Developing Oracle® Java CAPS Master Patient Indexes

Part No: 821–2566
March 2011

120126@25097

Contents

Developing Oracle Java CAPS Master Patient Indexes

The topics listed here provide procedures, conceptual information, and reference information for using Oracle Java CAPS Master Patient Index to design and create a master index application. These topics help you get started creating and using a master index application, but they do not include all activities related to the master index application. For a complete list of topics that provide information and instructions for implementing a master patient index application, see "Related Topics" on page 6.

What You Need to Know

These topics provide information you should to know before you start creating a master index application.

- "Oracle Java CAPS Master Patient Index Overview" on page 6
- "Master Index Development Process Overview" on page 18
- "The Master Patient Index Framework and the Runtime Environment" on page 19
- "Before You Begin Developing a Master Index" on page 20
- "Custom Plug-ins for Master Index Custom Transaction Processing" on page 22
- "Custom Plug-Ins for Master Index Custom Components" on page 27
- "Master Index Database Scripts and Design" on page 33

What You Need to Do

These topics provide instructions on how to design and create master index applications.

- "Implementing Master Index Custom Plug-ins" on page 29
- "Generating the Master Index Application" on page 31
- "Creating the Master Index Database" on page 38
- "Deleting Master Index Database Tables and Indexes" on page 49

More Information

These topics provide additional information you should know when creating a master index application

- "Master Index Database Table Description for sbyn_systems" on page 42
- "Master Index Database Table Description for sbyn_user_code" on page 46

Related Topics

Several topics provide information and instructions for implementing and using a master patient index application. The following topics are designed to be used together when implementing a master index application:

- *Developing Oracle Java CAPS Master Patient Indexes*
- *Getting Started With Oracle Java CAPS Master Patient IndexGetting Started With Oracle Java CAPS Master Patient Index*
- *Working With the EDM for Oracle Java CAPS Master Patient Index*
- *Configuring Oracle Java CAPS Master Indexes (Repository)*
- *Understanding Oracle Java CAPS Master Index Configuration Options (Repository)*
- *Configuring Oracle Java CAPS Master Index (Repository) Connectivity and Environments*
- *Deploying Oracle Java CAPS Master Indexes (Repository)*
- *Analyzing and Cleansing Data for a Master Index*
- *Loading the Initial Data Set for a Master Index*
- *Maintaining Oracle Java CAPS Master Indexes (Repository)*
- *Understanding Oracle Java CAPS Master Index Processing (Repository)*
- *Understanding the Oracle Java CAPS Match Engine*

Oracle Java CAPS Master Patient Index Overview

The following topics provide an overview of Oracle Java CAPS Master Patient Index and how it works with other Java CAPS components. They also include descriptions of the design-time and runtime components of a master patient index.

- "About Oracle Java CAPS Master Patient Index" on page 6
- "Oracle Java CAPS Master Patient Index Repository Components" on page 10
- "Oracle Java CAPS Master Patient Index Runtime Environment Components" on page 14
- "Oracle Java CAPS Master Patient Index Enterprise Records" on page 16

About Oracle Java CAPS Master Patient Index

In today's healthcare environment, vital patient data is generated and stored in several systems throughout an organization. Each of these systems typically assigns its own, independent local identifiers, making it difficult to share information between systems and departments. It is also difficult to create a comprehensive, reliable view of each patient across a healthcare enterprise.

Patient information should flow seamlessly and rapidly between internal systems and departments throughout the entire healthcare network, and each department should have access to the most current and reliable patient data. As organizations grow, merge, and form affiliations, sharing data between different information systems becomes a complicated task. Oracle Java CAPS Master Patient Index can help you manage this task and ensure that the data you have is the most current and accurate information available.

The topics below provide information about Oracle Java CAPS Master Patient Index and how it provides a solution for sharing and cleansing patient data.

- "The Oracle Java CAPS Master Patient Index Solution" on page 7
- "Configurability" on page 8
- "Standardization and Matching Logic" on page 8
- "Data Maintenance" on page 9
- "Oracle Java CAPS Master Patient Index and HIPAA" on page 9

The Oracle Java CAPS Master Patient Index Solution

Oracle Java CAPS Master Patient Index is an enterprise-wide master patient index (EMPI) built on the Oracle Java CAPS Master Index (Repository) platform. It provides a comprehensive, reliable view of patient information by uniquely identifying patients throughout a healthcare enterprise and maintaining the most current information about those patients. With Oracle Java CAPS Master Patient Index, it is possible to create a single source of patient information that synchronizes with your existing systems.

Oracle Java CAPS Master Patient Index cross-references data from all connected systems and automates record matching across disparate systems, simplifying the process of sharing data between departments and facilities. Oracle Java CAPS Master Patient Index is highly flexible and customizable, and you can configure the master index as needed to meet your data processing needs. The master index uniquely identifies each patient record throughout an organization to ensure that the most current and accurate data is available.

Oracle Java CAPS Master Patient Index provides an automatic, common identification process regardless of the system from which data originates. Records from various locations are cross-referenced using an enterprise-wide unique identifier assigned by Oracle Java CAPS Master Patient Index, allowing the master index to use the local identifiers generated by your internal systems to create an index of patient records. In addition, Oracle Java CAPS Master Patient Index employs configurable probabilistic matching technology, which uses a matching algorithm to formulate an effective statistical measure of how closely records match based on the data fields you specify. Using this matching logic, Oracle Java CAPS Master Patient Index consistently and precisely identifies patient records, flagging potentially duplicate records and automatically joining records that are considered a match. In this way, Oracle Java CAPS Master Patient Index provides continuous data cleansing as records are processed.

Oracle Java CAPS Master Patient Index centralizes the information about the patients that participate within your organization. Maintaining a centralized database for multiple systems enablesOracle Java CAPS Master Patient Index to integrate data throughout the enterprise while allowing your existing systems to continue to operate independently. The database, which is accessible throughout the enterprise, stores copies of local system records and their associated single best records (SBRs), which represent the most accurate and complete data for each patient. To facilitate up-to-date records in each system, you can configure Oracle Java CAPS Master Patient Index to generate a message to a JMS Topic each time a record is updated, added, merged, or unmerged in the master index. Using the Oracle Java CAPS Enterprise Service Bus (ESB), this information becomes available to those systems that are able to accept incoming messages.

Configurability

Oracle Java CAPS Master Patient Index provides a predefined data structure based on standard healthcare data requirements that can be used as is or can be easily customized if needed. Before deploying Oracle Java CAPS Master Patient Index, you define the components and processing capabilities of the system to suit your requirements. The matching and standardization rules, survivorship rules, queries, Patient Enterprise Data Manager (Patient EDM) appearance, and field validation rules can all be used as is or can be configured to better meet the needs of your organization. In essence, you control the data structure and the logic that determines how data is updated, standardized, weighted, and matched.

The data structure and processing logic is stored in a set of XML configuration files that are predefined but that can be customized. These files are defined within the context of Java CAPS project and are modified using the XML editor provided in NetBeans. You can also use the graphical configuration editor to customize some of the master patient index configuration.

Standardization and Matching Logic

Sharing data requires overcoming data quality problems such as name and address changes, transpositions, and phonetically similar names to be able to uniquely identify the same patient across multiple systems. Oracle Java CAPS Master Patient Index uses the Oracle Java CAPS Match Engine, a proprietary algorithm for probabilistic matching of patient records and data standardization. As records are processed through Oracle Java CAPS Master Patient Index, the standardization engine normalizes and phonetically encodes specified data and the match engine identifies records that potentially represent or do represent the same patient. The match engine uses user-defined logic, including configurable matching thresholds, comparison functions, data fields, and so on.

Matching Weight Determination

When comparing two records to determine the likelihood of a match, the match engine compares the match fields you specify between the records to determine a matching weight for each match field based on the reliability of the field and the comparison function used. The sum of the weights of the match fields is the total matching weight between the two records. The logic used by the standardization and match engines is highly customizable to provide the most reliable matching for the type of data you store.

Alias Processing

Oracle Java CAPS Master Patient Index provides alias name processing in the form of custom plug-ins to help find or match patient records in cases where the patient's name has changed or a nickname is used. In the default configuration, a name is added to a patient's alias list when a maiden name is added or updated and when a patient's first, last, or middle name is modified. Searches can be performed against a patient's primary and alias names, providing broad search capabilities and improving the chance of finding a match.

Data Maintenance

The Patient EDM is the web-based user interface for Oracle Java CAPS Master Patient Index. The Patient EDM 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. From the Patient EDM, you can perform searches using a variety of criteria and search types for a specific patient or a set of patients. For certain searches, the results are assigned a matching weight that indicates the probability of a match.

One of the most important features of Oracle Java CAPS Master Patient Index is its ability to match records and identify possible duplicates. Oracle Java CAPS Master Patient Index also provides the functionality to correct any duplication. Potential duplicate records are easily corrected by either merging the records in question or marking the records as “resolved”. If you find two records to represent the same person, you should merge the records (at either the enterprise record or system record level). At the enterprise record 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.

Finally, Patient EDM provides standard reports that provide information about the current state of the data in the master index, helping you monitor stored data and determine how that data needs to be updated. Report information also helps verify that the matching logic and weight thresholds are defined correctly. Standard reports are available through a command line or the Patient EDM. You can also create custom reports using any ODBC-compliant reporting tool, SQL, or Java.

Oracle Java CAPS Master Patient Index and HIPAA

Oracle Java CAPS Master Patient Index provides full audit capabilities in support of the Health Insurance Portability and Accountability Act (HIPAA) mandates. Transaction histories are stored in the database to track every change to every record and provide before and after images as well as who made the changes and when. In addition, the audit log maintains a record of each time patient data is accessed or viewed in the master index database, including who accessed the data and when. The audit log and transaction history can both be viewed on the Patient EDM.

Oracle Java CAPS Master Patient Index Repository Components

Oracle Java CAPS Master Patient Index has two types of components: Repository and runtime. The Repository components work within NetBeans and are used during the design and configuration phases to create and customize the master patient index and to define connectivity between external systems and Oracle Java CAPS Master Patient Index. The primary Repository components include the following:

- "Editors" on page 10
- "Project Components" on page 10
- "Environment Components" on page 14

Editors

Oracle Java CAPS Master Patient Index provides the following editors to help you customize the files in the Oracle Java CAPS Master Patient Index project.

- **Configuration Editor (Repository)** - Allows you to customize certain portions of the XML configuration files using a graphic interface. The Configuration Editor provides validation services for file structure and syntax.
- **XML Editor** - Allows you to review and customize the XML configuration files. The editor provides schema validation services and verification for XML syntax. The XML editor is automatically launched when you open a Oracle Java CAPS Master Patient Index configuration file.
- **Text Editor** – Allows you to review and customize the database scripts for the master index. This editor is very similar to the XML editor but without the verification services. The text editor is automatically launched when you open a Oracle Java CAPS Master Patient Index database script or configuration file.
- **Java Source Editor** – Allows you to create and customize custom plug-in classes for the master index application. 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.

Project Components

Oracle Java CAPS Master Patient Index is implemented within a project in NetBeans. The Oracle Java CAPS Master Patient Index project includes a set of configuration files, database files, and custom plug-ins that you can modify in order to customize your master index implementation. It includes additional components that are automatically updated when you generate the project, including a method Object Type Definition (OTD), an outbound OTD, Business Process methods, database scripts, and application JAR files. To complete the project, you create a Connectivity Map and Deployment Profile.

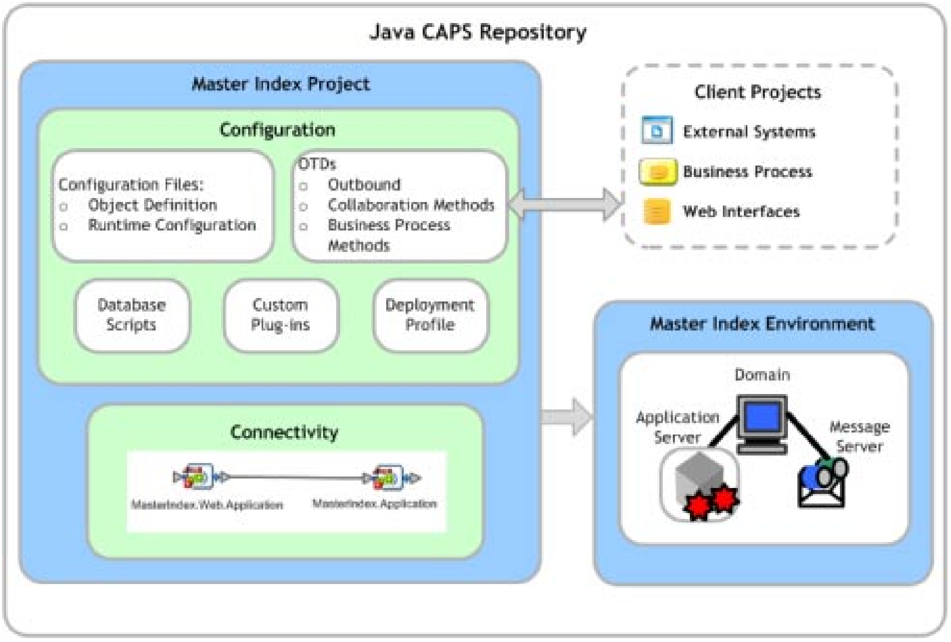
Additional Java CAPS components can be added to the client projects that share data with Oracle Java CAPS Master Patient Index, including Services, Collaborations, OTDs, Web Connectors, Adapters, JMS Queues, JMS Topics, Business Processes, and so on.

The primary Oracle Java CAPS Master Patient Index project components include the following. Each is described in more detail below.

- "Configuration Files" on page 12
- "Database Scripts" on page 12
- "Custom Plug-ins" on page 13
- "Match Engine Configuration Files" on page 13
- "Object Type Definition (OTD)" on page 13
- "Dynamic Java Methods" on page 13
- "Connectivity Components" on page 14
- "Deployment Profile" on page 14

The following figure illustrates the project and Environment components of Oracle Java CAPS Master Patient Index.

FIGURE 1 Master Patient Index Design-Time Components



Configuration Files

These files define the configuration of the runtime environment, such as the object structure, how matching is performed, how the SBR is created, and so on. The runtime components configured by these files are listed in "Oracle Java CAPS Master Patient Index Runtime Environment Components" on page 14

- **Object Definition** - Defines the data structure of the object being indexed in a master patient index application.
- **Enterprise Data Manager** - Configures the search functions and appearance of the Patient EDM, along with debug information and security information for authorization.
- **Candidate Select** - Configures the Query Builder component of the master index application and defines the queries available for the index.
- **Match Field** - Configures the Matching Service and defines the fields to be standardized or used for matching. It also specifies the match and standardization engines to use.
- **Threshold** - Configures the 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 single best record (SBR). You can define custom update procedures in this file.
- **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

These scripts contain the SQL statements used to create the database and the required start-up data. A script is also included to create additional indexes against the database based on the predefined blocking query. A script to drop the database is provided for testing purposes.

- **Systems** - Contains the SQL insert statements that add the external systems you specified in the wizard to the database. You can define additional systems in this file.
- **Code List** - Contains the SQL statements to insert processing codes and drop-down list values into the database. Some of the entries in this file are generated by the wizard. Code lists must be defined in this file to make them available to the master index application.
- **Create Person database** - Defines the structure of the master index database based on the object structure specified in the wizard. You can customize this file and then run it against a database instance to create a customized master index 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 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 the master index application 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.

Custom Plug-ins

Custom plug-ins allow you to incorporate custom logic into Oracle Java CAPS Master Patient Index by creating user-defined Java classes. Several custom plug-ins are already provided that automatically create aliases for person names when certain updates are made to a record. For example, if the first, last, middle, or maiden name is changed during a transaction, the previous name is added as an alias.

Match Engine Configuration Files

These files define characteristics of the standardization and matching processes. The configuration files under that 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 these files as needed.

Object Type Definition (OTD)

The outbound OTD in the Oracle Java CAPS Master Patient Index project is based on the object structure define in the Object Definition file. The OTD is used for distributing information that has been added or updated in Oracle Java CAPS Master Patient Index to external systems. 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 the master index application data available to external systems, you must define a JMS Topic in the master index Connectivity Map to which the master index application can publish transactions.

Dynamic Java Methods

These methods are used in Collaborations and Business Processes to process data through the master index. These methods are generated dynamically based on the object structure defines in the Object Definition file. 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 *Understanding Oracle Java CAPS Master Index Processing (Repository)*.

Connectivity Components

The master index project Connectivity Map consists of two required components: the web application service and the application service. Two optional components are a JMS Topic for broadcasting messages and a database Adapter for database connectivity. In client project Connectivity Maps you can use any of the standard project components to define connectivity and data flow to and from the master index application. Client projects include those created for the external systems sharing data with the index through a Collaboration or Business Process.

For client projects, you can use connectivity components from the master index server project and any standard Java CAPS connectivity components, such as OTDs, Services, Collaborations, JMS Queues, JMS Topics, and Adapters. Client project components transform and route incoming data into the master index database according to the rules contained in the Collaborations or Business Processes. They can also route the processed data back to the appropriate local systems through Adapters.

Deployment Profile

The Deployment Profile defines information about the production environment, including information about the assignment of Services and message destinations to application servers and JMS IQ Managers within the Oracle Java CAPS Master Patient Index system. Each Oracle Java CAPS Master Patient Index project must have at least one Deployment Profile, and can have several, depending on the project requirements and the number of Environments used. You need to deploy the Oracle Java CAPS Master Patient Index server project before deploying the client projects in order to make the master index application available to the client Deployment Profiles.

Environment Components

The Oracle Java CAPS Master Patient Index Environments define the deployment environment of the runtime components, including the Logical Host and application server. For client projects referencing the Oracle Java CAPS Master Patient Index project, an Environment might also include a JMS IQ Manager, constants, Web Connectors, and External Systems. Each Environment represents a unit of software that implements Oracle Java CAPS Master Patient Index. You must define and configure at least one Environment for Oracle Java CAPS Master Patient Index before you can deploy the application. The application server hosting Oracle Java CAPS Master Patient Index is configured within the Environment.

Oracle Java CAPS Master Patient Index Runtime Environment Components

Regardless of how you define the data structure and configure the runtime environment for Oracle Java CAPS Master Patient Index, the final product provides a customized master patient index to help you manage data from disparate systems and ensure that the data you have is the most current and accurate information available. The runtime environment includes all of the components you create for Oracle Java CAPS Master Patient Index connectivity as well as the web-based Patient EDM, which allows you to manually monitor and maintain patient data.

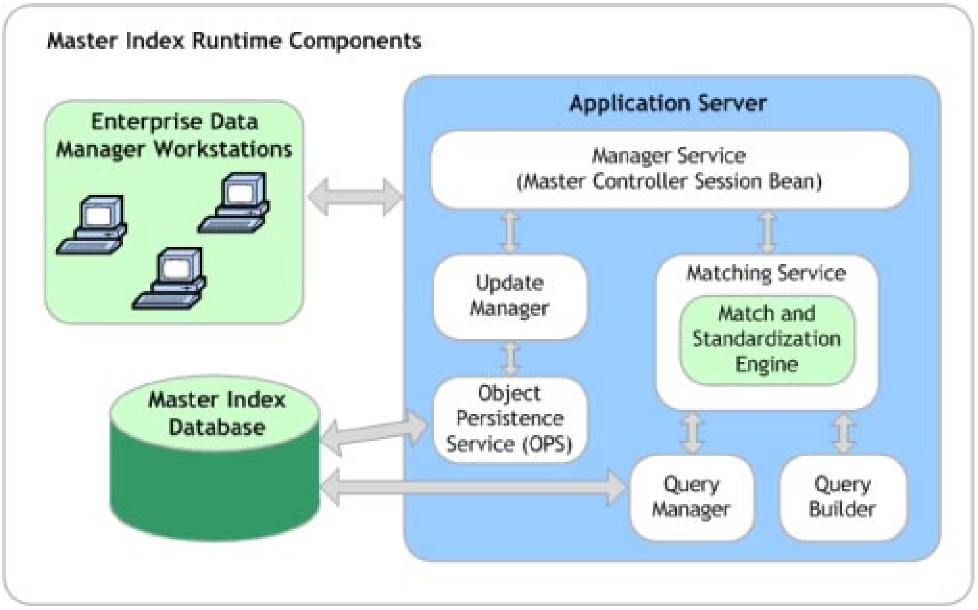
As with other master indexes built on the Oracle Java CAPS Master Index (Repository) platform, the Oracle Java CAPS Master Patient Index runtime environment is made up of several components that work together to form a complete indexing system. The runtime environment includes the following primary components:

- Matching Service
- eView Manager Service
- Query Builders
- Query Manager
- Update Manager
- Object Persistence Service (OPS)
- Database
- Enterprise Data Manager

In addition, Oracle Java CAPS Master Patient Index uses the connectivity components defined in the Oracle Java CAPS Master Patient Index server and client Projects to route data between external systems and the Oracle Java CAPS Master Patient Index database. The Repository stores information about the configuration and structure of the runtime environment. Because Oracle Java CAPS Master Patient Index is deployed to an application server, it can be implemented in a distributed environment.

The following figure illustrates the runtime components of a master patient index.

FIGURE 2 Master Patient Index Runtime Components


Master Index Runtime Components
Enterprise Data Manager Workstations
Application Server
Manager Service (Master Controller Session Bean)
Update Manager
Matching Service
Match and Standardization Engine
Master Index Database
Object Persistence Service (OPS)
Query Manager
Query Builder

For more information about the functions, features, and components of the eIndex runtime environment, see "Master Index Runtime Components" in *Developing Oracle Java CAPS Master Indexes (Repository)*.

Oracle Java CAPS Master Patient Index Enterprise Records

An *enterprise record* is identified by an EUID assigned by Oracle Java CAPS Master Patient Index and includes all components of a record that represents one patient. The structure of the data is defined in the Object Definition file.

Oracle Java CAPS Master Patient Index stores two different types of records in each enterprise record: system records and a single best record (SBR). A system record contains a patient's information as it appears in an incoming message from an external system. An enterprise record's SBR stores data from a combination of external systems and it represents the most reliable and current information contained in all system records for a patient. An enterprise record consists of one or more system records and one SBR.

System Records

The structure of a system record is different from the SBR in that each system record contains a system and local ID pair. The remaining information contained in the system records of an enterprise record is used to determine the best data for the corresponding SBR. If an enterprise record only contains one system record, the SBR is identical to that system record (less the system and local ID information). However, if the enterprise record contains multiple system records, the SBR might be identical to one system record but will more likely include a combination of information from all system records.

The Single Best Record

The SBR for a patient is created from the most reliable information contained in each system record representing that patient. The information used from each external system to populate the SBR is determined by the survivor calculator, which is configured in the Best Record file. This data is determined to be the most reliable information from all system records in the enterprise record. The survivor calculator can consider factors such as the relative reliability of an external system, how recent the data is, and whether the SBR contains any “locked” field values. You define the rules that select a field value to be persisted in the SBR.

Objects in an Enterprise Record

In Oracle Java CAPS Master Patient Index, each system record and SBR in an enterprise record typically contain a set of objects that store different types of information about a patient. A record contains one parent object and typically contains several child objects, but it can have no child objects at all. A record can have only one instance of the parent object, but can have multiple instances of each type of child object. For example, in the default configuration, the parent object (Person) contains demographic data. A record can only contain one patient name and social security number (stored in the Person object), but the record could have multiple addresses, telephone numbers, and aliases, which are defined in different child objects (*address*, *phone*, and *alias* objects respectively). A record can have multiple instances of each child object, such as a home and a billing address.

Oracle Java CAPS Master Patient IndexIdentification Codes

Another key component of an enterprise record is the identification codes used to identify and cross-reference each patient. Each record in the master index is assigned an enterprise-wide unique identification number (EUID) in addition to the local IDs assigned by the individual systems in the network. Each unique patient has one unique identification number throughout your organization, and a unique identification number within each system with which they are registered. Patients might also have several auxiliary IDs. An auxiliary ID is an identification code that does not necessarily uniquely identify a single patient within the database, but might identify a group of patients. For example, if a family shares the same account or insurance policy, every family member would have the same identification code for that account or policy.

Master Index Development Process Overview

Oracle Java CAPS Master Patient Index makes the process of implementing a master patient index simple by providing a default object structure and configuration based on standard healthcare data and processing requirements. It also provides the ability to customize the default configuration as needed to fine-tune the index to match your specific needs.

The following steps outline and link to the procedures you need to follow to develop a master index application. Not all tasks are covered in this document. Where a process is covered in a separate document, the link is in italics.

1. Perform a preliminary analysis of the data you plan to store in the master index application.
2. Define and build custom plug-ins, and specify the plug-ins in the appropriate configuration file (described in "Implementing Master Index Custom Plug-ins" on page 29).
3. Customize the configuration files (instructions for this step are provided in a separate document, *Configuring Oracle Java CAPS Master Indexes (Repository)*).
4. Generate the master index application (described in "Generating the Master Index Application" on page 31).
5. Create the database (described in "Creating the Master Index Database" on page 38).
   - Customize the database scripts by defining system information, processing codes, and drop-down menu values.
   - Create the database structure and any necessary indexes.
   - Define database connectivity in the application server.
6. Analyze and cleanse existing data that will be preloaded into the master index database (instructions for this step are provided in a separate document, *Analyzing and Cleansing Data for a Master Index*).
7. Create Connectivity Maps (described in *Configuring Oracle Java CAPS Master Index (Repository) Connectivity and Environments*).
   - Create and define the components in the Connectivity Maps, such as Collaborations, Services, and External Applications.
   - Configure the Connectivity Maps.
8. Define the Environment (described in *Configuring Oracle Java CAPS Master Index (Repository) Connectivity and Environments*).
9. Create the Deployment Profile and deploy the project (described in *Deploying Oracle Java CAPS Master Indexes (Repository)*).
10. Define security (described in "Defining Master Index Security (Repository)" in *Maintaining Oracle Java CAPS Master Indexes (Repository)*).
11. Load the initial data set into the master index database (this is described in a separate document, *Loading the Initial Data Set for a Master Index*).

The Master Patient Index Framework and the Runtime Environment

The values you enter in the Configuration Editor or directly in the XML files define how other components of the master index application are generated, such as the database scripts, the Patient Enterprise Data Manager, and the dynamic Java API. This section provides an overview of how the values you enter correspond to the runtime environment.

From XML to the Database

The master index database is created using a standard Oracle or SQL Server database and a database script generated directly from the Object Definition file. Additional scripts are created based on any user codes or menu lists you defined for the fields in the object structure. Running the database scripts against the database creates the tables necessary for your master index application and also creates startup data, such as external system information, processing codes, and so on.

From XML to the Patient Enterprise Data Manager

Based on information you specify in the Configuration Editor, edm.xml is generated to define the appearance of the Patient Enterprise Data Manager (Patient EDM) . This file defines the fields and appearance of the Patient EDM and also specifies the searches used by the Patient EDM. The available search types are defined in the Candidate Select file. You can customize many features of the Patient EDM, including the following.

- The fields that appear on the Patient EDM pages
- The field attributes, such as a display name, order of appearance, length, type, data format, and so on
- The types of searches that can be performed and the fields available for each type
- The appearance of search results lists
- The location of the fields on all windows

From XML to the Connectivity Components

When you generate the master index application, several connectivity components are created, including a method OTD, Business Process methods, and an outbound OTD. All are based on the Object Definition file. The method OTD contains certain Java methods for use in Collaborations to specify how data is processed into the database. The outbound OTD is used when publishing messages processed by the master index application for broadcasting to external systems. Generating a project also creates application files that you can drag into the Connectivity Map.

From XML to the Runtime Environment

The information you specify in the master index configuration files is stored in the Repository and is read at runtime when the domain is started. The only exception is the Object Definition file, which is stored only as a record of the object structure. You can modify the configuration files after moving to production; however, for the changes to take effect, you must regenerate the application and then rebuild and redeploy the project to apply the changes to the server. You also need to restart the Patient EDM and any Adapters or Binding Components connected to the application for the changes to take effect. Use caution when modifying these files; changing these files after moving to production might result in loss of data integrity or unexpected weighting and matching results.

Before You Begin Developing a Master Index

Creating a master index application requires in-depth analyses of your business requirements, legacy data, and data processing requirements. After the initial analysis, you can plan and design how you will create the master index application framework and how you will customize that configuration after creating the framework. After creating the master index application, you should perform an in-depth data analysis using the tools provided. This analysis will help you define matching and standardization rules and logic. The data analysis tool is provided in the service–enabled version of Oracle Java CAPS Master Patient Index only, so you would need to replicate your repository–based project using the service–enabled version in order to use the tool. You also need to plan and design each physical component of the Oracle Java CAPS Master Patient Index project and any projects referencing the master index application files..

The following topics provide information about what you need to know and do before you begin to create a master index application.

- "Preliminary Data Analysis for a Master Index" on page 20
- "Planning a Master Index Project" on page 21
- "Master Index Project Initiation Checklist" on page 21

Preliminary Data Analysis for a Master Index

Before creating the master index application, perform a preliminary analysis against the data that will be stored in the index database to determine the structure of the records. Analyzing your data requires reviewing the message structure of each legacy system that will share data with the master index application. The master index application does not need to store every field from every system, so you can narrow the master index application object structure to just the pertinent fields from each system. However, the master index application stores the same fields for each system. A more in-depth analysis of the field values in the legacy data occurs after the initial master index application framework is created.

Planning a Master Index Project

Before you create the Oracle Java CAPS Master Patient Index project, analyze the business requirements and determine which project components will help you meet those requirements. Planning the project includes defining how each external system will share information with the master index application and how the master index application will share information with those external systems. In addition, you can incorporate master index Java methods that define how the master index application processes incoming data. Master index methods can also be used to transform the data sent from external systems into a format that can be read by the master index application.

An additional consideration is whether to integrate the master index methods into a Business Process.

Master Index Project Initiation Checklist

Before you begin developing your master index application, make sure you have obtained the following information:

- The primary object to be indexed, such as a person, customer, business, and so on (in the case of Master Index, this is likely to be a Patient or Person object)
- Any secondary objects, such as telephone numbers and addresses
- All fields to be stored in the index for both the primary and secondary objects
- The name of each field as it appears on the Patient EDM and whether the field will be a standard text field or will be populated from a menu list (if a field will be populated from a menu list, you should also define an eight-character name for the list)
- The fields that are required in order to add a record or that are required for queries
- The fields that will appear on reports
- The fields that must be unique to an enterprise record (in other words, they uniquely identify a child object within an enterprise record)
- The fields that will be used for matching
- The fields that will need to be parsed or normalized prior to matching
- Any special formatting requirements, such as character types, the data type, minimum and maximum values, and field size
- The fields that will appear on Patient EDM search and search results windows
- The processing codes for the source systems being integrated into the index

Custom Plug-ins for Master Index Custom Transaction Processing

You can add custom processing to the master index application using the Custom Plug-ins module of a Oracle Java CAPS Master Patient Index project. This ability allows you to tailor how messages are processed by the master index application. Plug-ins can be used to customize field validations, update policies, match processing logic, and record retrieval, and to create custom components for the master index application, such as custom phonetic encoders, block pickers, or query builders. Oracle Java CAPS Master Patient Index includes several predefined custom plug-ins to generate and process alias names and to mask field values. You can create as many classes as you need to carry out the custom processes.

The following sections describe custom plug-ins that define custom processing. These are explained more fully in *Understanding Oracle Java CAPS Master Index Configuration Options (Repository)*.

- "Master Index Update Policy Plug-ins" on page 22 – Define custom processing logic to perform against the resulting record of a transaction before it is stored in the database.
- "Master Index Field Validation Plug-ins" on page 24 – Define validations to perform against specific fields, such as checking the local ID length and format.
- "Master Index Field Masking Plug-ins" on page 24 – Define how the values for sensitive fields are hidden on the Patient EDM from users who do not have permission to view them.
- "Master Index Match Processing Logic Plug-ins" on page 25 – Define custom logic based on predefined decision points for how records are matched during a transaction.
- "Master Index Custom Plug-in Exception Processing" on page 26 – Define how exceptions are handled by the custom plug-ins you create.

Master Index Update Policy Plug-ins

For the primary transactions performed by the master index application, you can define additional custom processing to perform against the record that results from a transaction. The policies you define are invoked by the Update Manager and are applied to the resulting records after they are processed by the survivor calculator. The modifications made to a record by an update policy determine how the record is stored in the database. By creating custom plug-ins, you can create additional Java classes to support the update policies you define.

Oracle Java CAPS Master Patient Index provides default custom plug-ins for each update policy to generate alias names when a patient's first, last, middle, or maiden names are modified. You can view and edit the Java code for each custom plug-in by expanding the Custom Plug-ins folder of the master index project and opening any of the Java files. Additional alias plug-ins are provided and are called by the custom update policies to process alias names after a transaction occurs.

Update policies are specified in the *UpdatePolicy* section of the Best Record file, and there are several different types. Each policy modifies an enterprise object (class `com.stc.eindex.objects.EnterpriseObject`) and must implement `com.stc.eindex.update.UpdatePolicy`, which contains one method, `applyUpdatePolicy`. The syntax is as follows:

```
public EnterpriseObject applyUpdatePolicy(EnterpriseObject before,
EnterpriseObject after)
```

This method throws two exceptions: `com.stc.eindex.master.UserException` and `com.stc.eindex.objects.exception.ObjectException`.

Enterprise Merge Policy

The enterprise merge policy defines additional processing to perform after two enterprise objects are merged. The processing defined in this policy acts against the surviving record of the merge. In the *EnterpriseMergePolicy* element in the Best Record file, enter the fully qualified name of this custom plug-in. The name of the default merge policy, `com.stc.eindex.update.impl.EnterpriseMergePolicy`, is already entered.

Enterprise Unmerge Policy

The enterprise unmerge policy defines additional processing to perform after an unmerge transaction occurs. The processing defined in this policy acts against the surviving record of the merge transaction that was unmerged. In the *EnterpriseUnmergePolicy* element of the Best Record file, enter the fully qualified name of this custom plug-in. The name of the default unmerge policy, `com.stc.eindex.update.impl.EnterpriseUnmergePolicy`, is already entered.

Enterprise Update Policy

The enterprise update policy defines additional processing to perform after a record is updated. In the *EnterpriseUpdatePolicy* element of the Best Record file, enter the fully qualified name of this custom plug-in. The name of the default update policy, `com.stc.eindex.update.impl.EnterpriseUpdatePolicy`, is already entered.

Enterprise Create Policy

The enterprise create policy defines additional processing to perform after a new record is inserted into the master index database. In the *EnterpriseCreatePolicy* element of the Best Record file, enter the fully qualified name of this custom plug-in. The name of the default create policy, `com.stc.eindex.update.impl.EnterpriseCreatePolicy`, is already entered.

System Merge Policy

The system merge policy defines additional processing to perform after two system objects are merged. The processing defined in this file acts against the surviving enterprise record of the merge (and not the system record). In the *SystemMergePolicy* element of the Best Record file, enter the fully qualified name of this custom plug-in. The name of the default merge policy, `com.stc.eindex.update.impl.SystemMergePolicy`, is already entered.

System Unmerge Policy

The system unmerge policy defines additional processing to perform after system objects are unmerged. The processing defined in this file acts against the surviving enterprise record of the system merge transaction that was unmerged. In the *SystemUnmergePolicy* element in the Best Record file, enter the fully qualified name of this custom plug-in. The name of the default merge policy, `com.stc.eindex.update.impl.SystemUnmergePolicy`, is already entered.

Undo Assumed Match Policy

The undo assumed match policy defines additional processing to perform after an assumed match transaction is reversed. In the *UndoAssumeMatchPolicy* element in the Best Record file, enter the fully qualified name of this custom plug-in.

Master Index Field Validation Plug-ins

You can define validations to be performed against certain fields before information is entered into the master index database. Once you create the custom plug-ins containing the validation logic, enter the name of the plug-in in the Field Validation file. Follow these guidelines when implementing custom field validators.

- The custom validation classes must implement `com.stc.eindex.objects.validation.ObjectValidator`.
- The exception thrown is `com.stc.eindex.objects.validation.exception.ValidationException`.

One default field validator, *validate-local-id*, is provided to validate system and local ID fields before processing data into the database. This is described in *Understanding Oracle Java CAPS Master Index Configuration Options (Repository)*.

Master Index Field Masking Plug-ins

There might be instances where you want to mask certain data in records from general users of the Patient Enterprise Data Manager and only allow access by administrators. To do this, you can create a custom plug-in that displays asterisks (or other symbols) in place of the field values on the Patient EDM. Once you define the custom plug-in, specify the name of the custom plug-in Java class in the *object-sensitive-plug-in-class* element of edm.xml).

Oracle Java CAPS Master Patient Index provides a default custom plug-in that defines field masking. The class is `com.stc.eindex.security.VIPObjectSensitivePlugIn`, which is specified in the *object-sensitive-plug-in-class* of edm.xml. This class contains logic that checks the value of the VIP Flag field. If the value is “V” (VIP) or “E” (Employee), the values are masked by a series of Xs for any field for which the *is-sensitive* element is set to **true** in the first section of edm.xml).

Master Index Match Processing Logic Plug-ins

You can implement custom plug-ins that customize the way the execute match methods process data into the master index application. When a record is entered into the master index system, match processing is performed by calling one of the following “execute match” functions from the `MasterController` class.

- `executeMatch`
- `executeMatchUpdate`
- `executeMatchDupRecalc`
- `executeMatchUpdateDupRecalc`
- `executeMatchGui` (this method is only called by the Patient EDM)

These methods contain standard logic for processing records through the master index database, weighting incoming records against existing records, and then using those weights to determine whether to insert a new record or update an existing record. In addition to configuring the match processing logic in the Threshold file, you can customize certain aspects of the processing logic using custom plug-ins that contain functions found in the `ExecuteMatchLogics` class.

Custom Match Processing Logic Methods

There are five decision branches where custom logic can be inserted. At specific points in match processing, the execute match methods look for the value of the methods listed below. For more information about the methods, see the Javadocs for the master index. These methods are contained in the `ExecuteMatchLogics` class in the package `com.stc.eindex.master`. For information about where the decision points are reached in the processing logic and how to change the logic, see *Understanding Oracle Java CAPS Master Index Processing (Repository)*. The following methods specify the custom logic.

- `bypassMatching` - Indicates whether to perform the match process on incoming records or to bypass the match process.
- `disallowAdd` - Indicates whether an incoming message can be inserted as a new record.
- `disallowUpdate` - Indicates whether an incoming record can update an existing record.
- `rejectAssumedMatch` - Indicates whether to accept or reject an assumed match of two records.
- `rejectUpdate` - Indicates whether to accept or reject an update to an existing record.

Custom Match Processing Logic Plug-in Requirements

The custom plug-ins you create to define custom execute match logic must extend the `ExecuteMatchLogics` class. In addition, the following classes must be imported into the custom plug-in.

- `com.stc.eindex.objects.SystemObject`
- `com.stc.eindex.objects.EnterpriseObject`
- `com.stc.eindex.objects.exception.ObjectException`
- `com.stc.eindex.master.ExecuteMatchLogics`
- `com.stc.eindex.master.CustomizationException`

Custom Match Processing Configuration

If you create a custom plug-in that defines custom processing logic for the execute match methods, you must specify those custom plug-ins in the Threshold file in the master index project. If you create a plug-in for customizing logic in the execute match methods used by the back-end, such as by Collaborations or Business Processes, specify the name of that class in the *logic-class* element. If you create a plug-in for the Patient EDM, specify the name of that class in the *logic-class-gui* element. For example:

```
<logic-class>com.stc.eindex.user.CustomCollaboration</logic-class>
<logic-class-gui>com.stc.eindex.user.CustomPatient EDM</logic-class-gui>
```

For more information about the Threshold file, see *Understanding Oracle Java CAPS Master Index Configuration Options (Repository)*.

Master Index Custom Plug-in Exception Processing

If a custom plug-in throws an exception of the class `ObjectException` or `SystemObjectException`, multiple stack traces are logged in the server log file, which can make operational management tasks more difficult. For cases where you do not want stack traces to be logged, configure your custom plug-ins to throw exceptions of the class `UserException` or one of its derived classes (`DataModifiedException` or `ValidationException`). This is useful for user errors on the Patient Enterprise Data Manager (Patient EDM). When one of these exceptions is thrown, no stack trace is entered in the log file but an error message still appears on the Patient EDM.

For more information about these exception classes, see the Javadocs for Oracle Java CAPS Master Index.

Custom Plug-Ins for Master Index Custom Components

Oracle Java CAPS Master Patient Index provides a flexible framework, allowing you to create custom Java classes to plug in to most master index application components. The following topics provide descriptions of some components for which you can create custom classes to use in your master index application.

- "Master Index Survivor Calculator Plug-ins" on page 27 – Define how the single best record (SBR) is generated.
- "Master Index Query Builder Plug-ins" on page 27 – Define how queries are performed against the master index database.
- "Master Index Block Picker Plug-ins" on page 28 – Define how the blocks of fields in a blocking query are selected during a query.
- "Master Index Pass Controller Plug-ins" on page 28 – Define how the master index application determines whether to perform additional match passes against a record.
- "Match Engine Plug-ins" on page 29 – Define logic that connects to a match engine other than the Oracle Java CAPS Match Engine.
- "Standardization Engine Plug-ins" on page 29 - Define logic that connects to a standardization engine other than that provided by the Oracle Java CAPS Match Engine.
- "Phonetic Encoders Plug-ins for a Master Index" on page 29 – Define logic that connects to phonetic encoders other than those provided.

Master Index Survivor Calculator Plug-ins

The survivor calculator determines which field values from the various system records will populate the SBR for the enterprise record. You can create a custom survivor calculator class that selects the surviving field values. Your custom class must implement the survivor calculator interface. Call `selectField` in `com.stc.eindex.survivor.SurvivorStrategyInterface` to return the SBR value for each field. For more information about the classes and methods to use, see the Javadocs for Oracle Java CAPS Master Index. The primary classes are contained in the `com.stc.eindex.survivor` package. Enter the fully qualified class path for the custom survivor calculator in the Best Record file.

Master Index Query Builder Plug-ins

The query builder defines the different types of queries that can be used in the master index application. You can implement custom queries using custom plug-ins. To create a new query builder, you must define a class that extends the base abstract `com.stc.eindex.querybuilder.QueryBuilder` and then specify that class in a *query-builder* element in the Candidate Select file. The exception thrown is `com.stc.eindex.querybuilder.QueryBuilderException`. The following methods must be implemented.

- `init` - This method receives the XML elements after the *config* element of the Candidate Select file so the query builder can read its custom configuration.
- `getApplicableQueryIds` - This method returns an array of string IDs indicating the query objects that can be generated given the available criteria. For example, in the blocking configuration, the unique ID of each block definition is the string that is returned by `getApplicableQueryIds`.
- `buildQueryObject` - This method constructs the query object based on one of the applicable query IDs provided as an input argument.

For more information about query-related Java classes, see the master index Javadocs.

Master Index Block Picker Plug-ins

The block picker chooses which block definition in the blocking query to use for the next matching pass. You can create a custom block picker class to select query blocks in a customized manner. If you create a custom block picker, specify the fully qualified name of this custom plug-in for the *block-picker* element of the Match Field file. Follow these guidelines when implementing a custom block picker.

- Implement the `com.stc.eindex.matching.BlockPicker` interface to select the blocks in the desired order.
- If none of the remaining blocks should be executed, throw a `NoBlockApplicableException` from the `pickBlock` method.

Master Index Pass Controller Plug-ins

The matching process can be executed in multiple stages. After a block is evaluated, the pass controller determines whether the results found are sufficient or if matching should continue by performing another match pass. If you create a custom pass controller, specify the name of the custom Pass Controller in the *pass-controller* element of the Match Field file. Follow these guidelines when implementing a custom pass controller.

- Implement the `com.stc.eindex.matching.PassController` interface to evaluate whether to do another pass or not.
- Return **`true`** from `evalAnotherPass` to specify that an additional pass be performed; return **`false`** to specify that no additional passes are performed.

Match Engine Plug-ins

You can define classes to connect to a custom match engine instead of the Oracle Java CAPS Match Engine. Specify the names of the custom classes you create in the *matcher-api* and *matcher-config* elements of the Match Field file. Follow these guidelines when implementing custom match engine classes.

- Implement the `com.stc.eindex.matching.MatcherAPI` interface to communicate with the match engine.
- Implement the `com.stc.eindex.matching.MatchEngineConfiguration` interface to retrieve any configuration values the match engine requires for initialization.

Standardization Engine Plug-ins

You can define classes to connect to a custom standardization engine instead of the Oracle Java CAPS Match Engine. Specify the names of the custom classes you create in the *standardizer-api* and *standardizer-config* elements of the Match Field file. Follow these guidelines when implementing custom standardization engine classes.

- Implement the `com.stc.eindex.matching.StandardizerAPI` interface to communicate with the standardization engine.
- Implement the `com.stc.eindex.matching.StandardizerEngineConfiguration` interface to retrieve any configuration values the standardization engine requires for initialization.

Phonetic Encoders Plug-ins for a Master Index

The master index application supports several phonetic encoders, and you can define custom classes to implement additional phonetic encoders if needed. Specify the names of the custom classes you create in the *encoder-implementation-class* element of the Match Field file. When creating a custom phonetic encoder class, implement the `com.stc.eindex.phonetic.PhoneticEncoder` interface.

Implementing Master Index Custom Plug-ins

Custom plug-ins are created in the `com.stc.eindex.user` package, and the name you specify for each plug-in is the name of the Java class created for the plug-in. You can create multiple source files that make up a plug-in. When you specify the custom plug-in in the configuration files, use the fully qualified class name of the class the master index application should call to carry out the custom processing. For example, if you create a custom plug-in named `MergePolicy`, the value to enter for the class in the Best Record file is `com.stc.eindex.user.MergePolicy`.

Follow these steps to implement a custom plug-in.

- "Creating Master Index Custom Plug-ins" on page 30
- "Building Master Index Custom Plug-ins" on page 31

Note – You can create custom plug-ins that define custom processing or that define custom components. For additional information about how to implement specific custom-plug ins, see the following topics:

- "Custom Plug-ins for Master Index Custom Transaction Processing" on page 22
- "Custom Plug-Ins for Master Index Custom Components" on page 27

Creating Master Index Custom Plug-ins

You create a custom plug-in by composing Java code into a custom plug-in file in NetBeans. When you create a custom plug-in, a file is automatically created for you with the first line already entered (`package com.stc.eindex.user;`).

▼ To Create Custom Plug-ins

1. **In the Projects window, expand the master index project and then expand the master index application.**

2. **For each source file that defines a custom plug-in, do the following:**

   a. **Right-click the Custom Plug-ins folder, and select New from the context menu that appears.**

   b. **Enter the name of the Java class you want to create and then click OK.**

      The custom plug-in file appears in the NetBeans editor.

   c. **Create the custom processing rules using Java code.**

   d. **Close and save the file.**

3. **Build the custom plug-in files, as described under "Building Master Index Custom Plug-ins" on page 31.**

4. **Specify the name of the class the master index application should call for the custom plug-in in the appropriate configuration file.**

Building Master Index Custom Plug-ins

In order for the custom plug-ins you create to become a part of the master index application, you must build the plug-in files. This compiles the Java code and incorporates it into the application files. Compiling errors for custom plug-ins are not written to a log. An error message trace appears on a console window alerting you to the errors that occurred.

▼ To Build Custom Plug-ins

1 **In the master index project, right-click the Custom Plug-ins folder.**

2 **Select Build from the context menu that appears.**

Note – If you modify a custom plug-in file after it has been checked in, be sure to check the file out before making any changes; otherwise, any changes will be lost when you try to save the file. Rebuild the plug-in after you save the changes, and then regenerate the application to incorporate the changes. Regenerating the application also rebuilds all custom plug-ins.

Generating the Master Index Application

Before you generate the application, review the configuration files and make any necessary modifications (see *Configuring Oracle Java CAPS Master Indexes (Repository)* for more information). Once all modifications to the configuration files are complete and any custom plug-ins are built, generate the master index application to update the application components. If you modify any of the configuration files, match and standardization engine files, or custom plug-ins after you generate the application, you need to regenerate the application to update the custom components.

Note – If any errors occur while compiling the application, they appear in the output panel in the bottom of the window. This window also displays the status of the generate process.

▼ To Generate the Application for the First Time

1 **Save any configuration changes to the master index project.**

2 **Right-click the master index application in the Projects window.**

3 **Select Generate.**

The project components are generated. This might take a few minutes.

4 **On the NetBeans toolbar, click Save.**

5 **If you are using command line reports, do the following:**

a. **Export the generated `Person_stc_eindex_client.jar` and `Person_stc_eindex_util.jar` files to the `/lib` subdirectory in the reports home directory.**

b. **In the `/lib` subdirectory, rename `Person_stc_eindex_client.jar` to `stc_eindex_client.jar` and rename `Person_stc_eindex_util.jar` to `stc_eindex_util.jar`.**

▼ To Regenerate the Application

1 **Save any configuration changes to the master index project.**

2 **Right-click the master index application in the Projects window.**

3 **Select Generate, and then click Yes on the dialog box that appears.**

The application components are regenerated. This might take a few minutes. The output panel displays the status and any errors that occur.

4 **On the NetBeans toolbar, click Save.**

5 **If there are any client projects with Collaborations that reference the master index server project, refresh those Collaborations.**

6 **If there were any changes to how incoming data is processed (such as changes to the output OTD), re-import the `.jar` files.**

a. **Open the Collaboration in the Collaboration Editor and click Import JAR Files.**

The Add/Remove Jar Files dialog box appears.

b. **For each master index application `.jar` file, highlight the filename on the Add/Remove Jar Files dialog box, and then click Remove.**

c. **For each file to re-import, click Add, double-click the master index server project, select the `.jar` file, and then click Import.**

7 **If you are using command line reports, do the following:**

a. **Export the regenerated`Person_stc_eindex_client.jar` and `Person_stc_eindex_util.jar` files to the `/lib` subdirectory in the reports home directory.**

b. In the `/lib` subdirectory, rename `Person_stc_eindex_client.jar` to `stc_eindex_client.jar` and rename `Person_stc_eindex_util.jar` to `stc_eindex_util.jar`.

Master Index Database Scripts and Design

Before you create the master index database, familiarize yourself with the database scripts and the database structure. Analyze your database requirements, including hardware considerations, startup data, indexing needs, performance, and so on.

The following topics provide information to help you in your analysis.

- "Master Index Database Scripts" on page 33
- "Master Index Database Requirements" on page 33
- "Master Index Database Structure" on page 35
- "Designing the Master Index Database" on page 36

- "Master Index Database Table Description for sbyn_systems" on page 42
- "Master Index Database Table Description for sbyn_user_code" on page 46

Master Index Database Scripts

The database scripts include scripts for defining code lists and external systems, for creating tables and indexes, and for dropping tables and indexes. These scripts appear under the Database Script node of the master index project, and are named Systems, Code List, Create User Indexes, Drop User Indexes, Create User Code Data, Create Person Database, and Drop Person Database. You can modify these scripts as needed to customize the tables, indexes, startup data, and database distribution. You can also create new database scripts if needed.

Master Index Database Requirements

When configuring the master index database, there are several factors to consider, including basic software requirements, operating systems, disk space, and so on. This section provides a summary of requirements for the database. For more detailed information about designing and implementing the database, refer to the appropriate Oracle or SQL Server documentation. The person responsible for the database configuration should be an Oracle or SQL Server database administrator familiar with the master index database and with your data processing requirements.

Database Platform Requirements

The master index database can be run on SQL Server or on Oracle. For specific version information, see "Java CAPS 6.3 Components and Supported External Systems" in *Planning for*

Oracle Java CAPS 6.3 Installation . You must have this software installed before beginning the database installation. Make sure you also install the latest patches for the version you are using.

Operating System Requirements

The database can be installed on any operating system supported by the database platform you are using. See the Oracle or SQL Server documentation for more information.

Hardware Requirements

This section describes the minimum recommended hardware configuration for a database installation. These requirements are based on the minimum requirements recommended by Oracle and SQL Server for a typical installation. Depending on the size of the database and expected volume, you should increase these recommendations as needed. See your Oracle or SQL Server documentation for more information and for supported operating systems.

Oracle Database

For a Windows database server, the following configuration is recommended as a minimal installation:

- Windows 2000 SP3 or later, Windows XP SP2, or Windows Server 2003
- Pentium 266 or later
- 1 GB RAM (increase this based on the number of users, connections to the database, and volume)
- Virtual memory should be double the amount of RAM
- 3 GB disk space plus an additional 2 KB for each system record to be stored in the database (note that this is a conservative estimate per system record, assuming that most records do not contain complete data). This depends on the Oracle environment you install. Enterprise Edition can take up to 5 GB.
- 256-color video

For a UNIX database server, the following configuration is recommended as a minimal installation:

- 256 MB RAM (increase this based on the number of users and connections to the database)
- Swap space should be a minimum of twice the amount of RAM
- 2 GB disk space plus an additional 2 KB for each system record to be stored in the database (note that this is a conservative estimate per system record, assuming that most records do not contain complete data).

Note – Disk space recommendations do not take into account the volume and processing requirements or the number of users. These are minimal requirements to install a generic database. At a minimum, the empty database and the database software will require 2.5 GB of disk space.

Microsoft SQL Server

The following configuration is recommended as a minimal installation for a SQL Server database.

- Pentium III-compatible processor or higher
- 512 MB RAM as a minimum; at least 1 GB is recommended (increase this based on the number of users, connections to the database, and volume)
- 3 GB disk space plus an additional 2 KB for each system record to be stored in the database (note that this is a conservative estimate per system record, assuming that most records do not contain complete data). This depends on the SQL Server environment you install.
- VGA or higher resolution

Note – Disk space recommendations do not take into account the volume and processing requirements or the number of users. These are minimal requirements to install a generic database. At a minimum, the empty database and the database software will require 1.6 GB of disk space.

Master Index Database Structure

The master index database contains some common tables that are created for all implementations and some that are customized for each implementation. The common tables include standard Oracle or SQL Server tables and supporting tables, such as sbyn_seq_table, sbyn_common_header, and sbyn_common_detail. These tables do not store information about the enterprise object structure you defined. The names of the tables that store information about the enterprise object are customized based on the object structure.

Two tables store information about the primary, or parent, object you defined: sbyn_*parent_object* and sbyn_*parent_object*sbr, where *parent_object* is the name you specified for the parent object in the object structure. The sbyn_*parent_object* table stores parent object data from each local system and the sbyn_*parent_object*sbr table stores the parent object data contained in the SBRs. Similar tables are created for each child object you defined in the object structure.

For a complete description of the database tables, see *Understanding Oracle Java CAPS Master Index Configuration Options (Repository)*.

Designing the Master Index Database

In designing the database, there are several factors to consider, such as the volume of data stored in the database and the number of transactions processed by the database daily. The master index database should be created in its own tablespaces. The following sections describe some of the analyses to perform along with considerations to take into account when designing the database.

Designing for Performance Optimization

The Oracle and SQL Server installation guides provide detailed information about installing the database software for optimal performance. Both database platforms include guides containing information about monitoring and fine-tuning your database, including tuning memory, swap space, I/O, CPU usage, block and file size, and so on. You should be familiar with these concepts prior to creating the database.

Data Structure Analysis

Before defining the object structure, you analyzed the structure of the legacy data to help you define the object structure and the attributes of each field. You can use this data analysis to determine the amount of data that will be stored in the database, which will help you size the master index database and decide how to best distribute the database. Knowing the volume of existing data plus the expected daily transaction volume will help you plan the requirements of the database server, such as networking needs, disk space, memory, swap space, and so on.

The data structure analysis also helps you determine the processing codes and descriptions to enter in the common tables (described below), and should help you determine any default values that have been entered into certain fields that could skew the matching probability weights.

Common Table Data

Common table data analysis involves gathering information about the abbreviations used for specific data elements in each sending system, such as system codes and codes for certain attributes of the patients in your database, such as language, race, and marital status codes. The processing codes and their descriptions are stored in a set of database tables known as common maintenance tables. The master index project includes a script to help you load the processing codes into the database.

When an enterprise object appears on the Patient EDM, the master index application translates the processing codes defined in the common tables into their descriptions so the user is not required to decipher each code. The data elements stored in the common maintenance tables are also used to populate the drop-down lists that appear for certain fields in the Patient EDM. Users can select from these options to populate the associated fields.

User Code Data

User code data analysis involves gathering information about the abbreviations used for specific data elements in each sending system for a field whose format or possible values are constrained by a separate field. For example, if you store credit card information, you might have a drop-down list in the Credit Card field for each credit card type. The format of the field that stores the credit card number is dependent on the type of credit card you select. You could also use user code data to validate cities with postal codes. The abbreviations and related constraint information are stored in the sbyn_user_code table. A sample script, Create User Code Data, is provided to help you insert data into this table.

Database Considerations

When you create the master index database, you need to consider several factors, such as sizing, distribution, indexes, and extents. By default, all of the master index database tables for an Oracle database are installed in the system tablespace. You should install the master index tables in different tablespaces, depending on the original size and expected volume of the database. For SQL Server, the master index tables belong to “dbo” by default.

Database Sizing

To begin the database installation, you first create an Oracle or SQL Server database instance using the provided configuration tools. Use the tools provided by Oracle or Microsoft to define the tablespace and extent sizing for the database.

Database Distribution

When you create the database instance, you can define the distribution of your system tables, data tables, rollback logs, dump files, control files, and so on. Use internal policies regarding relational database distribution to determine how to best distribute your master index database.

Database Indexes

By default, indexes are defined for the following tables: sbyn_appl, sbyn_common_header, sbyn_common_detail, sbyn_enterprise, sbyn_transaction, sbyn_assumedmatch, sbyn_potentialduplicates, sbyn_audit, and sbyn_merge. Index scripts are also created for indexing the fields included in the default blocking query in the Candidate Select file. You can create additional indexes against the database to optimize the searching and matching processes. At a minimum, it is recommended that all combinations of fields used for blocking or matching be indexed. For each query block defined in the blocking query, create an index containing the fields in that block.

The following indexes are automatically created to improve performance when running large reports from the command line or Patient EDM.

```
CREATE INDEX SBYN_POTENTIALDUPLICATES3 ON SBYN_POTENTIALDUPLICATES
(TRANSACTIONNUMBER ASC);
```

```
CREATE INDEX SBYN_ASSUMEDMATCH2 ON SBYN_ASSUMEDMATCH (TRANSACTIONNUMBER ASC);

CREATE INDEX SBYN_TRANSACTION4 on SBYN_TRANSACTION (EUID2 ASC, TIMESTAMP ASC);

CREATE INDEX SBYN_TRANSACTION3 on SBYN_TRANSACTION (TIMESTAMP ASC,
TRANSACTIONNUMBER ASC);
```

Note – To improve performance, these four indexes should be dropped prior to performing an initial load or batch load of data. They can be recreated once the load is complete if you are running the provided reports.

Creating the Master Index Database

Once you have customized the configuration files and generated the master index application, you can create the master index database. Before you begin, make sure you have Oracle or SQL Server installed on the database server.

During this process you can define custom startup data, such as code lists and source systems. Startup data is predefined in the Code Lists and Systems SQL scripts, and you can customize the statements to insert the data relevant to your object structure. The code lists you define are used to translate processing codes from incoming messages into descriptions for the Patient EDM fields and to create drop-down lists for Patient EDM fields. System information is required in order to add records to the master index application.

Follow these steps to create the database.

- “Step 1: Analyze the Master Index Database Requirements” on page 39
- “Step 2: Create a Master Index Database and User” on page 39
- “Step 3: Define Master Index Database Indexes” on page 40
- “Step 4: Define Master Index External Systems” on page 41
- “Step 5: Define Master Index Code Lists” on page 43
- “Step 6: Define Master Index User Code Lists” on page 45
- “Step 7: Create Custom Master Index Database Scripts” on page 47
- “Step 8: Create the Master Index Database Structure” on page 48
- “Step 9: Specify a Starting EUID for a Master Index” on page 49

You can also delete the database for testing purposes using the supplied script. See “Deleting Master Index Database Tables and Indexes” on page 49 for more information.

Step 1: Analyze the Master Index Database Requirements

Before you begin to create the master index database, perform an analysis of the structure of the legacy data to be stored in the database and determine the amount of data that will be processed daily. During the analysis, be sure to define the processing codes that need to be stored in the common maintenance tables and the systems that will share data with the master index application. You should also know the length and format of the local IDs assigned by each system.

A database administrator who is familiar with your data and processing requirements should perform this task. After this task is complete, continue to "Step 2: Create a Master Index Database and User" on page 39.

For additional information and guidelines about how to set up your database, see "Master Index Database Scripts and Design" on page 33.

Step 2: Create a Master Index Database and User

Before beginning this step, complete "Step 1: Analyze the Master Index Database Requirements" on page 39. After you create the database instance and user, continue to "Step 3: Define Master Index Database Indexes" on page 40 if you want to define additional database indexes; otherwise skip to "Step 4: Define Master Index External Systems" on page 41.

For this step you need to create a database in which the master index database instance will be created. Use your Oracle or SQL Server tools to create the database. Using these tools, you define tablespaces, including their sizes and locations; extents; and dump file, log file, and rollback file sizes and locations. Make sure these issues have been thoroughly analyzed and designed before creating the database.

Once you create the database, you can use standard SQL to create the master index application user for the database. The user you create in this step will be used to create the database structure and to connect to the database through the Patient EDM and through the application server.

For Oracle, assign the user to the "connect" and "resource" roles for the master index tablespaces. For example:

```
create user username identified by password;
grant connect, resource to username;
commit;
```

where *username* is the login ID of the administrator user and *password* is the login password of the administrator user.

For SQL Server, assign this user to the "db_owner" role. You need to create the server login, create the user, and then assign the user to the role. For example:

```
CREATE LOGIN loginname WITH PASSWORD = 'password', DEFAULT_DATABASE = database;
CREATE USER username FOR LOGIN loginname; USE database;
EXECUTE sp_addrolemember 'db_owner', ' username' GO
```

where *loginname* is the login ID for the administrator user, *password* is the login password, *database* is the database name, and *username* is the owner of the database tables created in the master index database.

Note – SQL Server allows Windows Authentication, where only a user name is required. Java CAPS products require full authentication, including both a user name and password. You need to create a database user specifically for the master index application.

If you prefer to assign individual permissions to the user instead of roles, the following permissions are needed.

- alter any index
- alter any procedure
- alter any table
- alter any trigger
- create any index
- create procedure
- create session
- create table
- create trigger
- create view
- delete any table
- drop any index
- drop any procedure
- drop any table
- drop any trigger
- drop any view
- insert any table
- select any table
- update any table

Step 3: Define Master Index Database Indexes

To optimize data processing in the master index application, you can define additional indexes for the database tables that store object data. Best practice is to define indexes for each field used for searching, blocking, or matching. You can define these indexes in the Create User Indexes file or create a new script. Before you begin this step, complete "Step 2: Create a Master Index Database and User" on page 39.

▼ To Define an Index

1. **Under the master index project in the Projects window, expand Database Scripts and then open the Create User Indexes file in the NetBeans editor.**
2. **Do any of the following:**
   - **Remove an existing index definition (not recommended).**
   - **Create new index definitions for the required fields.**
   - **Modify an existing index definition.**
3. **Save and close the file.**
4. **Continue to "Step 4: Define Master Index External Systems" on page 41.**

Step 4: Define Master Index External Systems

The Systems file in the master index project defines one default source system for the master index application. You can define additional systems as needed or delete the default system. Before you begin this step, complete "Step 2: Create a Master Index Database and User" on page 39 and, optionally, "Step 3: Define Master Index Database Indexes" on page 40.

▼ To Define an External System

1. **Under the master index project in the Projects window, expand Database Scripts and then open the Systems file in the NetBeans editor.**
2. **For each system, create an INSERT statement using the column descriptions in "Master Index Database Table Description for sbyn_systems" on page 42 to define the VALUES clause.**

   For example:

   ```
   INSERT into sbyn_systems (systemcode, description, status, id_length,
   format, input_mask, value_mask, create_date, create_userid) VALUES ('ARS',
   'Automated Registration System', 'A', 10, '[0-9]{10}', 'DDD-DDD-DDDD',
   'DDD^DDD^DDDD', sysdate, "admin');
   ```
3. **Delete any default INSERT statements you do not need.**
4. **Save and close the file.**
5. **Continue to "Step 5: Define Master Index Code Lists" on page 43.**

Master Index Database Table Description for sbyn_systems

The following table lists and describes the columns in the sbyn_systems database table so you can create SQL statements to insert source system information into the master index database.

Column	Description
systemcode	The unique processing code of the system. This field accepts any of the following characters: ■ ! _ ~ ( ) { } + \Q # $ % & : ; - / ■ a-z ■ A-Z ■ 0-9 ■ þ ÿ Þ ß 'à-ö ø-ý À-Ö Ø-Ý
description	A brief description of the system, or the system name. **Tip –** It is best to keep this short if possible; these values appear in the tree views on the Patient Enterprise Data Manager and can cause the box containing the tree views to increase in width to accommodate all characters.
status	The status of the system in the master index system. Specify **A** for active or **D** for deactivated.
id_length	The length of the local identifiers assigned by the system. This length does not include any additional characters added by the input mask (see below). **Note –** The default maximum length of the LID database columns is 25. If the system generates longer local IDs, be sure to increase the length of all LID columns in the database.
format	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. Note that the pattern specified here might be further restricted by the input mask described below.

Column	Description
input_mask	A mask used by the Patient EDM to add punctuation to the local ID. For example, you can add an input mask to display the local IDs with hyphens or constant characters. To define an input mask, enter a character type for each character in the field and place any necessary punctuation between the types. For example, to insert a hyphen after the second and fifth characters in an 8-digit ID, the input mask would be **DD-DDD-DDD**. The following character types can be used; any other characters are treated as constants. ■ **D** - indicates a numeric character. ■ **L** - indicates an alphabetic character. ■ **A** - indicates an alphanumeric character. If you use an input mask, you should also define a value mask to remove the punctuation from the stored value.
value_mask	A mask used to strip any extra characters that were added by the input mask. This mask ensures that data is stored in the database in the correct format. To specify a value mask, type the same value entered for the input mask, but type an “x” in place of each punctuation mark. Using the 8-digit input mask described above, you would specify a value mask of **DDxDDDxDDD**. This strips the hyphens before storing the ID in the database.
create_date	The date the system information was inserted into the database. You can specify “sysdate” for this column, or use the variables defined in of the sample script.
create_userid	The logon ID of the user who inserted the system information into the database. You can enter the logon ID or use the variables defined in of the sample script.

Step 5: Define Master Index Code Lists

You only need to perform this step if you defined any fields in the object structure to have a code module. The SQL script for entering processing codes and descriptions into the database is written in PL/SQL. The Code List file contains a stanza for each type of common table data element for which you use processing codes. You can create additional common table data types and additional common table data elements. This script inserts data into two tables: sbyn_common_header, which lists the types of common table data, and sbyn_common_detail, which lists each common table data element. Before you begin this step, complete “Step 4: Define Master Index External Systems” on page 41.

Note – The codes you specify in this file can be no longer than eight characters (the codes are the second value in the value list for each common table data type and data element).

▼ To Customize Common Table Data for Oracle

1. **Under the master index project in the Projects window, expand Database Scripts and then open the Code Lists file in the NetBeans editor.**

2. **Scroll to the following line.**

   ```
   codes tCodeList := tCodeList(
   ```

   The statements following this line can be customized.

3. **Add or modify the elements of each stanza as needed. A sample stanza is shown below.**

   ```
   -- **** PHONTYPE ****
   tCode('L', 'PHONTYPE', 'TELEPHONE TYPE'),
   tCode('V', 'H', 'HOME'),
   tCode('V', 'M', 'MOBILE'),
   tCode('V', 'F', 'FAX'),
   tCode('V', 'O', 'OFFICE'),
   ```

   Note – Do not modify the “L” or “V” in each row. These characters define whether the information is inserted into the sbyn_common_header (L) or sbyn_common_detail (V) table. Following the table indicator is the processing code, and the final item in each row is a description. The descriptions appear on the Patient EDM and should be kept short.

   ---

4. **In the last code module stanza, make sure each line except the last contains a comma at the end.**

   For example:

   ```
   -- **** ADDRTYPE ****
   tCode('L', 'ADDRTYPE', 'ADDRESS TYPE'),
   tCode('V', 'H', 'HOME'),
   tCode('V', 'B', 'BUSINESS'),
   tCode('V', 'M', 'MAILING')
   ```

5. **Save and close the file.**

6. **Do one of the following:**

   - **If you need to define user code lists, continue to “Step 6: Define Master Index User Code Lists” on page 45.**
   - **If you need to create a custom database script, skip to “Step 7: Create Custom Master Index Database Scripts” on page 47.**
   - **If you are ready to create the master index database structure, skip to “Step 8: Create the Master Index Database Structure” on page 48.**

▼ To Customize Common Table Data for SQL Server

1 Under the master index project in the Projects window, expand Database Scripts and then open the Code Lists file in the NetBeans editor.

2 Scroll to the following line.

```
begin
```

The statements following this line can be customized.

3 Add or modify the elements of each stanza as needed. A sample stanza is shown below.

```
-- **** PHONTYPE ****
insert into @codelist values('L', 'PHONTYPE', 'TELEPHONE TYPE'),
insert into @codelist values('V', 'H', 'HOME'),
insert into @codelist values('V', 'M', 'MOBILE'),
insert into @codelist values('V', 'F', 'FAX'),
insert into @codelist values('V', 'O', 'OFFICE'),
```

Note – Do not modify the "L" or "V" in each row. These characters define whether the information is inserted into the sbyn_common_header or sbyn_common_detail table. Following the table indicator is the processing code, and the final item in each row is the description that appears on the Patient EDM. The descriptions should be kept short.

---

4 Save and close the file.

5 Do one of the following:

- **To define user code lists, continue to "Step 6: Define Master Index User Code Lists" on page 45.**
- **To create a custom database script, skip to "Step 7: Create Custom Master Index Database Scripts" on page 47.**
- **To create the master index database structure, skip to "Step 8: Create the Master Index Database Structure" on page 48.**

Step 6: Define Master Index User Code Lists

If you specified a value for the`constraint-by` and `user-code` elements of a field, you must define the user code values for those fields. Below is a sample insert statement for the sbyn_user_code table.Oracle Java CAPS Master Patient Index provides this sample in a database script you can customize.

```
insert into sbyn_user_code (code_list, code, descr, format, input_mask,
value_mask) values ('AUXIDDEF', 'CC', 'CREDIT CARD', '[0-9]{16}',
'DDDD-DDDD-DDDD-DDDD', 'DDDD^DDDD^DDDD^DDDD');
```

To learn more about the sbyn_user_code table, see "Master Index Database Table Description for sbyn_user_code" on page 46. Complete "Step 5: Define Master Index Code Lists" on page 43 before beginning this step.

▼ To Define a User Code List

1. **In the Projects window, expand the Database node and then double-click the Create User Code Data file.**

   The file opens in the NetBeans editor.

2. **Use the above sample to define a value for the user code drop-down list and the required format for the dependent fields.**

3. **Repeat the above step for each drop-down list value and type (for example you might have one list for credit cards and another for postal codes and their corresponding cities).**

4. **Save and close the file.**

5. **Continue to "Step 7: Create Custom Master Index Database Scripts" on page 47 or to "Step 8: Create the Master Index Database Structure" on page 48 if you do not need to create any custom database scripts.**

Master Index Database Table Description for sbyn_user_code

The following table lists and describes the columns in the sbyn_user_code database table so you can create SQL statements to insert user code data into the master index database.

Column Name	Description
code_list	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	The processing code of each user code element.
description	A brief description or name for the user code element. This is the value that appears in the drop-down list.

Column Name	Description
format	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. Note that the pattern might be further restricted by the value of the input mask described below.
input-mask	A mask used by the Patient 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. If you use an input mask, you should also use a value mask to strip the punctuation for database storage (see below). The following character types can be used. ■ **D** – Numeric character ■ **L** – Alphabetic character ■ **A** – Alphanumeric character
value-mask	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.

Step 7: Create Custom Master Index Database Scripts

You can insert additional information into the database by creating a custom script under the Database Script node. For information about the structure of the master index database, see *Understanding Oracle Java CAPS Master Index Processing (Repository)*.

▼ To Create a Custom Script

1 **In the master index project, right-click the Database Script node.**

2 **Click New.**

3 **Enter the name of the script, and then click OK.**

The new script appears under the Database Script node.

4 **Double-click the new script.**

The text editor appears.

5 **In the text editor, create the SQL script to insert the custom data.**

6 **Save and close the file.**

7 **Continue to "Step 8: Create the Master Index Database Structure" on page 48.**

Step 8: Create the Master Index Database Structure

After you create the database instance and customize the database scripts, you can create the master index tables and insert the custom data.

▼ To Create the Database Structure

1 **In the master index project, right-click the Database Script node, and then select Properties from the Database Script context menu.**

The Properties of Database Script dialog box appears.

2 **Do one of the following:**

- **For Oracle: In the Database Server field, change** *<host>* **to the database server address and change** *<SID>* **to the SID name of the database you created in "Step 2: Create a Master Index Database and User" on page 39. If Oracle is not listening on the default port, 1521, enter the correct port number. You can use "localhost" as the hostname if the database resides on the same machine as NetBeans.**

  For example:

  ```
  jdbc:oracle:thin:@localhost:1521:IndexDB
  ```

- **For SQL Server: In the Database Server field, change the URL to the following:**

  `jdbc:sqlserver://`*server*`:`*port*`;databaseName=`*database*

  where *server* is the address of the database server, *port* is the port number on which SQL Server is listening, and *database* is the name of the database. You can use "localhost" for the hostname if the database resides on the same machine as NetBeans.

3 **In the Password field, enter the password of the administrator user you created when you created the database (creating an administrator user is described under "Step 2: Create a Master Index Database and User" on page 39).**

4 **In the User field, enter the administrator user's logon ID.**

Note – Make sure you enter the database logon credentials for the administrator user you created. You cannot use the logon credentials for the default system user (the database tables will be created, but the master index application will not function correctly).

---

5 **Close the dialog box by clicking the "X" icon in the upper right corner of the dialog box.**

6 **Right-click Create Person Database, and then select Run. On the confirmation dialog box, click OK.**

7 **For each additional script to run against the database, right-click the name of the script, and then select Run. On the confirmation dialog box, click OK.**

Step 9: Specify a Starting EUID for a Master Index

By default, the EUIDs assigned by the master index application start with "0", with padded zeroes added to the left to make the EUID number the correct length (for more information, see *Understanding Oracle Java CAPS Master Index Configuration Options (Repository)*). You can modify this numbering format by changing the value of the seq_name column of the sbyn_seq_table database table where the sequence name is "EUID". For example:

```
update sbyn_seq_table set seq_count=1000000001 where seq_name='EUID';
```

Deleting Master Index Database Tables and Indexes

Scripts are provided to drop the default database tables and indexes created in "Step 8: Create the Master Index Database Structure" on page 48. This is useful while testing the master index application implementation.

▼ To Delete Database Tables and Indexes

1 **To drop the indexes created by the Create User Indexes script:**

   a. **Right-click Drop User Indexes.**

   b. **Select Run.**

   c. **On the confirmation dialog box, click OK.**

2 **To drop the database tables:**

   a. **Right-click Drop Person Database.**

   b. **Select Run.**

   c. **On the confirmation dialog box, click OK.**

3 **If the database is running on the Oracle 10*g* platform, open a SQL prompt and run the following command.**

```
PURGE RECYCLEBIN;
```

Defining a Database Connection Pool Through the Application Server

If you are using a database Adapter, you configure the database connection pool in the database external system in the Environment and do not need to perform this step. If the application is running on GlassFish Application Server and is not using a database Adapter to connect to the database, you need to create two JDBC connection pools and resources using the Admin Console.

To set up the connection pools, you create the connection pools and then define a JDBC resource for each pool. This section provides general instructions for setting up the connection pools. For more information about the procedures in this section, see the online help provided with the GlassFish Admin Console.

Perform the following steps to define database connectivity through the application server:

- "Step 1: Add the Oracle Driver to the Application Server" on page 50
- "Step 2: Create the JDBC Connection Pools" on page 50
- "Step 3: Create the JDBC Resources" on page 52

Step 1: Add the Oracle Driver to the Application Server

If you are using an Oracle database, you need to manually install or copy the database driver to the application server environment. If you are using a SQL Server database, you can skip this step.

You can either install the Oracle driver on the application server or copy the `ojdbc14.jar` file from your Oracle client installation (*Oracle_client*`\jdbc\lib`) to *app_server_home*`\lib`. To install the driver, see the documentation for the GlassFish Application Server.

Once the driver is installed or copied, continue to "Step 2: Create the JDBC Connection Pools" on page 50.

Step 2: Create the JDBC Connection Pools

The JDBC connection pools provide connections for the master index database. Before proceeding, make sure you have the relevant information about the master index database (such as the database name, URL, and administrator login credentials).

▼ To Create the JDBC Connection Pools

Before You Begin If you are using an Oracle database, be sure to add the database driver to the application server environment, as described in "Step 1: Add the Oracle Driver to the Application Server" on page 50.

1 **Log in to the GlassFish Application Server Admin Console.**

You can access the console from the Services window in NetBeans.

2 **In the left portion of the Admin Console, expand Resources, expand JDBC, and then select Connection Pools.**

3 **On the Create Connection Pool page, click New.**

4 **In the Name field, enter a name for the connection pool.**

5 **In the Resource Type field, select the appropriate Java class.**

Note – You can use either `javax.sql.DataSource` or `javax.sql.ConnectionPoolDataSource`.

---

6 **In the Database Vendor field, select the database platform of the master index database.**

7 **Click Next.**

8 **In the DataSource Classname field, enter the Java class for the data source, or accept the default value if it is provided.**

9 **Modify the Pool Settings properties according to your business practices.**

10 **Modify the Connection Validation properties according to your business practices.**

11 **Modify the Transaction properties if necessary.**

12 **In the additional properties section, enter the values for the master index database. Be sure to enter the following information at a minimum (you might need to create some of these properties):**

- URL – The URL that points to the database. The syntax of the URL is:
  - For Oracle, `jdbc:oracle:thin:@`*host*`:`*port*`:`*database_name*.
  - For SQL Server, `jdbc:sqlserver://`*server*`:`*port*`;databaseName=`*database*
- user – The login ID for the user you created in "Step 2: Create a Master Index Database and User" on page 39.
- password – The password for the above user.
- ImplicitCachingEnabled – An indicator of whether implicit statement caching is enabled. Set this property to **`true`**.
- MaxStatements – The maximum number of statements in the cache. Set this property to **`1000`**.

13 **Repeat the above steps to create a connection pool with a different name for the sequence manager.**

14 **Continue to "Step 3: Create the JDBC Resources" on page 52.**

Step 3: Create the JDBC Resources

A JDBC resource (also known as a data source) gives the master index application the ability to connect to the database. You need to create one for each connection pool created earlier.

▼ To Create a JDBC Resource

Before You Begin Create the JDBC connection pool, as described in "Step 2: Create the JDBC Connection Pools" on page 50.

1 **In the left portion of the Admin Console, expand Resources, expand JDBC, and then select JDBC Resources.**

2 **On the Create JDBC Resource page, click New.**

3 **In the JNDI Name field, enter a unique name for the JDBC resource.**

The name must be in the form **`jdbc/`***application_name***`DataSource`**, where *application_name* is the name of the master index application. For example, **`jdbc/PersonDataSource`**.

4 **In the Pool Name field, enter the name of the first JDBC connection pool you created in "Step 2: Create the JDBC Connection Pools" on page 50.**

5 **(Optional) In the Description field, enter a brief description of the resource.**

6 **In the Status field, select the Enabled check box.**

7 **Click OK.**

8 **Repeat the above steps to create another JDBC resource for the sequence manager using these guidelines:**

- For the connection pool, select the second connection pool you created in "Step 2: Create the JDBC Connection Pools" on page 50.
- The name of the resource must be in the form **`jdbc/`***application_name***`SequenceDataSource`**, where *application_name* is the name of the master index application. For example, **`jdbc/PersonSequenceDataSource`**.