Developing Oracle® Java CAPS Master Indexes (Repository)

Part No: 821–2557
March 2011

ORACLE

120126@25097

Contents

Developing Oracle Java CAPS Master Indexes (Repository)

The topics listed here provide procedures, conceptual information, and reference information for using Oracle Java CAPS Master Index to design and create a master index application. These topics provide information to 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 Oracle Java CAPS Master Index (Repository) documentation, see "Related Topics" on page 8.

Note that Java CAPS includes two versions of Oracle Java CAPS Master Index. Oracle Java CAPS Master Index (Repository) is installed in the Java CAPS repository and provides all the functionality of previous versions in the new Java CAPS environment. Oracle Java CAPS Master Index is a service-enabled version of the master index that is installed directly into NetBeans. It includes all of the features of Oracle Java CAPS Master Index (Repository) plus several new features, like data analysis, data cleansing, data loading, and an improved Data Manager GUI. Both products are components of the Master Data Management (MDM) Suite. This document relates to Oracle Java CAPS Master Index (Repository) only.

What You Need to Know

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

What You Need to Do

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

- "Creating a Master Index Application (Repository)" on page 26
- "Implementing Master Index Custom Plug-ins (Repository)" on page 53
- "Generating the Master Index Application (Repository)" on page 55
- "Creating the Master Index Database (Repository)" on page 62
- "Deleting Master Index Database Tables (Repository)" on page 74
- "Defining a Database Connection Pool Through the Application Server" on page 74

More Information

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

- "Master Index Wizard Field Properties and Name Restrictions (Repository)" on page 41
- "Master Index Database Table Description for sbyn_systems (Repository)" on page 65
- "Master Index Database Table Description for sbyn_user_code (Repository)" on page 71

Related Topics

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

- *Developing Oracle Java CAPS Master Indexes (Repository)*
- *Working With the EDM for Oracle Java CAPS Master 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 Index Overview

The following topics provide information about Oracle Java CAPS Master Index, how it is used to create a master index application, and the master index applications you create with Oracle Java CAPS Master Index. It also includes a description of the files stored in the Java CAPS repository, the XML files that define the structure and configuration of the master index environment, and the runtime components.

About Oracle Java CAPS Master Index

Oracle Java CAPS Master Index provides a flexible framework to allow you to create matching and indexing applications called enterprise-wide master index applications. It is an application building tool to help you design, configure, and create a master index application that will uniquely identify and cross-reference the business objects stored in your system databases. Business objects can be any type of entity for which you store information, such as customers, patients, vendors, businesses, hardware parts, and so on. In Oracle Java CAPS Master Index, you define the data structure of the business objects to be stored and cross-referenced as well as the logic that determines how data is updated, standardized, weighted, and matched in the master index application.

The structure and logic you define is located in a group of XML configuration files that you create using the Wizard Editor for Master Index. These files are created within the context of a Java CAPS project, and can be further customized using the XML editor provided in the NetBeans IDE.

Oracle Java CAPS Master Index Features

Oracle Java CAPS Master Index provides features and functions to allow you to create and configure an enterprise-wide master index application for any type of data. The primary function of Oracle Java CAPS Master Index is to automate the creation of a highly configurable master index application . Oracle Java CAPS Master Index provides a wizard to guide you through the initial setup steps, and various editors so you can further customize the configuration of the master index application. Oracle Java CAPS Master Index automatically generates the components you need to implement a master index application.

Here are some of the features or Oracle Java CAPS Master Index.

- **Rapid Development** - Oracle Java CAPS Master Index allows for rapid and intuitive development of a master index application using a wizard to create the master index configuration and using XML documents to configure the attributes of the index. Templates are provided for quick development of person and company object structures.
- **Automated Component Generation** - Oracle Java CAPS Master Index automatically creates the Oracle Java CAPS Master Index configuration files that define the primary attributes of the master index application, including the configuration of the Enterprise Data Manager (EDM). Oracle Java CAPS Master Index also generates scripts that create the appropriate database schemas and an Object Type Definition (OTD) based on the object definition you create and configure.
- **Configurable Survivor Calculator** - Oracle Java CAPS Master Index provides predefined strategies for determining which field values to populate in the single best record (SBR). You can define different survivor rules for each field and you can create a custom survivor strategy to implement in the master index application.
- **Flexible Architecture** - Oracle Java CAPS Master Index provides a flexible platform that allows you to create a master index application for any business object. You can customize the object structure so the master index application can match and store any type of data, allowing you to design an application that specifically meets your data processing needs.
- **Configurable Matching Algorithm** - Oracle Java CAPS Master Index provides standard support for the Match Engine and also provides the ability to plug in a custom matching algorithm of your choice.
- **Custom Java API** - Oracle Java CAPS Master Index generates a Java API that is customized to the object structure you define. You can call the methods in this API in the Collaborations and Business Processes that define the transformation rules for data processed by the master index application.
- **Standard Reports** - Oracle Java CAPS Master Index provides a set of standard reports with each master index application that can be run from a command line or from the EDM. The reports help you monitor the state of the data stored in the master index application and help you identify configuration changes that might be required. You can also create custom reports using any ODBC-compliant reporting tool, SQL, or Java.

Master Index Repository Components

The components of Oracle Java CAPS Master Index are designed to work within Java CAPS to create and configure the master index application and to define connectivity between external systems and the master index application. The primary components of Oracle Java CAPS Master Index include the following.

- "Wizard Editor" on page 11
- "Editors" on page 11

Wizard Editor

The Wizard Editor takes you through each step of the master index application setup process, and creates the XML files that define the configuration of the application. The wizard allows you to define the name of the master index application, the objects to store, the fields in each object and their attributes, the Enterprise Data Manager (EDM) configuration, and the database and match engine platforms to use. The wizard generates a set of configuration files and database scripts based on the information you specify. You can further customize these files as needed.

Editors

Oracle Java CAPS Master Index provides the following editors to help you customize the files generated in the Oracle Java CAPS Master 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 created by the wizard. 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 Index configuration file.
- **Text Editor** - Allows you to review and customize the database scripts created by the wizard. 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 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

A master index application is implemented within a project in the Java CAPS repository. When you create an master index application, a set of configuration files and a set of database files are generated based on the information you specified in the wizard. When you generate the project, additional components are created, including a method OTD, an outbound OTD, Business Process methods, necessary `.jar` files, and a Custom Plug-in function that allows you to define additional custom processing for the index. To complete the project, you create a Connectivity Map and Deployment Profile.

Additional components can be added to the client projects that access the master index application, including Services, Collaborations, OTDs, Web Connectors, Adapters, JMS

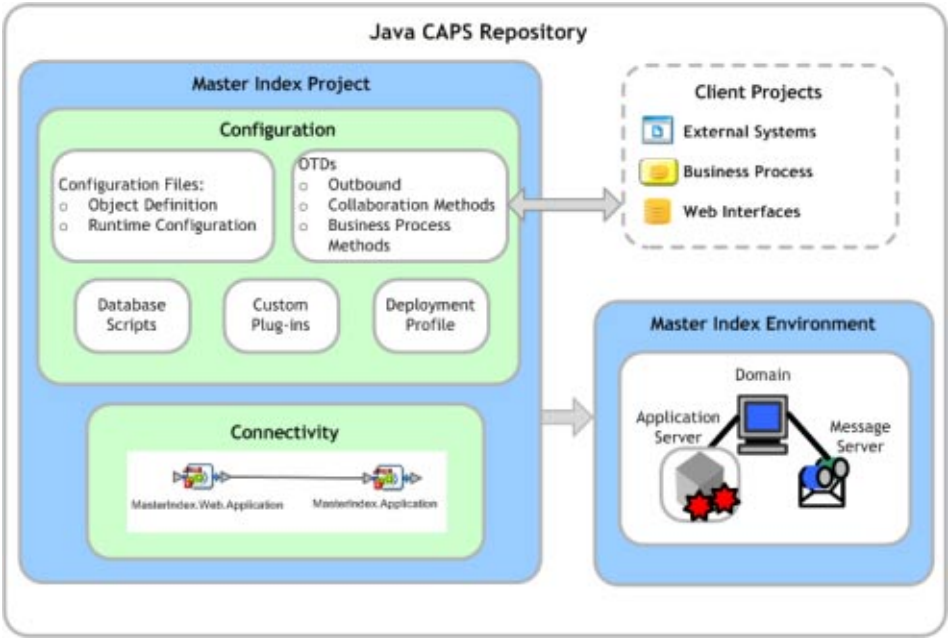
Queues, JMS Topics, Business Processes, and so on. You can use the standard Java CAPS editors, such as the OTD or Collaboration editors, to create these components.

Following is a list of Oracle Java CAPS Master Index project components.

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

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

FIGURE 1 Master Index Design-Time Components



Configuration Files

Several XML files together determine certain characteristics of the master index application, such as how data is processed, queried, and matched. These files configure the runtime components of the master index application, which are listed in "Master Index Runtime Components" on page 18.

- **Object Definition** - Defines the data structure of the object being indexed in a master index application.
- **Enterprise Data Manager** - Configures the search functions and appearance of the 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

Two database scripts are generated by the wizard to define external systems and code lists. Additional scripts to create or drop database tables are created when you generate the project (or by the wizard if you choose to generate all project files in the wizard).

- **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 database script** - 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 an Oracle or SQL Server database to create a customized master index database.
- **Drop database script** - 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.

You can also create custom scripts to store in the master index application project and run against the master index database.

Custom Plug-ins

Oracle Java CAPS Master Index provides a method by which you can create custom processing logic for the master index application. To do this, you need to define and name a custom plug-in, which is a Java class that performs the required functions. Once you create a custom plug-in, you incorporate it into the index by adding it to the appropriate configuration file. You can create custom update procedures and field validations, as well as define custom master index components. Update procedures must be referenced in the update policies of the Best Record file; field validations must be referenced in the Field Validation file; and custom components must be referenced in the configuration file for that component. For example, if you create a custom Query Builder, it must be listed in the Candidate Select file to be accessible to the master index application.

Match Engine Configuration Files

Several configuration files for the Oracle Java CAPS Match Engine are created in the master index project. The configuration files under the Match Engine node define certain weighting characteristics and constants for the match engine. The configuration files under the Standardization Engine node define how to standardize names, business names, and address fields. You can customize these files as needed.

Outbound Object Type Definition (OTD)

Oracle Java CAPS Master Index generates an outbound OTD based on the object structure defined in the Object Definition file. This OTD is used for distributing information that has been added or updated in the master index application 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 API

Due to the flexibility of the object structure, Oracle Java CAPS Master Index generates several dynamic Java methods for use in Collaborations and in Business Processes. One set is provided in a method OTD for use in Collaborations and one set is provided for Business Processes. The names, parameter types, and return types of these methods vary based on the objects you defined in the object structure.

Generating the master index application creates a *method OTD* that includes Java functions you can use to define data processing rules in Collaborations. These functions allow you to define

how messages received from external systems are processed by the Service. You can define rules for inserting new records, retrieving record information, updating existing records, performing match processing, and so on.

In addition to the method OTD, which can be used in Collaborations, Oracle Java CAPS Master Index creates a set of Java methods that can be incorporated into Business Processes and into Web Services. These methods are a subset of those defined for the method OTD, providing the ability to view, retrieve, and match information in the master index application database.

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 an Oracle or SQL Server 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 of the master index application. It contains information about the assignment of Services and message destinations to the application server and JMS IQ Managers within the master index system. Each master 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 must deploy the project before you can use the custom master index application you created using Oracle Java CAPS Master Index.

Environment Components

The Oracle Java CAPS Master Index Environments define the deployment environment of the master index application, including the domain, application server, external systems, and so on. If master index client projects use the same Environment, it might also include a JMS IQ Manager, constants, Web Connectors, and External Systems. Each Environment represents a unit of software that implements one or more master index applications. You must define and configure at least one Environment for the master index application before you can deploy the application. The application server hosting the master index application is configured within the Environment in NetBeans.

Learning about the Master Index Runtime Environment

In today's business environment, important information about certain business objects in your organization might exist in many disparate information systems. It is vital that this information flow seamlessly and rapidly between departments and systems throughout the entire business network. As organizations grow, merge, and form affiliations, sharing data between different information systems becomes a complicated task. The master index applications you create from Oracle Java CAPS Master Index can help you manage this data and ensure that the data you have is the most current and accurate information available.

Regardless of how you define the structure of the business object and configure the runtime environment for the master index application, the final product will include much of the same functions and features. The master index application provides a cross-reference of centralized information that is kept current by the logic you define for unique identification, matching, and update transactions.

Functions of the Runtime Environment

In the runtime environment, the master index application provides the following functions to help you monitor and maintain the data shared throughout the index system.

- **Transaction History** - The system provides a complete history of each object by recording all changes to each object's data. This history is maintained for both the local system records and the SBR.
- **Data Maintenance** - The web-based user interface supports all the necessary features for maintaining data records. It allows you to add new records; view, update, deactivate, or reactivate existing records; and compare records for similarities and differences. You can perform these functions against each local system record or SBR associated with an enterprise object.
- **Search** - The information contained in each SBR or system record can be obtained from the database using a variety of search criteria. You can perform searches against the database for a specific object or a set of objects. For certain searches, the results are assigned a matching weight that indicates the probability of a match.
- **Potential Duplicate Detection and Handling** - One of the most important features of the master index application is its ability to match records and identify possible duplicates. Using matching algorithm logic, the index identifies potential duplicate records and provides the functionality to correct the duplication. Potential duplicate records are easily corrected by either merging the records in question or marking the records as "resolved".
- **Merge and Unmerge** - You can compare potential duplicate records and then merge the records if you find them to be actual duplicates of one another. You can merge records at either the EUID or system record level. You can determine which record to retain as the active record and what information from each record to preserve in the resulting record.

- **Reports** - You can generate reports that provide information about the current state of the data in the master index application, 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.

Features of the Runtime Environment

The runtime components of the master index application are designed to uniquely identify, match, and maintain information throughout a business enterprise. These components are highly configurable, allowing you to create a custom master index application suited to your specific data processing needs.

Primary features of the master index application include the following:

- **Centralized Information** - The master index application maintains a centralized database, enabling the integration of data records throughout the enterprise while allowing local systems to continue operating independently. The index stores copies of local system records and of SBRs, which represent the most accurate and complete data for each object. This database is the central location of information and identifiers, and is accessible throughout the enterprise.
- **Configurability** - Before deploying the master index application, you define the components and processing capabilities of the system to suit your organization's processing requirements. You can configure the object structure, matching and standardization rules, survivorship rules, queries, EDM appearance, and field validation rules.
- **Cross-referencing** - The master index application is a global cross-referencing application that automates record matching across disparate source systems, simplifying the process of sharing data between systems. The master index application uses the local identifiers assigned by your existing systems as a reference, allowing you to maintain your current systems and practices while maintaining the most current and accurate information.
- **Data Cleansing** - The master index application uses configurable matching algorithm logic to uniquely identify object records and to identify duplicate and potential duplicate records. The index provides the functionality to easily merge or resolve duplicates and can be configured to automatically match records that are found to be duplicates of one another.
- **Data Updates** - The master index application provides the ability to add, update, deactivate, and delete data in the database tables through messages received from external systems. Records received from external systems are checked for potential duplicates during processing. Merges can also be performed through external system messages. Data updates from external systems can occur in real time or as batch processes.
- **Identification** - The master index application employs configurable probabilistic matching technology, which uses a matching algorithm to formulate an effective statistical measure of how closely records match. Using a state-of-the-art algorithm in real-time mode and establishing a common method of locating records, the index consistently and precisely identifies objects within an enterprise.

- **Integration** - Relying on the application server, the master index application provides the power and flexibility to identify, route, and transform data to and from any system or application throughout your business enterprise. It can accept incoming transactions and distribute updates to external systems, providing seamless integration with the systems in your enterprise.
- **Matching Algorithm** - The master index application is designed to use the Match Engine or a custom matching algorithm to provide a matching probability weight between records. Oracle Java CAPS Master Index provides the flexibility to create user-defined matching thresholds, which control how potential duplicates and automatic matches are determined.
- **Shared Information** - Each time a record is updated, added, merged, or unmerged from the EDM, the master index application generates a message that can be transmitted to external systems. It also receives, processes, and broadcasts messages containing information about the objects in your index.
- **Unique Identifier** - Records from various systems are cross-referenced using an enterprise-wide unique identifier, known as an EUID, that the index assigns to each object record. The index uses the EUID to cross-reference the local IDs assigned to each object by the various computer systems throughout the enterprise.

Master Index Runtime Components

The master index applications created by Oracle Java CAPS Master Index are made up of several components that work together to form the complete indexing system. The primary components of the master index application include the following:

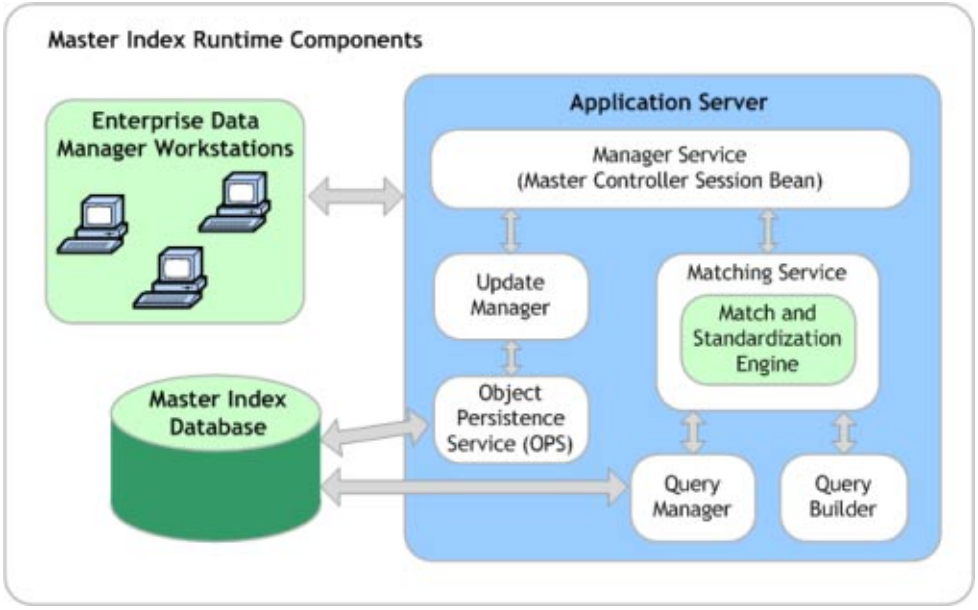
- "Matching Service" on page 19
- "Manager Service" on page 19
- "Query Builder" on page 19
- "Query Manager" on page 20
- "Update Manager" on page 20
- "Object Persistence Service (OPS)" on page 20
- "Database" on page 20
- "Enterprise Data Manager" on page 20

In addition, the master index application uses the connectivity components defined in the Oracle Java CAPS Master Index server and client projects to route data between external systems and the master index application.

The Java CAPS repository stores information about the configuration and structure of the master index environment. Because the master index application is deployed within the repository, it can be implemented in a distributed environment. The master index system requires the GlassFish Application Server.

The components of a master index application are illustrated in the following figure.

FIGURE 2 Master Index Runtime Components



Matching Service

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

Manager Service

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

Query Builder

The Query Builder defines all queries available to the master index application, including the queries performed automatically by the master index application when searching for possible matches to an incoming record. It also includes the queries performed manually through the

Enterprise Data Manager (EDM). The EDM queries can be either alphanumeric or phonetic and have the option of using wildcard characters. The configuration of the Query Builder is defined in the Candidate Select file.

Query Manager

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

Update Manager

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

Object Persistence Service (OPS)

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

Database

The master index application uses an Oracle or SQL Server database to store the information you specify for the business objects being cross-referenced. The database stores local system records, the single best record for each object record, and certain administrative information, such as drop-down menu lists, processing codes, and information about the systems from which data originates. The scripts that are generated to create the database tables are based on the information specified in the Object Definition file.

Enterprise Data Manager

The Enterprise Data Manager (EDM) is a web-based interface that allows you to monitor and maintain the data in your master index database. Most of the configurable attributes of the EDM are defined by information you specify in the wizard, but you can further configure the EDM in the Enterprise Data Manager file after you generate the Oracle Java CAPS Master Index application. The EDM provides the ability to manually search for records; update, add, deactivate, and reactivate records; merge and unmerge records; view potential duplicates; and view comparisons of object records.

Enterprise Records

An *enterprise record* includes all components of a record that represents one entity. The master index application stores two different types of records in each enterprise record: system records and a single best record (SBR). A system record contains an enterprise record'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 an enterprise record. An enterprise record consists of both system records and the 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 SBR in that enterprise record. 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 an object is created from the most reliable information contained in each system record representing that object. 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 a master index application, each system record and SBR in an enterprise record typically contains a set of objects that store different types of information about the business object. A record usually contains a parent object and several child objects. A record can have only one parent object, but can have multiple child objects and multiple instances of each type of child object. For example, if the business object being indexed is a person, the record can only contain one primary name and social security number, which would be contained in the parent object (for example, a person object). However, the record could have multiple addresses, telephone numbers, and aliases, which would each be defined in different child objects (for example, in address, phone, and alias objects). Each address would be stored in a different instance of an address object.

Working with Project Components

Oracle Java CAPS Master Index supports standard Java Composite Application Platform Suite functions for version control, XML validation, and copy, cut, and paste functions.

Version Control

Oracle Java CAPS Master Index supports the version control functionality provided by Java CAPS. You can check files in and out, retrieve older versions to a workspace, view a version history, and so on. In addition, Oracle Java CAPS Master Index supports recursive check-ins and check-outs. When you select Recurse Project, you can check in or out all components below the selected node or a subset of those components.

Copying, Cutting, and Pasting Files

You can use standard cut, copy, and paste commands to copy or move files between projects. Oracle Java CAPS Master Index follows the standard functionality, with the exception that you can only copy or move a component from one project into the same node of another project. For example, you can only paste a copied configuration file into the Configuration node of another project. In addition, you cannot cut or delete components that are essential to a project, such as the configuration files, match and standardization files, and so on.

Master Index Development Process Overview (Repository)

This document provides instructions for creating a master index application using Oracle Java CAPS Master Index. It includes creating the master index application framework, creating custom Java code, generating the application files, and creating the database.

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. Create a new project and a new Oracle Java CAPS Master Index application within that project (described in "Creating a Master Index Application (Repository)" on page 26).
3. Define the object structure, operating environment, and certain runtime characteristics using the wizard (described in "Creating a Master Index Application (Repository)" on page 26).
4. Define and build custom plug-ins, and specify the plug-ins in the appropriate configuration file (described in "Implementing Master Index Custom Plug-ins (Repository)" on page 53).
5. Customize the configuration files (instructions for this step are provided in a separate document, *Configuring Oracle Java CAPS Master Indexes (Repository)*).
6. Generate the master index application (described in "Generating the Master Index Application (Repository)" on page 55).

7. Create the database (described in "Creating the Master Index Database (Repository)" on page 62).
   - 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.
8. 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*).
9. 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.
10. Define the Environment (described in *Configuring Oracle Java CAPS Master Index (Repository) Connectivity and Environments*).
11. Create the Deployment Profile and deploy the project (described in *Deploying Oracle Java CAPS Master Indexes (Repository)*).
12. Define security (described in "Defining Master Index Security (Repository)" in *Maintaining Oracle Java CAPS Master Indexes (Repository)*).
13. 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 Index Framework and the Runtime Environment (Repository)

The values you enter in the wizard, 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 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 Enterprise Data Manager

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

- The fields that appear on the 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 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 (Repository)

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 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 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 (Repository)" on page 25
- "Planning a Master Index Project (Repository)" on page 25
- "Master Index Project Initiation Checklist (Repository)" on page 26

Preliminary Data Analysis for a Master Index (Repository)

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 (Repository)

Before you create the Oracle Java CAPS Master 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 (Repository)

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
- 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 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 EDM search and search results windows
- The processing codes for the source systems being integrated into the index

Creating a Master Index Application (Repository)

The wizard provides a simple method for you to create the object definition and the runtime configuration files for your master index application. This section provides instructions for creating a new project and using the wizard to create the configuration files for the master index application. To create the initial master index framework, follow these steps.

- "Step 1: Create a Project and Start the Wizard (Repository)" on page 27
- "Step 2: Name the Master Index Application (Repository)" on page 28
- "Step 3: Define Source Systems (Repository)" on page 30
- "Step 4: Define the Deployment Environment (Repository)" on page 31
- "Step 5: Define Parent and Child Objects (Repository)" on page 33
- "Step 6: Define the Fields for Each Object (Repository)" on page 36
- "Step 7: Generate the Project Files (Repository)" on page 40

- "Step 8: Review the Configuration Files (Repository)" on page 41

Step 1: Create a Project and Start the Wizard (Repository)

Before you can access the wizard, you must create and name a new NetBeans project under the Java CAPS Repository.

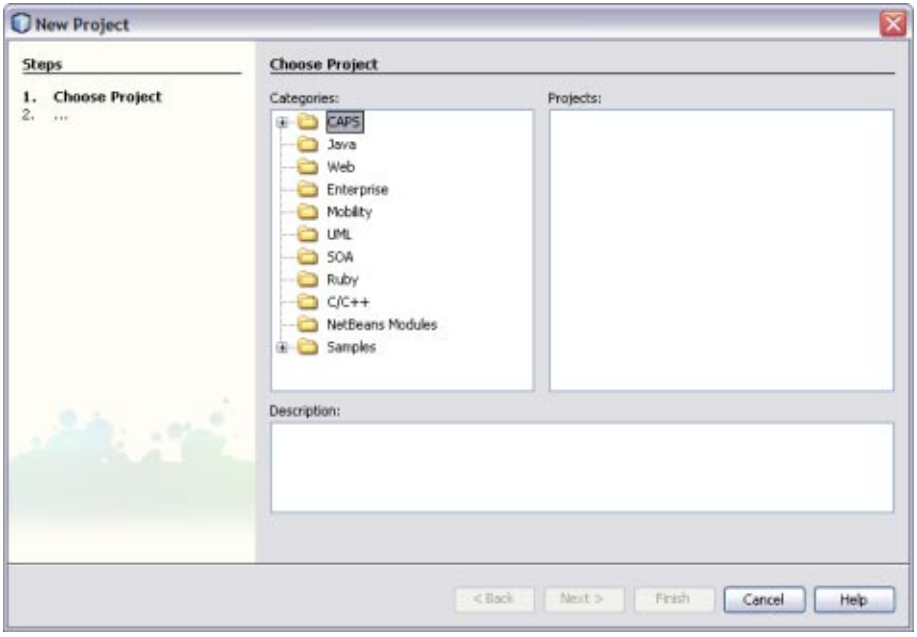
▼ To Create a Project

Before You Begin Before you begin this step, make sure you are connected to the Java CAPS Repository in NetBeans.

1 **Right-click in the NetBeans Projects window, and then select New Project.**

The New Project wizard appears.

FIGURE 3 Creating a New Project

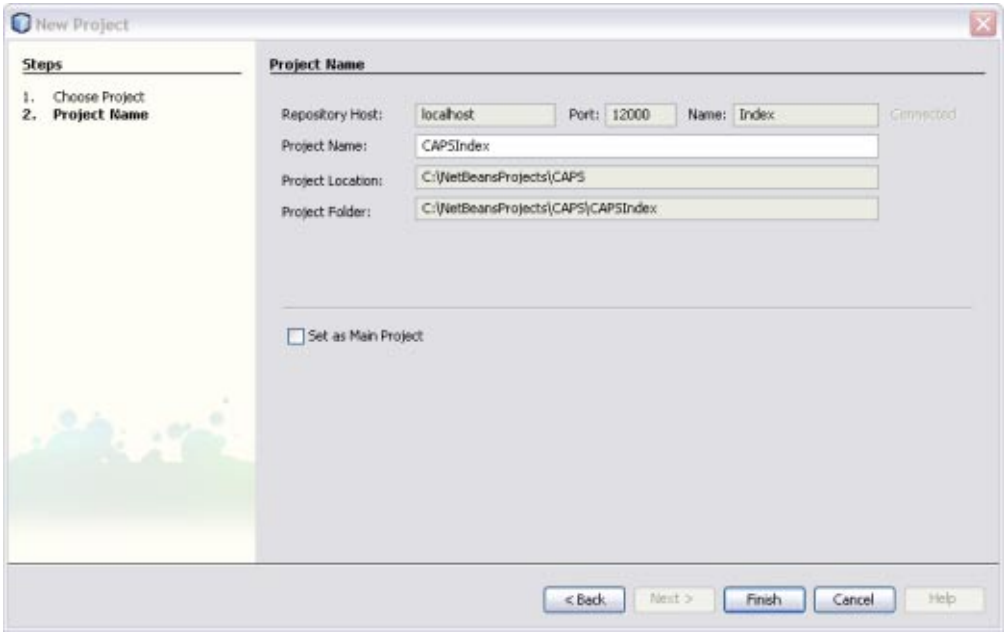


2 **Under Categories, select CAPS and then ESB.**

3 **Under Projects, select CAPS Repository–Based Project.**

The Project Name window appears.

FIGURE 4 Project Name and Location



4 **On the Project Name window, enter a name for the project.**

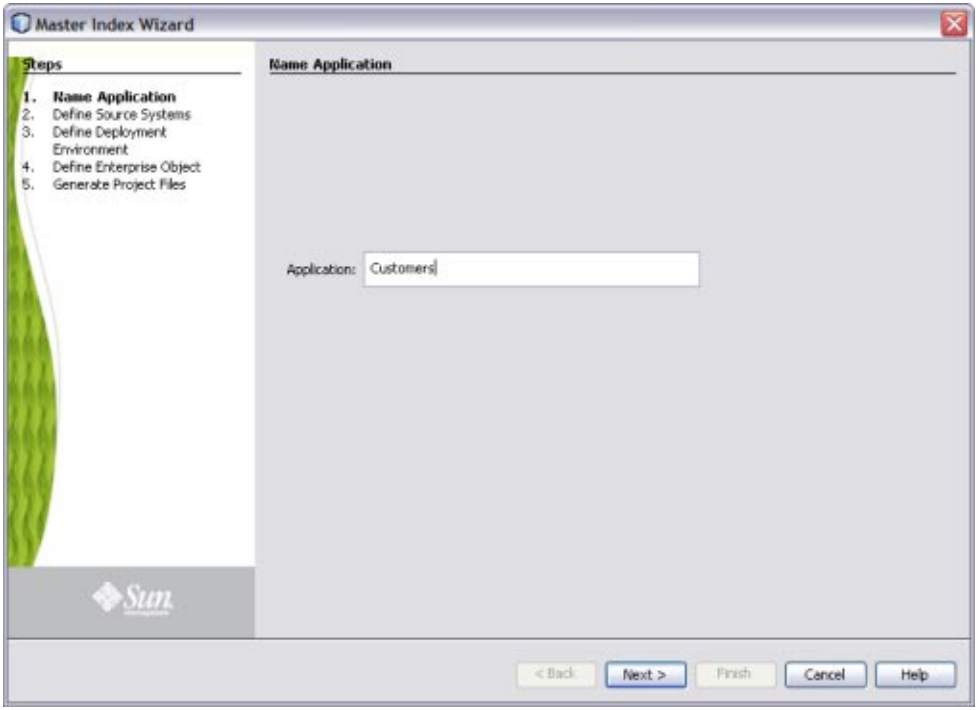
5 **Click Finish.**

6 **Continue to "Step 2: Name the Master Index Application (Repository)" on page 28.**

Step 2: Name the Master Index Application (Repository)

Before you can configure the new master index application, you must give the master index application a unique name.

FIGURE 5 Name Application Window



▼ To Name the Master Index Application

1 **Complete "Step 1: Create a Project and Start the Wizard (Repository)" on page 27.**

2 **In the Projects window, right-click the new project, point to New, and then select Master Index Application.**

The Master Index wizard appears.

3 **In the Application field of the Name Application window, enter a name for the new master index application, and then click Next.**

Note – The name you enter for the application should be the same name you will give to the parent object when you define the object structure.

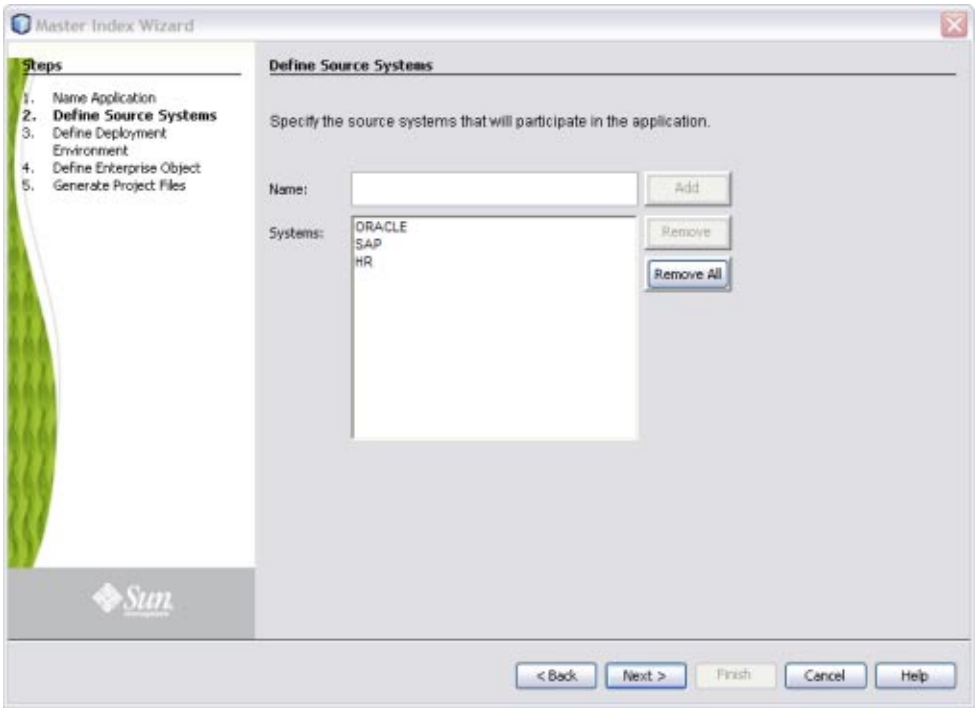
The Define Source Systems window appears.

4 **Continue to "Step 3: Define Source Systems (Repository)" on page 30.**

Step 3: Define Source Systems (Repository)

After you specify a name for the new master index application, you need to specify the processing codes for the source systems that will be integrated into the master index system.

FIGURE 6 Define Source Systems



▼ To Define Source Systems

1. **Complete "Step 2: Name the Master Index Application (Repository)" on page 28.**

2. **In the Name field of the Define Source Systems window, enter the processing code of one of the source systems that will share data in the master index system, and then click Add.**

   You can enter any of the following characters:

   - ! _ ~ ( ) { } + \Q # $ % & : ; - /
   - a-z
   - A-Z
   - 0-9

- þ ÿ Þ ß ’à-ö ø-ý À-Ö Ø-Ý

  The value you entered appears in the Systems box.

Note – Be sure to enter the processing code of the system and not the name. This value is entered in the Best Record file for defining survivor strategies for the SBR.

---

3 **Do any of the following:**

- **To define additional systems, repeat the above step for each source system that will share information with the master index application.**
- **To remove a system from the list, highlight the name of that system in the Systems box, and then click Remove.**
- **To remove all systems from the list, click Remove All.**

4 **When you have defined all required source systems, click Next.**

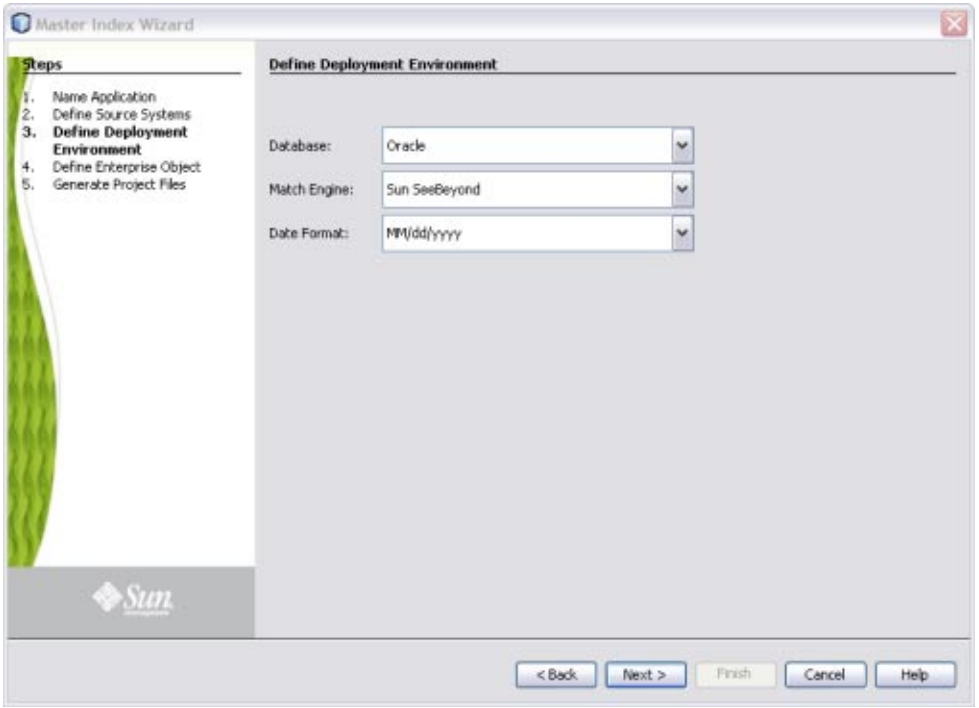
The Define Deployment Environment window appears.

5 **Continue to “Step 4: Define the Deployment Environment (Repository)” on page 31.**

Step 4: Define the Deployment Environment (Repository)

Once you define systems, you must specify information about the deployment environment, including the database, match engine, and standardization engine vendors.

FIGURE 7 Define Deployment Environment



▼ To Define the Deployment Environment

1. **Complete "Step 3: Define Source Systems (Repository)" on page 30.**

2. **On the Define Deployment Environment window, enter the following information:**

   - **Database** – The type of database being used for the master index application. You can select Oracle or SQL Server.
   - **Match Engine** – The type of match and standardization engine to use for the implementation. Currently, the only option is "Sun Match Engine".
   - **Date Format** – The date format for the master index system. This defines how dates should be entered and how they appear on the EDM. You can select MM/dd/yyyy, dd/MM/yyyy, or yyyy/MM/dd.

3. **When you have defined the deployment environment fields, click Next.**

4. **Continue to "Step 5: Define Parent and Child Objects (Repository)" on page 33.**

Step 5: Define Parent and Child Objects (Repository)

After you define the deployment environment for the master index application, you can begin to define the structure of the object you want to index. The primary object will be the parent object for any other objects defined. Child objects are not required if all information is stored under the parent object.

You can create new undefined objects, create objects using predefined templates, or use a combination of both methods to create the objects in your enterprise object. Perform any of the following actions to define the objects in the enterprise object.

- "Creating Undefined Objects" on page 33
- "Creating Objects from a Template" on page 35
- "Deleting an Object from the Structure" on page 36

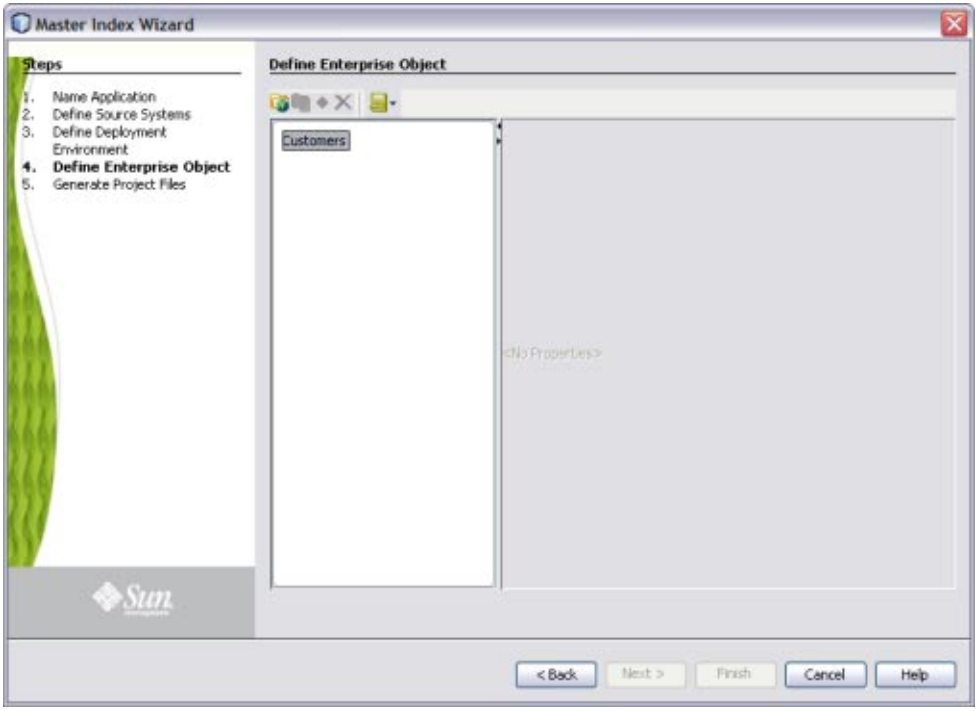
Complete "Step 4: Define the Deployment Environment (Repository)" on page 31 before performing these procedures.

Note – The names of database constraints are created based on the parent and child object names. Due to database naming restrictions, the length of the parent object name plus the length of any of the child object names must be 21 characters or less. Give the parent object the same name you gave to the application earlier in the wizard.

Creating Undefined Objects

When you create undefined objects, you create an empty object with no predefined fields or child objects.

FIGURE 8 Define Enterprise Object



▼ To Create Undefined Parent and Child Objects

1. **On the Define Enterprise Object window, click Add Primary Object.**

   The initial node appears on the tree. By default, the name of the field is the same as the name of the application you defined in "Step 2: Name the Master Index Application (Repository)" on page 28.

2. **Accept the default name by pressing Enter, or type a new name and press Enter.**

3. **To create a new child object, select the primary object created above and then click Add Sub Object.**

   The new child node appears on the tree.

4. **Accept the default name by pressing Enter, or type a new name and press Enter.**

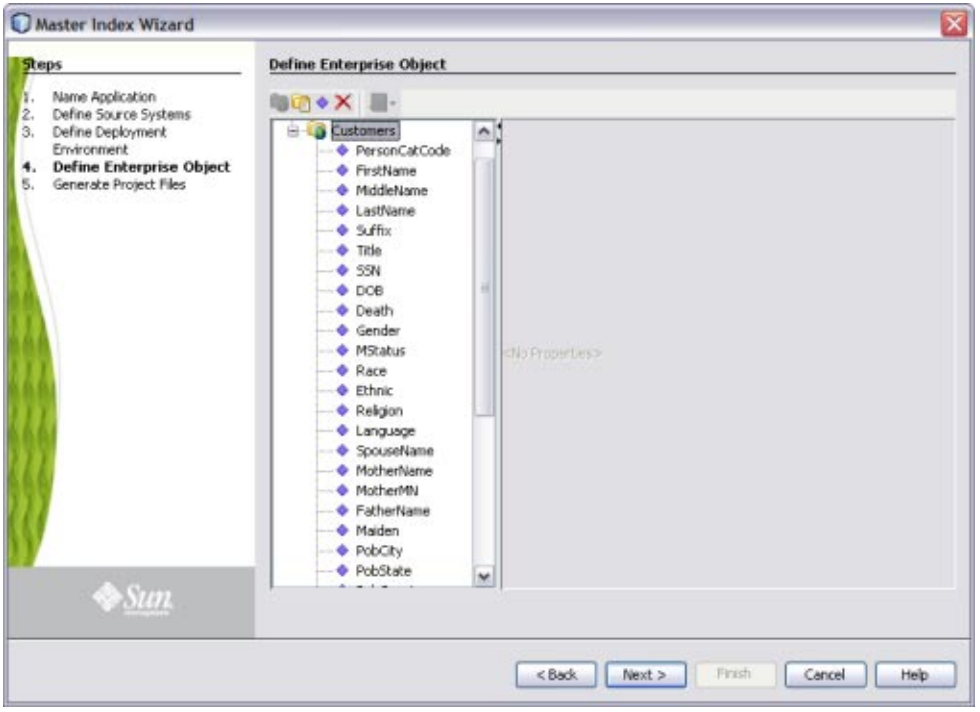
5. **Repeat the previous two steps for each child object.**

6. **Continue to "Step 6: Define the Fields for Each Object (Repository)" on page 36.**

Creating Objects from a Template

When you create objects from a template, secondary objects and fields are predefined. You can modify the objects, fields, and field properties in a template to suit your processing needs.

FIGURE 9 Define Enterprise Object



▼ To Create Parent and Child Objects From a Template

1. **To create a complete object structure from a template, click Templates on the Define Enterprise Object toolbar, and select the template you want to use.**

   The objects and fields from the template appear in the tree-view panel in the center of the window.

2. **To create a child object from a template after creating the parent object, right-click the parent object, point to Template, and then select the name of the template you want to use.**

   The new object and any defined fields appear in the object tree.

3 **If necessary, change the name of the new object by doing the following:**

   a. **Click twice on the name.**

   b. **Type the new name.**

   c. **Press Enter.**

4 **Repeat steps 2 and 3 for each child object template you want to create.**

5 **Continue to** **"Step 6: Define the Fields for Each Object (Repository)" on page 36****.**

Deleting an Object from the Structure

If you add an object in error, or do not want to use one of the objects in a predefined template, you can delete the object from the structure.

▼ To Delete an Object From the Structure

1 **On the Define Enterprise Object window, select the object you want to remove.**

2 **Do any of the following:**

- **Right-click in the object tree panel, and then select Delete.**
- **Press the Delete key.**
- **In the wizard toolbar, click Delete.**

  The object and any fields associated with that object are deleted. If you remove the parent object, all child objects are deleted.

Step 6: Define the Fields for Each Object (Repository)

After you define all the parent and child objects for your enterprise object, you need to define the fields in each object. Every field has a set of properties that must be configured before creating the master index configuration files. If you chose a predefined template to create your objects, be sure to check the properties for all predefined fields to be sure they are configured correctly for your implementation.

After you define the parent and child objects, you can perform any of the following actions to define the fields for those objects.

- "Adding a Field" on page 37

- "Configuring Field Properties" on page 37
- "Deleting a Field" on page 39

Adding a Field

If you created an empty object in "Step 5: Define Parent and Child Objects (Repository)" on page 33, you need to create each field that belongs to the object. If you created objects using a predefined template, you can add new fields to the object if needed.

▼ To Add a Field

1. **Complete "Step 5: Define Parent and Child Objects (Repository)" on page 33.**
2. **In the object tree panel of the Define Enterprise Object window, do one of the following:**
   - **To add the field to the end of the object's field list, select the name of the object to which you want to add a new field and then click Add Field.**
   - **To add the field immediately following an existing field, select the field after which you want to add the new field and then click Add Field.**

     The tree expands and a new field is inserted.
3. **Do one of the following:**
   - **To accept the default name, press Enter.**
   - **To change the name, type the new name and then press Enter.**

     For information about field name restrictions, see "Master Index Wizard Field Name Restrictions (Repository)" on page 42.
4. **Continue to "Configuring Field Properties" on page 37.**

Configuring Field Properties

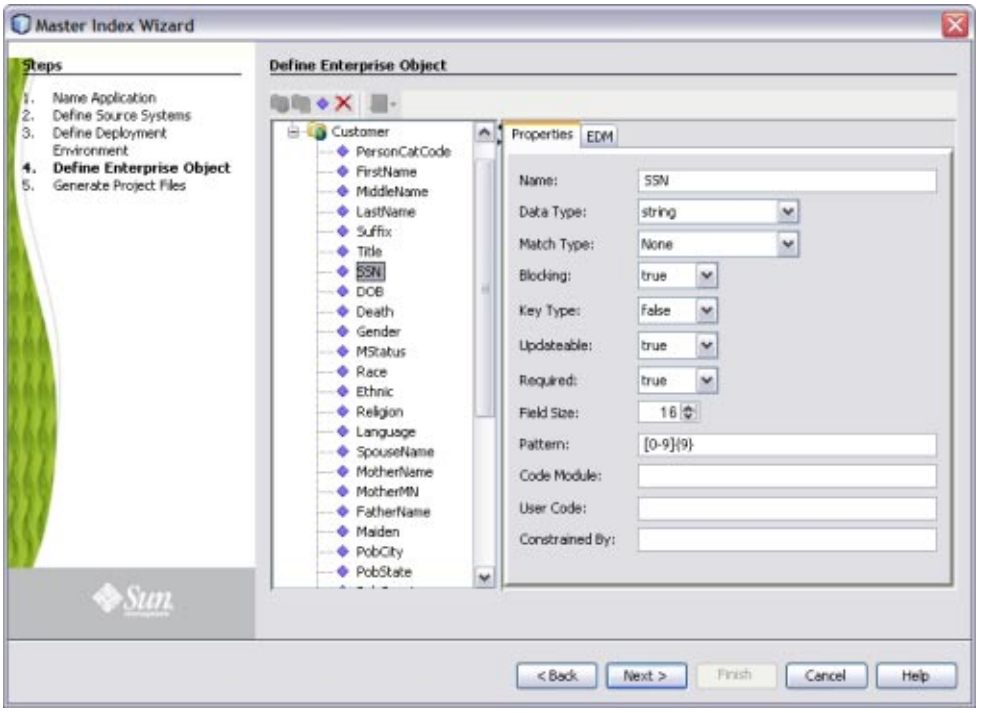
When you create a field, a set of default properties is defined for that field. You can modify the property configuration for each field to suit your data processing, storage, and display requirements. After you modify a property value, press `Enter` to apply the change.

▼ To Configure Field Properties

1. **Complete the steps under "Adding a Field" on page 37.**
2. **In the object tree panel of the Define Enterprise Object window, select the field you want to configure.**

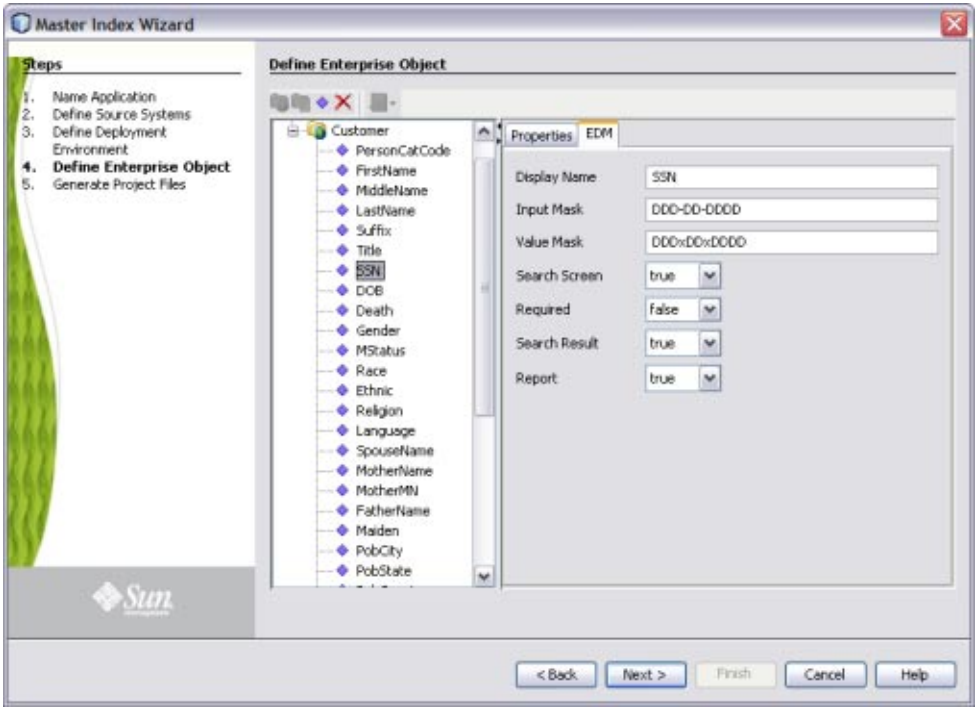
3 **On the Properties page in the right side of the window, modify the value of any of the properties listed in "Master Index Wizard General Field Properties (Repository)" on page 42.**

FIGURE 10 Field Properties



4 **On the right side of the window, click the EDM tab, and then modify the value of any of the properties listed in "Master Index Wizard EDM Field Properties (Repository)" on page 44.**

FIGURE 11 Field EDM Properties



5. **When you have created and configured all of the necessary fields for each object, click Next.**

6. **Continue to "Step 7: Generate the Project Files (Repository)" on page 40.**

Deleting a Field

If you add a field in error, or do not need one of the predefined fields from a template, you can delete the field.

▼ To Delete a Field

1. **In the object tree panel of the Define Enterprise Object window, select the field you want to delete.**

2. **Right-click in the object tree panel.**

3. **From the context menu, select Delete.**

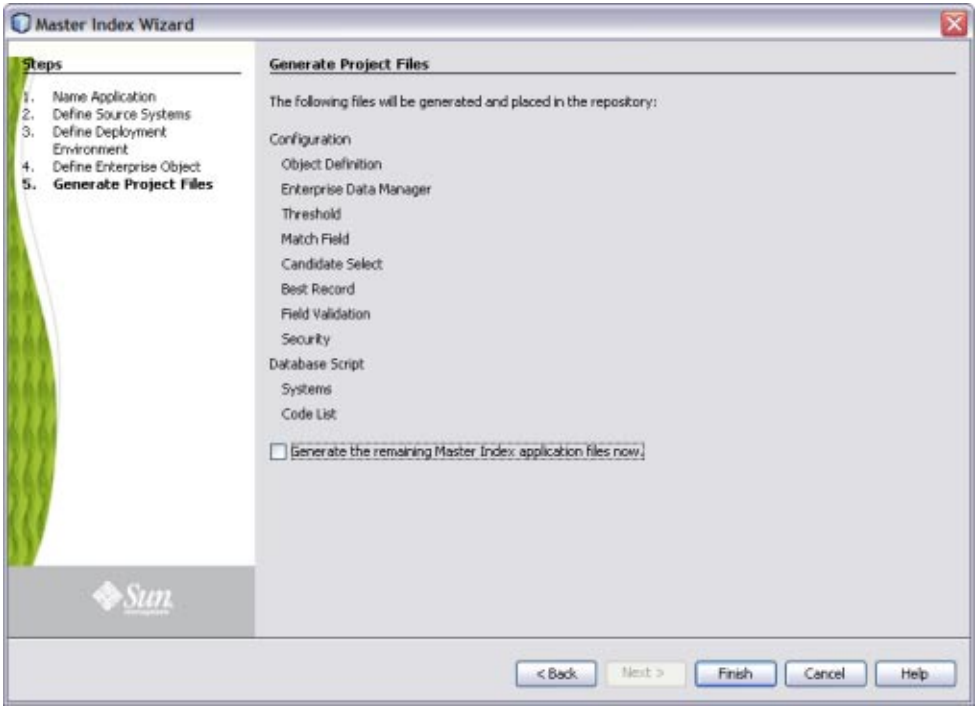
   The field is removed from the object tree.

Step 7: Generate the Project Files (Repository)

Once you have named the application and configured the source systems, deployment environment, objects, and fields for the master index application, you need to generate the configuration files and database scripts. You have the option to create all additional project files at this time (such as the JAR files and OTDs) or to wait until you have customized the configuration files for the master index application. Review the configuration files to be sure the application is set up correctly for your data processing environment.

Note – Modifying the configuration files is not covered in this document. For instructions on configuring a master index application, see *Configuring Oracle Java CAPS Master Indexes (Repository)*.

FIGURE 12 Generate Project Files



▼ To Generate the Configuration Files

1. **Complete "Step 6: Define the Fields for Each Object (Repository)" on page 36.**
2. **Verify that all of the information you have entered is complete and correct.**
3. **To generate all application files for the project, select the check box at the bottom of the window. To generate them later, after reviewing the configuration files, leave this check box unchecked.**
4. **On the Generate Configuration Files window, click Finish.**

   The configuration files are generated, and are stored in the Repository.
5. **Continue to "Step 8: Review the Configuration Files (Repository)" on page 41.**

Step 8: Review the Configuration Files (Repository)

After the wizard is complete, several nodes representing the master index configuration files are placed in the project for the master index application. Verify that the configuration files are customized correctly for your implementation. If you need to modify the configuration files, see *Configuring Oracle Java CAPS Master Indexes (Repository)* for information about configuration tasks. For information about the configuration files and configurable options, see *Understanding Oracle Java CAPS Master Index Configuration Options (Repository)*.

Master Index Wizard Field Properties and Name Restrictions (Repository)

When you create fields in the object structure of the master index application, you can specify several properties for each field, such as whether the field is required, whether the field will appear as a drop-down menu on the EDM, whether the field will be used in a blocking query, and so on. There are also some restrictions for how fields can be named and how the properties are defined.

The following topics provide information about the naming restrictions and about the field properties of the wizard.

Master Index Wizard Field Name Restrictions (Repository)

When you name the fields in your object structure, be sure to keep the following guidelines in mind to avoid errors when compiling or running the master index application.

- Oracle Java CAPS Master Index automatically creates a field for each object named *object*Id, where *object* is the name of an object or sub-object. You cannot create fields with those names. For example, you cannot create a field named "AddressId" if there is an Address object in the object structure.
- If you enter a field name longer than 20 characters, a warning dialog box appears. While most databases can handle names up to at least 30 characters, Oracle Java CAPS Master Index appends text to the end of fields defined for phonetic encoding or standardization. For fields that will be parsed, normalized, or phonetically encoded, make sure the name of the original field does not exceed 20 characters. Any other field can have a name up to 30 characters long. For information about the names of the fields automatically created by the wizard, see *Understanding Oracle Java CAPS Master Index Processing (Repository)*.
- Do not use characters or names restricted by the database platform (Oracle or SQL Server), Java, or XML in field names.

Master Index Wizard General Field Properties (Repository)

The following table lists and describes the properties you can define on the General Properties page of the wizard.

Property	Description
Data Type	The master index data type of the field. The following data types are supported: ■ **string** - Contains a string of characters. ■ **date** - Contains a date value. ■ **float** - Contains a floating point integer. ■ **int** - Contains an integer. ■ **char** - Contains a single character. ■ **boolean** - Contains either **`true`** or **`false`**.

Property	Description
Match Type	The type of matching to be performed against the field, if the field is to be used for match weight generation. You must define at least one field for matching or no weights will be generated. The match types you specify here define the structure of the Match Field file, including the match string. The match types in the Match Field file might differ from the wizard match types. See *Understanding Oracle Java CAPS Master Index Configuration Options (Repository)* for information about the available options for this field and how the wizard match types correlate to the Match Field file types.
Blocking	An indicator of whether the field will be used in the blocking query. Specify **`true`** to add the field to the blocking query; specify **`false`** to omit it from the blocking query.
Key Type	An indicator of whether the field is used to identify unique objects. For example, a business index might store several addresses for each business. Each address is assigned an address type and each business can only have one address of each type. Specify **`true`** if the field is a unique record identifier, or **`false`** if it is not. Key type fields should also be required fields (see below), unless a combination of fields are specified as key types for an object. **Note –** It is recommended that each child object contain a key type field, but this is not required. If child objects do not contain one or more key type fields, each enterprise object might accumulate a very large number of child objects depending on the survivor strategy used.
Updateable	An indicator of whether the field can be updated from the EDM and external system messages. Specify **`true`** if the field can be updated or **`false`** if it cannot.
Required	An indicator of whether the field is required in order to save an enterprise object to the database. Specify **`true`** if the field is required or **`false`** if it is not. If only one key type field is defined for an object, that field should be required.
Size	The number of characters allowed in each field. This determines the number of characters allowed in the database columns and defines the maximum number of characters that can be entered into each field on the EDM.
Pattern	The required data pattern for the field. 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. You might want to define patterns for date, telephone, or SSN fields. Note that for the EDM, the pattern is further restricted by the value entered for the input mask described in the previous table. If no input mask is specified, all regex patterns are supported.

Property	Description
Code Module	The identification code for the drop-down list that appears for this field in the EDM. **Note –** This value must match an entry in the code column of the sbyn_common_header database table and, by default, an entry for the code you enter is created in the Code List database script. You can further customize code lists in the script after completing the wizard.
User Code	The processing code for the drop-down list that appears for the fields defined by the Constrained By property. For more information, see the description of the Constrained By property below. **Note –** This must match an entry in the code_list column of the sbyn_user_code database table.
Constrained By	The name of the field that contains the corresponding User Code value (described above). The User Code and Constrained By properties are used in conjunction to fulfill two purposes. The first purpose is to define a drop-down list for the field that contains the User Code value. The second purpose is to validate the field that contains the Constrained By value against definitions for the field with the User Code value. For example, if you store non-unique IDs such as credit card numbers or insurance policy numbers, you could create a field named ID Type that has a User Code value of CREDCARD, which is also defined as a code in the sbyn_user_code table. This gives the ID Type field a drop-down list based on the definitions for CREDCARD in the sbyn_user_code table. You could then create a field named ID that would be constrained by the formats defined for the ID Type field. Any IDs you enter would be validated against the value of the ID Type field.

Master Index Wizard EDM Field Properties (Repository)

The following table lists and describes the field properties you can define on the EDM Field Properties page of the wizard.

Property	Description
Display Name	The name of the field as it will appear on the EDM.

Property	Description
Input Mask	A mask used by the GUI to add punctuation to a field. For example, if users enter the date in the format MMDDYYYY, you can add an input mask to display the dates as MM/DD/YYYY. Use the value mask (described below) to strip the punctuation from the value before storing it in the database. To define an input mask, enter a character type for each character in the field and place any necessary punctuation between the character types. For example, the input mask for the above date format is DD/DD/DDDD. ■ **D** – indicates a numeric character. ■ **L** – indicates an alphabetic character. ■ **A** – indicates an alphanumeric character. Note that the value you enter can further restrict the data pattern for the field (this is the Pattern field property described in "Master Index Wizard General Field Properties (Repository)" on page 42). The following character types can be used.
Value Mask	A mask used by the index to strip any extra characters that were added by the input mask (see above). This mask ensures that data is stored in the database in the correct format. To specify a value mask, type the same value as is entered for the input mask, but type an "x" in place of each punctuation mark. For example, if an SSN field has an input mask of DDD-DD-DDDD, specify a value mask of DDDxDDxDDDD to strip the dashes before storing the SSN. A value mask is not required for date fields.
Search Screen	An indicator of whether the field appears on the search windows of the EDM. Specify **`true`** to display the field or **`false`** to hide it.
Required	An indicator of whether the field must be populated on the search windows of the EDM when performing a search. This property can only be modified if the Search Screen property is set to **`true`** (see above). Specify **`true`** to make the field required or specify **`false`** to make it optional. You can also specify **`oneof`** to create a group of fields of which at least one must be populated to perform a search. (Be sure to specify **`oneof`** for each field in the group.) **Tip –** If a field is required for a search, the field should also be required for creating a record (by specifying true for the Required property on the Properties page). Otherwise, searches performed from the EDM could result in no matches even though possible matches exist.
Search Result	An indicator of whether the field appears on the search results windows of the EDM. Specify **`true`** to display the field, or **`false`** to hide it.
Report	An indicator of whether the field appears on the reports generated from the EDM. Specify **`true`** to display the field on the reports; otherwise specify **`false`**.

Custom Plug-ins for Master Index Custom Transaction Processing (Repository)

You can add custom processing to the master index application using the Custom Plug-ins module of a Oracle Java CAPS Master 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. 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 (Repository)" on page 46 – 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 (Repository)" on page 48 – Define validations to perform against specific fields, such as checking the local ID length and format.
- "Master Index Field Masking Plug-ins (Repository)" on page 48 – Define how the values for sensitive fields are hidden on the EDM from users who do not have permission to view them.
- "Master Index Match Processing Logic Plug-ins (Repository)" on page 48 – Define custom logic based on predefined decision points for how records are matched during a transaction.
- "Master Index Custom Plug-in Exception Processing (Repository)" on page 50 – Define how exceptions are handled by the custom plug-ins you create.

Master Index Update Policy Plug-ins (Repository)

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.

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.

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.

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.

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.

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.

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.

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 (Repository)

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 (Repository)

There might be instances where you want to mask certain data in records from general users of the 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 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 the Enterprise Data Manager file).

Master Index Match Processing Logic Plug-ins (Repository)

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 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 (Repository)

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 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.CustomEDM</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 (Repository)

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 Enterprise Data Manager (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 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 (Repository)

Oracle Java CAPS Master 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 (Repository)" on page 51 – Define how the single best record (SBR) is generated.
- "Master Index Query Builder Plug-ins (Repository)" on page 51 – Define how queries are performed against the master index database.
- "Master Index Block Picker Plug-ins (Repository)" on page 52 – Define how the blocks of fields in a blocking query are selected during a query.
- "Master Index Pass Controller Plug-ins (Repository)" on page 52 – Define how the master index application determines whether to perform additional match passes against a record.
- "Match Engine Plug-ins (Repository)" on page 52 – Define logic that connects to a match engine other than the Oracle Java CAPS Match Engine.

- "Standardization Engine Plug-ins (Repository)" on page 52 - 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 (Repository)" on page 53 – Define logic that connects to phonetic encoders other than those provided.

Master Index Survivor Calculator Plug-ins (Repository)

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 (Repository)

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 (Repository)

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 (Repository)

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 (Repository)

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 (Repository)

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 (Repository)

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 (Repository)

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 (Repository)" on page 54
- "Building Master Index Custom Plug-ins (Repository)" on page 54

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 (Repository)" on page 46
- "Custom Plug-Ins for Master Index Custom Components (Repository)" on page 50

Creating Master Index Custom Plug-ins (Repository)

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 (Repository)" on page 54.**
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 (Repository)

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 (Repository)

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 create or 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** *Application*`_stc_eindex_client.jar` **and** *Application*`_stc_eindex_util.jar` **files to the** `/lib` **subdirectory in the reports home directory.**

   b. **In the** `/lib` **subdirectory, rename** *Application*`_stc_eindex_client.jar` **to** `stc_eindex_client.jar` **and rename** *Application*`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 *Application*`_stc_eindex_client.jar` and *Application*`_stc_eindex_util.jar` files to the `/lib` subdirectory in the reports home directory.**

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

Master Index Database Scripts and Design (Repository)

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 (Repository)

The wizard creates SQL scripts based on information you specified about code lists and external systems that you can use to define startup data for the master index application. When you generate the application, additional scripts are generated for creating or dropping database tables. These scripts appear under the Database Script node of the master index project, and are named Systems, Code List, Create *Application_Name* Database, and Drop *Application_Name* Database (where *Application_Name* is the name you defined for the application in the wizard). 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 (Repository)

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 versions, 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 (Repository)

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 Processing (Repository)*.

Designing the Master Index Database (Repository)

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 objects in your database. For example, if you are indexing person objects, there might be processing codes for genders, such as F for female, M for male, and so on. The processing codes and their descriptions are stored in a set of database tables known as common maintenance tables. The wizard creates a script to help you load the processing codes into the database.

When an enterprise object appears on the 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 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.

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. 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 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 (Repository)

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. The wizard defines SQL statements in the Code List and Systems SQL files (`codelist.sql` and `system.sql`) to insert startup data into the database based on information you specify, 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 EDM fields and to create drop-down lists for 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 (Repository)" on page 62
- "Step 2: Create a Master Index Database and User (Repository)" on page 63
- "Step 3: Define Master Index Database Indexes (Repository)" on page 64
- "Step 4: Define Master Index External Systems (Repository)" on page 65
- "Step 5: Define Master Index Code Lists (Repository)" on page 67
- "Step 6: Define Master Index User Code Lists (Repository)" on page 70
- "Step 7: Create Custom Master Index Database Scripts (Repository)" on page 71
- "Step 8: Create the Master Index Database Structure (Repository)" on page 72
- "Step 9: Specify a Starting EUID for a Master Index (Repository)" on page 73

You can also delete the database for testing purposes using the supplied script. See "Deleting Master Index Database Tables (Repository)" on page 74 for more information.

Step 1: Analyze the Master Index Database Requirements (Repository)

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 (Repository)" on page 63.

For additional information and guidelines about how to set up your database, see "Master Index Database Scripts and Design (Repository)" on page 57.

Step 2: Create a Master Index Database and User (Repository)

Before beginning this step, complete "Step 1: Analyze the Master Index Database Requirements (Repository)" on page 62. After you create the database instance and user, continue to "Step 3: Define Master Index Database Indexes (Repository)" on page 64 if you want to define additional database indexes; otherwise skip to "Step 4: Define Master Index External Systems (Repository)" on page 65.

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 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 (Repository)

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 *Application_name* Database file file or create a new script. Before you begin this step, complete "Step 2: Create a Master Index Database and User (Repository)" on page 63.

▼ To Define an Index

1. **Under the master index project in the Projects window, expand Database Scripts and then open the Create *Application_name* Database 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 (Repository)" on page 65.**

Step 4: Define Master Index External Systems (Repository)

A SQL script is automatically created to insert the systems you specified in the wizard. These statements are provided in the Systems file in the master index project. Before you begin this step, complete "Step 2: Create a Master Index Database and User (Repository)" on page 63 and, optionally, "Step 3: Define Master Index Database Indexes (Repository)" on page 64.

▼ 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 (Repository)" on page 65 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 (Repository)" on page 67.**

Master Index Database Table Description for sbyn_systems (Repository)

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 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.
input_mask	A mask used by the 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.

Column	Description
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 (Repository)

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 wizard creates a stanza in the Code List file (located under the Database Script node of the project) for each code list you specified in the field properties. You need to customize the file by defining the entries for each code list. 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 (Repository)" on page 65.

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 must be customized.

3. **In the first code list stanza, change "module description" in the first line to a brief description of the code type.**

   For example:

```
-- **** PHONTYPE ****
tCode('L', 'PHONTYPE', 'TELEPHONE TYPE'),
```

4 **Create the entries for the module using the following syntax:**

```
tCode('V', 'code', 'code description'),
```

where "code" is the processing code of the data element and "code description" is the description of the element as you want it to appear on the Enterprise Data Manager windows. For example:

```
-- **** PHONTYPE ****
tCode('L', 'PHONTYPE', 'TELEPHONE TYPE'),
tCode('V', 'H', 'HOME'),
tCode('V', 'C', 'CELL'),
tCode('V', 'F', 'FAX'),
tCode('V', 'O', 'OFFICE'),
tCode('V', 'HB', 'HOME BUSINESS'),
```

5 **Repeat the previous two steps for each code list type defined in the file.**

6 **If you specified additional code list fields in the Object Definition file and the Enterprise Data Manager file after the database scripts were generated, add a new stanza for each new code type.**

7 **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')
```

8 **Save and close the file.**

9 **Do one of the following:**

- **If you need to define user code lists, continue to "Step 6: Define Master Index User Code Lists (Repository)" on page 70.**
- **If you need to create a custom database script, skip to "Step 7: Create Custom Master Index Database Scripts (Repository)" on page 71.**
- **If you are ready to create the master index database structure, skip to "Step 8: Create the Master Index Database Structure (Repository)" on page 72.**

▼ 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 must be customized.

3 **In the first code list stanza, change "module description" in the first line to a brief description of the code type.**

For example:

```
-- **** PHONTYPE ****
insert into @codelist values('L', 'PHONTYPE', 'TELEPHONE TYPE')
```

4 **Create the entries for the module using the following syntax:**

```
insert into @codelist values('V', 'code', 'code description')
```

where "code" is the processing code of the data element and "code description" is the description of the element as you want it to appear on the Enterprise Data Manager windows. For example:

```
-- **** PHONTYPE ****
insert into @codelist values('L', 'PHONTYPE', 'TELEPHONE TYPE')
insert into @codelist values('V', 'H', 'HOME')
insert into @codelist values('V', 'C', 'CELL')
insert into @codelist values('V', 'F', 'FAX')
insert into @codelist values('V', 'O', 'OFFICE')
insert into @codelist values('V', 'HB', 'HOME BUSINESS')
```

5 **Repeat the previous two steps for each code list type defined in the file.**

6 **If you specified additional code list fields in the Object Definition file and the Enterprise Data Manager file, add a new stanza for each new code type.**

7 **Save and close the file.**

8 **Do one of the following:**

- **To define user code lists, continue to "Step 6: Define Master Index User Code Lists (Repository)" on page 70.**

- **To create a custom database script, skip to "Step 7: Create Custom Master Index Database Scripts (Repository)" on page 71.**
- **To create the master index database structure, skip to "Step 8: Create the Master Index Database Structure (Repository)" on page 72.**

Step 6: Define Master Index User Code Lists (Repository)

If you specified a value for the`Constrained By` and `User Code` properties 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.

```
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 (Repository)" on page 71. Complete "Step 5: Define Master Index Code Lists (Repository)" on page 67 before beginning this step.

▼ To Define a User Code List

1. **In the Projects window, right-click the Database Script node of the master index project.**
2. **In the Database context menu, select New.**
3. **Enter the name of the script, and then click OK.**
   The file opens in the NetBeans editor.
4. **Use the above sample to define a value for the user code drop-down list and the required format for the dependent fields.**
5. **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).**
6. **Save and close the file.**
7. **Continue to "Step 7: Create Custom Master Index Database Scripts (Repository)" on page 71 or to "Step 8: Create the Master Index Database Structure (Repository)" on page 72 if you do not need to create any custom database scripts.**

Master Index Database Table Description for sbyn_user_code (Repository)

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.
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 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 (Repository)

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 (Repository)" on page 72.**

Step 8: Create the Master Index Database Structure (Repository)

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 (Repository)" on page 63. 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 (Repository)" on page 63).**

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***app_name* **(where** *app_name* **is the name of the master index application) 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 (Repository)

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 (Repository)

Scripts are provided to drop the default database tables and indexes created in "Step 8: Create the Master Index Database Structure (Repository)" on page 72. This is useful while testing the master index application implementation.

▼ To Delete Database Tables (Repository)

1. **Right-click Drop *app_name* Database (where *app_name* is the name of the master index application).**
2. **Select Run.**
3. **On the confirmation dialog box, click OK.**
4. **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 the GlassFish Application Server and is not using a database Adapter to connect to the database, you need to create and configure 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 75
- "Step 2: Create the JDBC Connection Pools" on page 75
- "Step 3: Create the JDBC Resources" on page 76

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 75.

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 75.

1. **Log in to the GlassFish 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 (Repository)" on page 63.
- 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 76.**

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 the JDBC Resources

Before You Begin Create the JDBC connection pool, as described in "Step 2: Create the JDBC Connection Pools" on page 75.

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 75.**

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 75.
- 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`**.