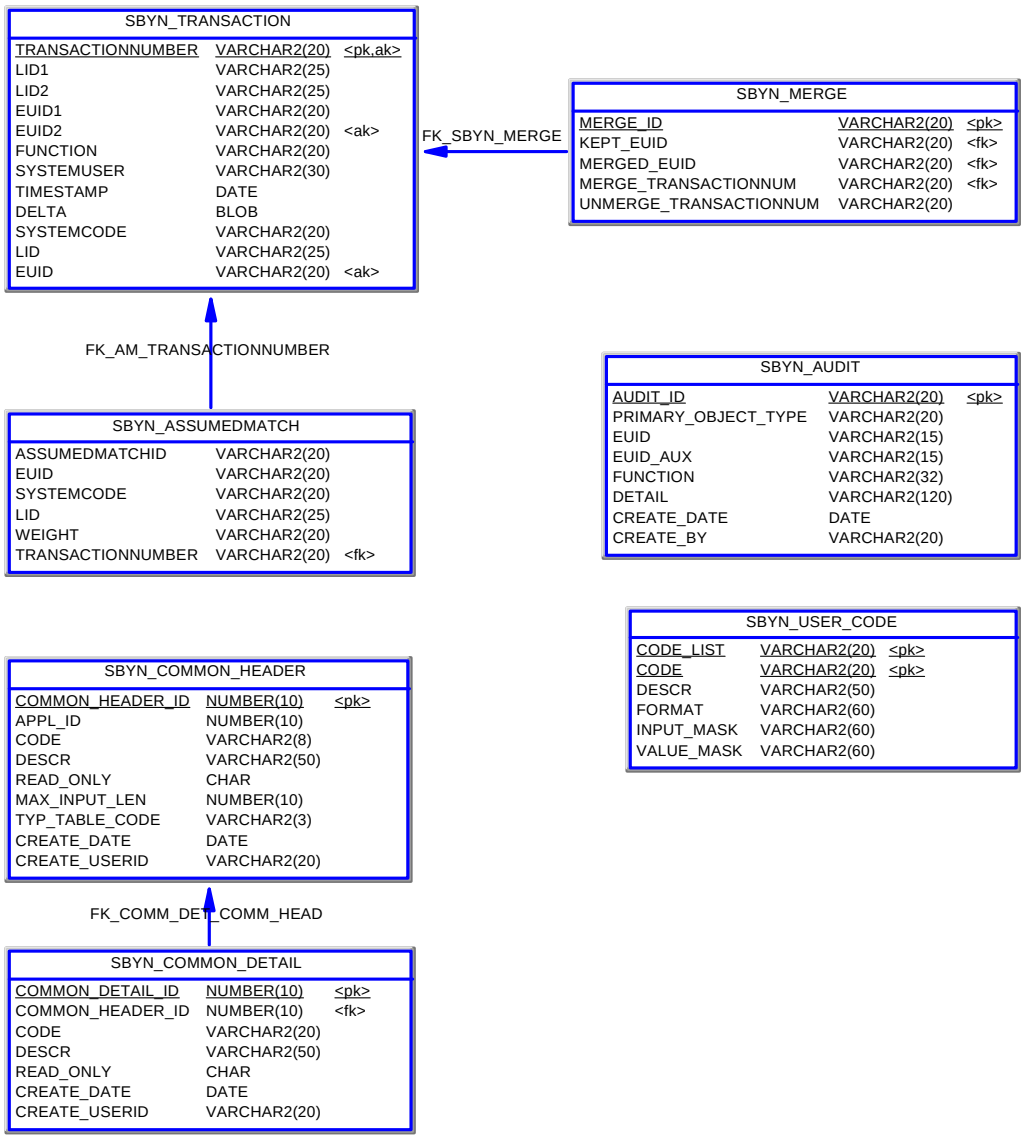
4.3 Sample Database Model

The diagrams on the following pages illustrate the table structure and relationships for a sample eIndex 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)
FK_ADDRESS_PERSONID
SBYN_PHONE
PERSONID VARCHAR2(20) <ak,fk>
PHONEID VARCHAR2(20) <pk>
PHONETYPE VARCHAR2(8) <ak>
PHONE VARCHAR2(20)
PHONEEXT VARCHAR2(6)
FK_PHONE_PERSONID
To SBYN_SYSTEMS by FK_SYSTEMOBJECT_SYSTEMCODE
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)
FK_PERSON_SYSTEMCODE_LID
From SBYN_ENTERPRISE by FK_ENTERPRISE_SYSTEMCODE_LID
SBYN_COMMENT
PERSONID VARCHAR2(20) <ak,fk>
COMMENTID VARCHAR2(20) <pk>
COMMENTCODE VARCHAR2(8) <ak>
ENTERDATE DATE
COMMENTTEXT VARCHAR2(1000)
FK_COMMENT_PERSONID
SBYN_AUXID
PERSONID VARCHAR2(20) <ak,fk>
AUXIDID VARCHAR2(20) <pk>
AUXIDDEF VARCHAR2(10) <ak>
ID VARCHAR2(40) <ak>
FK_AUXID_PERSONID
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)
FK_ALIAS_PERSONID


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)
From SBYN_SYSTEMOBJECT by
FK_SYSTEMOBJECT_SYSTEMCODE
To SBYN_SYSTEMOBJECT by
FK_ENTERPRISE_SYSTEMCODE_LID
SBYN_ENTERPRISE
SYSTEMCODE VARCHAR2(20) <pk,fk2>
LID VARCHAR2(25) <pk,fk2>
EUID VARCHAR2(20) <pk,fk1>
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)
FK_PERSONSBR_EUID
FK_ENTERPRISE_EUID
FK_SYSTEMSBR_EUID
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)
To SBYN_SYSTEMSBR by FK_PERSONSBR_EUID
FK_PHONESBR_PERSONID
SBYN_PHONESBR
PERSONID VARCHAR2(20) <ak,fk>
PHONEID VARCHAR2(20) <pk>
PHONETYPE VARCHAR2(8) <ak>
PHONE VARCHAR2(20)
PHONEEXT VARCHAR2(6)
FK_COMMENTSBR_PERSONID
SBYN_COMMENTSBR
PERSONID VARCHAR2(20) <ak,fk>
COMMENTID VARCHAR2(20) <pk>
COMMENTCODE VARCHAR2(8) <ak>
ENTERDATE DATE
COMMENTTEXT VARCHAR2(1000)
FK_AUXIDSBR_PERSONID
SBYN_AUXIDSBR
PERSONID VARCHAR2(20) <ak,fk>
AUXIDID VARCHAR2(20) <pk>
AUXIDDEF VARCHAR2(10) <ak>
ID VARCHAR2(40) <ak>
FK_ALIASSBR_PERSONID
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)
FK_ADDRESSSSBR_PERSONID
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 DATE
DELTA BLOB
SYSTEMCODE VARCHAR2(20)
LID VARCHAR2(25)
EUID VARCHAR2(20) <ak>
FK_SBYN_MERGE
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)
FK_AM_TRANSACTIONNUMBER
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)
FK_COMM_DET_COMM_HEAD
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 5

Working with the Java API

eIndex provides several Java classes and methods to use in the Collaborations for an eIndex Project. The eIndex API is specifically designed to help you maintain the integrity of the data in the eIndex database by providing specific methods for updating, adding, and merging records in the database.

What's in This Chapter

- **Overview** on page 64
- **Dynamic Object Classes** on page 65
- **Dynamic OTD Methods** on page 78
- **Dynamic eInsight Integration Methods** on page 94
- **Helper Classes** on page 95

5.1 Overview

This chapter provides an overview of the Java API for eIndex, and describes the dynamic classes and methods that are generated based on the object structure of eIndex. For detailed information about the static classes and methods, refer to the eIndex 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.

5.1.1 Java Class Types

eIndex provides a set of static API classes that can be used with any object structure. eIndex 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.

Dynamic Object Classes

The eIndex 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 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 and to define and retrieve field values for those objects.

Dynamic eInsight Integration Methods

The eIndex 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 API and can be used to query eIndex 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.

5.2 Dynamic Object Classes

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

5.2.1 Parent Object Classes

AJava class is created to represent each parent object defined in the object definition of eIndex. 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 each 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

```
public class <ObjectName>Object
```

Methods

- **<ObjectName>Object** on page 66
- **add<Child>** on page 67
- **addSecondaryObject** on page 67
- **copy** on page 68
- **dropSecondaryObject** on page 68
- **get<ObjectName>Id** on page 69
- **get<Child>** on page 70
- **get<Field>** on page 69
- **getChildTags** on page 70
- **getMetaData** on page 70
- **getSecondaryObject** on page 71
- **getStatus** on page 71
- **set<ObjectName>Id** on page 72
- **set<Field>** on page 72
- **setStatus** on page 73
- **structCopy** on page 73

<*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

```
public 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:* `public 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

None.

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

```
public 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

```
public 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

```
public 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 eIndex.

Syntax

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

Parameters

None.

Returns

A string containing the unique ID of the parent 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 parent object. For example, if the parent object contains a field named "FirstName", the getter method for this field is named "getFirstName".

Syntax

```
public 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:* `public 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*>

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

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

Parameters

None.

Returns

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

Throws

None.

getChildTags

Description

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

Syntax

```
public ArrayList getChildTags()
```

Parameters

None.

Returns

An array of child object names.

Throws

SystemObjectException

getMetaData

Description

getMetaData retrieves the metadata for the parent object.

Syntax

```
public 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

```
public 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

getStatus

Description

getStatus retrieves the status of the object.

Syntax

```
public String getStatus()
```

Parameters

None.

Returns

A string containing the status of the object.

Throws

ObjectException

set<*ObjectName*>Id

Description

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

Syntax

```
public 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

```
public 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

setStatus

Description

setStatus sets the status of the parent object.

Syntax

```
public void setStatus(Object value)
```

Parameters

Name	Type	Description
value	Object	An object containing the value of the **status** field.

Returns

None.

Throws

ObjectException

structCopy

Description

structCopy copies the structure of the specified object node.

Syntax

```
public ObjectNode structCopy()
```

Parameters

None.

Returns

A copy of the structure of the object node.

Throws

ObjectException

5.2.2 Child Object Classes

One Java class is created for each child object defined in the object definition of eIndex. 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 following methods described for the child objects, <*Child*> indicates the name of the child object and <*Field*> indicates the names of a field defined for that object.

Definition

```
public class <Child>Object
```

Methods

- **<Child>Object** on page 74
- **copy** on page 74
- **get<Child>Id** on page 75
- **get<Field>** on page 75
- **getMetaData** on page 76
- **getParentTag** on page 76
- **set<Child>Id** on page 76
- **set<Field>** on page 77
- **structCopy** on page 77

<*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

```
public 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 eIndex.

Syntax

```
public 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

```
public 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:* `public 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

```
public 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 given child object.

Syntax

```
public 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

```
public 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

```
public 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

```
public ObjectNode structCopy()
```

Parameters

None.

Returns

A copy of the structure of the object node.

Throws

ObjectException

5.3 Dynamic OTD Methods

A set of Java methods are created in an OTD for use in the eIndex 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 78
- **activateSystemRecord** on page 79
- **addSystemRecord** on page 79
- **deactivateEnterpriseRecord** on page 80
- **deactivateSystemRecord** on page 81
- **executeMatch** on page 81
- **executeMatchUpdate** on page 82
- **findMasterController** on page 83
- **getEnterpriseRecordByEUID** on page 83
- **getEnterpriseRecordByLID** on page 84
- **getEUID** on page 85
- **getLIDs** on page 85
- **getLIDsByStatus** on page 86
- **getSBR** on page 86
- **getSystemRecord** on page 87
- **getSystemRecordsByEUID** on page 87
- **getSystemRecordsByEUIDStatus** on page 88
- **lookupLIDs** on page 89
- **mergeEnterpriseRecord** on page 89
- **mergeSystemRecord** on page 90
- **searchBlock** on page 91
- **searchExact** on page 91
- **searchPhonetic** on page 92
- **transferSystemRecord** on page 92
- **updateEnterpriseRecord** on page 93
- **updateSystemRecord** on page 94

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, System<ObjectName> systemObject)
```

Parameters

Name	Type	Description
euid	String	The EUID of the enterprise object to which you want to add the system object.
systemObject	System<*ObjectName*>	The system object to be added 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

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 Manager Service and associated runtime components (the second method is **executeMatchUpdate** on page 82). 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** function instead.*

Syntax

```
MatchColResult executeMatch(System<ObjectName> systemObject)
```

Parameters

Name	Type	Description
systemObject	System<*ObjectName*>	The system object to be added to or updated in 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

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

Throws

RemoteException

ProcessingException

UserException

executeMatchUpdate

Like **executeMatch** on page 81, **executeMatchUpdate** processes the system object based on the configuration defined for the eIndex 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(System<ObjectName> systemObject)
```

Parameters

Name	Type	Description
systemObject	System<*ObjectName*>	The system object to be added to or updated in 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

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

Throws

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, see the Javadoc provided with eIndex.

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, 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.

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 not 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 typos 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 an existing enterprise object in the eIndex database with the new values of the specified enterprise object.

Syntax

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

Parameters

Name	Type	Description
enterpriseObject	Enterprise<*ObjectName*>	The enterprise object to be updated.

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

5.4 Dynamic eInsight Integration Methods

A set of Java methods are included in the eIndex Project for use in eInsight interfaces. 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 eInsight interfaces. They are described in the previous section, **"Dynamic OTD Methods"**.

- **executeMatch** on page 81
- **executeMatchUpdate** on page 82
- **getEnterpriseRecordByEUID** on page 83
- **getSBR** on page 86
- **getSystemRecordsByEUID** on page 87
- **getSystemRecordsByEUIDStatus** on page 88

- **getEnterpriseRecordByLID** on page 84
- **getEUID** on page 85
- **getLIDs** on page 85
- **getLIDsByStatus** on page 86
- **lookupLIDs** on page 89
- **searchBlock** on page 91
- **searchExact** on page 91
- **searchPhonetic** on page 92

5.5 Helper Classes

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

5.5.1 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 96**). 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

```
public class System<ObjectName>
```

Fields

- **ClearFieldIndicator Field** on page 96

Methods

- **System<ObjectName>** on page 96
- **getClearFieldIndicator** on page 96
- **get<Field>** on page 97
- **set<ObjectName>** on page 99
- **setClearFieldIndicator** on page 98
- **set<Field>** on page 98
- **set<ObjectName>** on page 99

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 primary parent object as null when performing an update from an external system. When an update is performed in eIndex, 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 eIndex 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

```
public String getClearFieldIndicator()
```

Parameters

None.

Returns

A String 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

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

or

```
public 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

```
public <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

```
public 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

```
public 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

```
public String set<ObjectName>(<ObjectName>Bean object)
```

Parameters

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

Returns

None.

Throws

ObjectException

5.5.2 Parent Beans

A Java Bean is created to represent each parent object defined in the object definition of eIndex. 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

```
public final class <ObjectName>Bean
```

Methods

- **<ObjectName>Bean** on page 100
- **count<Child>** on page 100
- **countChildren** on page 101
- **countChildren** on page 101
- **delete<Child>** on page 102
- **get<Child>** on page 102
- **get<Child>** on page 103
- **get<Field>** on page 103
- **get<ObjectName>Id** on page 104
- **set<Child>** on page 104
- **set<Child>** on page 105
- **set<Field>** on page 105
- **set<ObjectName>Id** 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)

<*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

```
public 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

```
public 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

```
public 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 eIndex.

Syntax

```
public 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

```
public <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

```
public <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

```
public 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:* `public 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 eIndex.

Syntax

```
public 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

```
public 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

```
public 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

```
public 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 eIndex. Do not set this field manually.*

Syntax

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

Parameters

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

Returns

None.

Throws

ObjectException

5.5.3 Child Beans

A Java Bean is created to represent each child object defined in the object definition of eIndex. 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 parent object.

Definition

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

Methods

- **set<ObjectName>Id** on page 106
- **<Child>Bean** on page 107
- **delete** on page 107
- **get<Field>** on page 108
- **get<Child>Id** on page 108
- **set<Field>** on page 109
- **set<Child>Id** on page 109

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

Syntax

```
public 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

```
public 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:* `public 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 eIndex.

Note: *This ID is set internally by eIndex. Do not set this field manually.*

Syntax

```
public 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

```
public 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.

Syntax

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

Parameters

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

Returns

None.

Throws

ObjectException

5.5.4 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

```
public class DestinationEO
```

Methods

- **getEnterprise<ObjectName>** on page 110

---

getEnterprise<*ObjectName*>

Description

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

Syntax

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

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

Parameters

None.

Returns

The surviving enterprise object from a merge transaction.

Throws

None.

5.5.5 Search<*ObjectName*>Result

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

Definition

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

Methods

- **getEUID** on page 111
- **getComparisonScore** on page 111
- **get<ObjectName>** on page 111

getEUID

Description

getEUID retrieves the EUID of a search result record.

Syntax

```
public 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

```
public String getComparisonScore()
```

Parameters

None.

Returns

A string containing a comparison weight.

Throws

None.

get<*ObjectName*>

Description

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

Syntax

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

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

Parameters

None.

Returns

An object bean.

Throws

None.

5.5.6 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

```
public class SourceEO
```

Methods

- **getEnterprise<ObjectName>** on page 112

---

getEnterprise<*ObjectName*>

Description

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

Syntax

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

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

Parameters

None.

Returns

The non-surviving enterprise object from a merge transaction.

Throws

None.

5.5.7 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

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

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

Methods

- **System<ObjectName>PK** on page 113
- **getLocalId** on page 113
- **getSystemCode** on page 114

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 primary 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

```
public 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

```
public String getSystemCode()
```

Parameters

None.

Returns

A string containing the processing code for a system.

Throws

None.

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, (depending on the configuration of matching parameters) the objects are an assumed match and are merged automatically (see "Automatic Merge").

automatic merge
When two records are assumed to be matches of one another (see "Assumed Match"), the system performs an automatic merge to join the records rather than flagging them as potential duplicates.

Blocking Query
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. The criteria is defined in the Candidate Select file.

Candidate Select file
The eIndex 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.

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 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 Manager Service
An eIndex component that provides an interface to all eIndex 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 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 configuration file that specifies any default or custom Java classes that perform field validations when data is processed.

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 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 configuration file that defines normalization, parsing, phonetic encoding, and the match string for an instance of eIndex. 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.

match type
An indicator specified in the **MatchingConfig** section of the Match Field configuration file that tells the match engine which rules to use to match information.

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.

Matching Service
An eIndex 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.

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 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. 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 fields from which source systems should be 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 fields 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 "source 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 EDM tab on the Define Enterprise Object window to display the EDM attributes.

Threshold file
An eIndex 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

A

API classes **64**
appl_id column **47**, **49**
application server **21**
assumedmatch sequence number **53**
assumedmatchid column **47**
audience **12**
audit sequence number **53**
audit_id column **48**

B

Best Record file **16**, **17**
blocking query **38**
booleandata column **51**
bytedata column **51**

C

candidate pool **38**
Candidate Select file **16**
child Bean methods **107–110**
child class methods **74–77**
child objects **44**
childtype column **54**, **57**
client Projects **20**
code column **47**, **49**, **50**, **58**
Code List script **17**
common_detail_id column **49**
common_header_id column **49**
components
  eIndex Project **15**
  eIndex Repository **14**
  Environment **21**
  runtime **22–24**
connectivity components **20**
creatdate column **54**
Create database script **17**
Create User Code Data **17**
Create User Indexes **17**
create_by column **48**
create_date column **47**, **48**, **49**, **50**, **56**
create_userid column **47**, **49**, **50**, **56**
createdate column **57**
createfunction column **54**, **57**
createsystem column **57**
createuser column **54**, **57**
Custom Plug-ins **17**

D

data structure **9**
database
  diagram **59**
  tables **44–45**
database scripts
  Code List **17**
  Create database **17**
  Create User Code Data **17**
  Create User Indexes **17**
  Drop database **17**
  Drop User Indexes **17**
  Systems **17**
datedata column **52**
delta column **58**
Deployment Profile **21**
descr column **47**, **49**, **50**
description column **52**, **55**, **58**
DestinationEO methods **110**
detail column **48**
document
  conventions **12**
document conventions **12**
documents, related **12**
Drop database script **17**
Drop User Indexes **17**
DuplicateThreshold **37**

E

editors
  Java source **15**
  text **14**
  XML **14**
eGate Integrator **26**
eIndex
  runtime components **22–24**
eIndex Manager Service **16**, **23**
eIndex Projects
  components **15**
eIndex Repository
  components **14**
eInsight
  Java methods for **19**
eInsight Integration
  methods **94–95**
eInsight integration **65**
Enterprise Data Manager file **16**, **24**

Enterprise Designer **14**
  Projects **15**
Environment components **21**
EUID column **46**, **47**, **48**, **50**, **51**, **56**, **58**
EUID sequence number **53**
euid_aux column **48**
EUID1 column **53**, **57**
EUID2 column **52**, **58**
eVision Studio
  Java methods for **19**
exact match processing **38**
executeMatch **37**, **65**
External Systems **21**
  method OTD for **18**

F

Field Validation file **17**
floatdata column **52**
format column **55**, **59**
function column **48**, **58**

H

highmatchflag column **52**

I

id_length column **55**
inbound messages **26**
input_mask column **56**, **59**
integerdata column **51**
INTEGRITY **10**

J

Java API **10**, **64**
Java methods, dynamic **18**
Java reference **64**
Java source editor **15**
JMS IQ Managers **21**

K

kept_euid column **51**

L

lid column **46**, **47**, **50**, **54**, **58**
lid1 column **57**
lid2 column **57**
Logical Host **21**
longdata column **51**

M

MasterController **65**
match engine **16**
Match Engine node **18**
Match Field file **16**
match threshold **38**
matching algorithm **10**
Matching Service **16**, **23**
MatchThreshold **37**, **39**
max_input_len column **50**
merge **45**
merge sequence number **53**
merge_euid column **51**
merge_id column **51**
merge_transactionnum column **51**
message processing **38**
  blocking query **38**
  candidate pool **38**
  exact match **38**
  match threshold **38**
  potential duplicates **38**
  same system **38**–**39**
messages
  inbound **26**
  inbound processing **37**
  origin **26**
  outbound **34**
  processing **25**
  routing **27**
  transformation **27**
method OTD **18**, **37**, **65**, **78**–**94**
  classes
    child classes **73**
    parent class **65**
  helper classes
    child bean class **106**
    parent bean class **99**
    Search(Object)Result class **110**
    System(Object) class **110**, **112**, **113**

O

Object Definition **43**
Object Definition file **16**
Object Persistence Service **24**
object structure **10**, **18**
Object Type Definition **18**
OneExactMatch **37**, **39**
outbound messages **34**

P

parent Bean methods **100**–**106**

parent class methods **66–73**
parent objects **44**
path column **51**
potential duplicates **38**, **45**
potentialduplicate sequence number **53**
potentialduplicateid column **52**
primary_object_type column **48**
processing logic **37**
Project components
    Custom Plug-ins **17**
    Deployment Profile **21**
    for connectivity **20**
    Match Engine node **18**
    outbound OTD **18**
    Standardization Engine node **18**
Projects
    client **20**

Q

queries **38**
Query Builder **16**, **23**
Query Manager **24**

R

read_only column **47**, **49**, **50**
related publications **12**
resolvedcomment column **52**
resolveddate column **52**
resolveduser **52**
revisionnumber column **57**
runtime environment
    functions **21**
    overview **21**

S

same system processing **38–39**
SameSystemMatch **37**
SBR
    see single best record
sbyn_(child_object) **44**, **46**
sbyn_(child_object)sbr **44**, **46**
sbyn_(object_name) **44**, **45**
sbyn_(object_name)sbr **44**, **46**
sbyn_appl **44**, **47**
sbyn_appl sequence number **54**
sbyn_assumedmatch **44**, **47**
sbyn_audit **44**, **48**
sbyn_common_detail **44**, **49**
sbyn_common_detail sequence number **54**
sbyn_common_header **44**, **49**
sbyn_common_header sequence number **54**
sbyn_enterprise **45**, **50**
sbyn_merge **45**, **51**
sbyn_overwrite **45**, **51**
sbyn_potentialduplicates **45**, **52**
sbyn_seq_table **45**, **53**
sbyn_system **45**
sbyn_systemobject **45**, **54**
sbyn_systems **55**
sbyn_systemsbr **45**, **56**
sbyn_transaction **45**, **57**
sbyn_user_code **58**
sbyn_user_table **45**
Screenshots **12**
search object result methods **111**
Security **21**
    file **17**
SeeBeyond Match Engine **10**
    configuration files **18**
seq_count column **53**
seq_name column **53**
sequence numbers
     54
    assumedmatch **53**
    audit **53**
    EUID **53**
    merge **53**
    potentialduplicate **53**
    sbr **54**
    sbyn_appl **54**
    sbyn_common_detail **54**
    sbyn_common_header **54**
    transactionnumber **53**
Services **21**, **26**
single best record **10**, **24**, **43**, **44**
SourceEO methods **112**
standardization engine **16**
Standardization Engine node **18**
STATUS column **55**
status column **52**, **55**, **57**
stringdata column **51**
survivor calculator **10**, **16**, **24**
survivor strategy **24**
system object primary key methods **113–114**
system record **44**
systemcode column **46**, **47**, **50**, **54**, **55**, **58**
Systems script **17**
systemuser column **58**

T

text editor **14**
Threshold file **16**
timestamp column **58**

timestampdata column **52**
transaction history **21**, **45**
transactionnumber column **48**, **52**, **57**
transactionnumber sequence number **53**
typ_table_code column **50**
type column **51**, **52**

U

unmerge_transactionnum column **51**
update **38**
Update Manager **16**, **24**
update policies **17**
update_date column **56**
update_userid column **56**
UPDATEDATE column **55**
updatedate column **57**
updatefunction column **55**, **57**
update-mode **38**
updateuser column **55**, **57**

V

value_mask column **56**, **59**

W

Web Connectors **21**
weight column **48**, **52**

X

XML editor **14**