Developing Sun Master Indexes

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–3890–16
December 2008

091111@22749

Contents

Developing Sun Master Indexes

The topics listed here provide procedures, conceptual information, and reference information for using Sun Master Index to design and create a master index application. These topics help you get started with 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 documentation related to implementing a master index application, see "Related Topics" on page 6.

Note that Java CAPS includes two versions of Sun Master Index. Sun Master Index (Repository) is installed in the Java CAPS repository and provides all the functionality of previous versions in the new Java CAPS environment. Sun Master Index is a service-enabled version of the master index that is installed directly into NetBeans. It includes all of the features of Sun 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 Sun Master Data Management (MDM) Suite. This document relates to Sun Master Index 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 the Master Index Database" on page 45
- "Dropping Master Index Database Tables" on page 58
- "Defining the Database Connection Pools" on page 58

More Information

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

- "Master Index Wizard Field Properties and Name Restrictions" on page 25
- "Master Index Database Table Description for sbyn_systems" on page 49
- "Master Index Database Table Description for sbyn_user_code" on page 55

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 Sun Master Indexes*
- *Configuring Sun Master Indexes*
- *Understanding Sun Master Index Configuration Options*
- *Understanding Sun Master Index Processing*
- *Understanding the Master Index Match Engine*
- *Understanding the Master Index Standardization Engine*
- *Working With the Master Index Data Manager*
- *Analyzing and Cleansing Data for Sun Master Index*
- *Loading the Initial Data Set for a Sun Master Index*
- *Maintaining Sun Master Indexes*
- *Broadcasting Master Index Updates to External Systems*

Master Index Development Process Overview

This document provides instructions for creating a master index application using Sun Master Index. It includes creating the master index application framework, creating custom Java code, generating the application files, and creating the database. It also includes performing a data analysis using the framework created for the master index application.

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 Sun Master Index application within that project (described in "Creating a Master Index Application" on page 10).

3. Define the object structure, operating environment, and certain runtime characteristics using the wizard (described in "Creating a Master Index Application" on page 10).
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" on page 37).
5. Customize the configuration files (instructions for this step are provided in a separate document, *Configuring Sun Master Indexes*).
6. Generate the master index application (described in "Generating the Master Index Application" on page 38).
7. Create the database (described in "Creating the Master Index Database" on page 45).
   - 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 Sun Master Index*.
9. Build and deploy the project.
10. Define security for the Master Index Data Manager (MIDM) (described in "Defining Master Index Data Manager Security" in *Maintaining Sun Master Indexes*.
11. Load the initial data set into the master index database (this is described in a separate document, *Loading the Initial Data Set for a Sun Master Index*).

The Master Index Framework and the Runtime Environment

Note – MySQL is only supported in Java CAPS 6 Update 1.

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 Master Index 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 MySQL, Oracle or SQL Server database and a database script generated directly from object.xml. 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 Master Index Data Manager

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

- The fields that appear on the MIDM 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 EJB

When you generate the master index application, two additional projects are created, an EJB project and a web application project. The EJB project includes several Java classes that are common to all master index applications as well as a set of custom classes that are specific to the type of object you are indexing. These are based on object.xml. You can use these methods to specify how data is processed into the database. You can also call these methods from Business Processes.

From XML to the Runtime Environment

The information you specify in the master index configuration files is read at runtime when the domain is started. The only exception is object.xml, 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 MIDM and any eWays or Binding Components connected to the application for the changes to take effect. Use caution when modifying these files; changing these files after moving to production might result in loss of data integrity or unexpected weighting and matching results.

Before You Begin Developing a Master Index

Creating a master index application requires in-depth analyses of your business requirements, legacy data, and data processing requirements. After the initial analysis, you can plan and design how you will create the master index application framework and how you will customize that configuration after creating the framework. After creating the master index application, you should perform an in-depth data analysis using the tools provided. This analysis will help you define matching and standardization rules and logic. .

The 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

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

Planning a Master Index Project

Before you create the Sun 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

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

Creating a Master Index Application

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

When you create a new Master Index project in NetBeans, the wizard for creating a master index application automatically appears.

▼ To Create a Project and Start the Wizard

1 On the NetBeans toolbar, click New Project.

The New Project wizard appears.

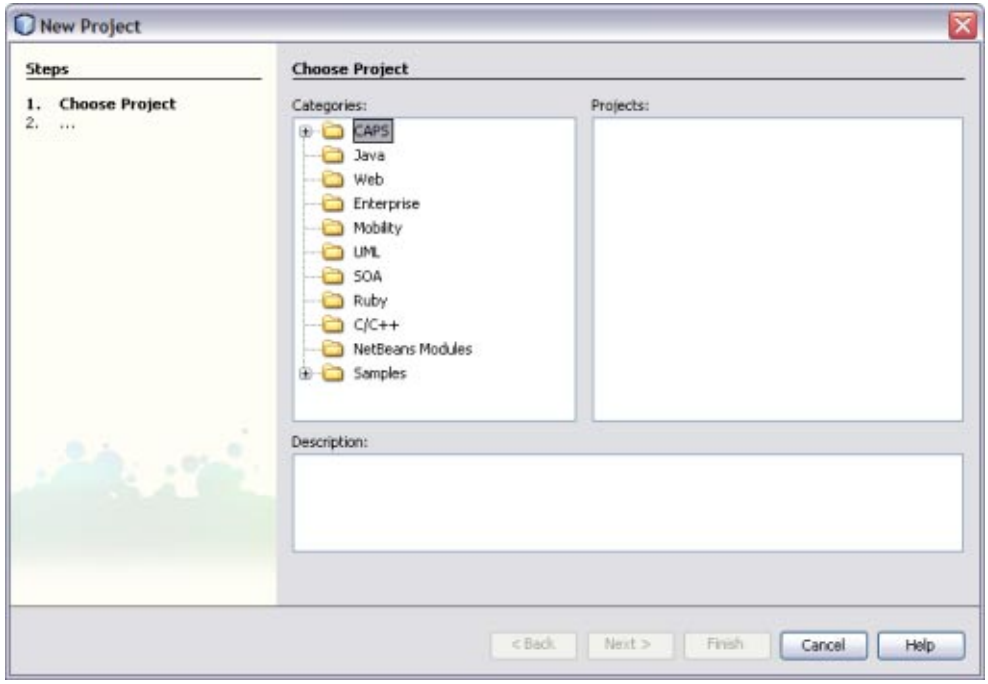

FIGURE 1 New Project Wizard

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

3 **Under Projects, select Master Index Application and then click Next.**

The Project Name page appears.

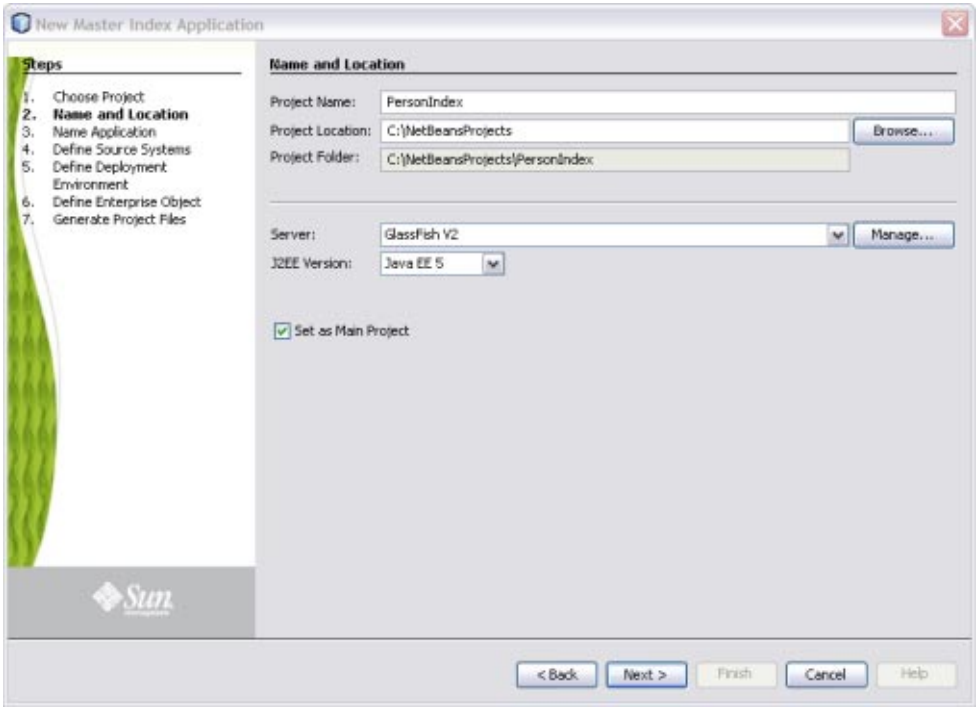

FIGURE 2 Project Name and Location

4 **Enter the project name and the path where you want to store the project files in the upper portion of the window.**

5 **Enter the application server and Java version you are using in the lower portion of the window.**

6 **To make the Master Index project a main project, select Set as Main Project.**

7 **Click Next.**

8 **Continue to "Step 2: Name the Master Index Application" on page 12.**

Step 2: Name the Master Index Application

Before you can configure the new master index application, you must give the master index application a unique name. It is a good practice to give each master index application a unique name you create because certain components within the application server are based on the application name. Dependent components include the data source for the JDBC connection and the outbound Topic (if one is used).

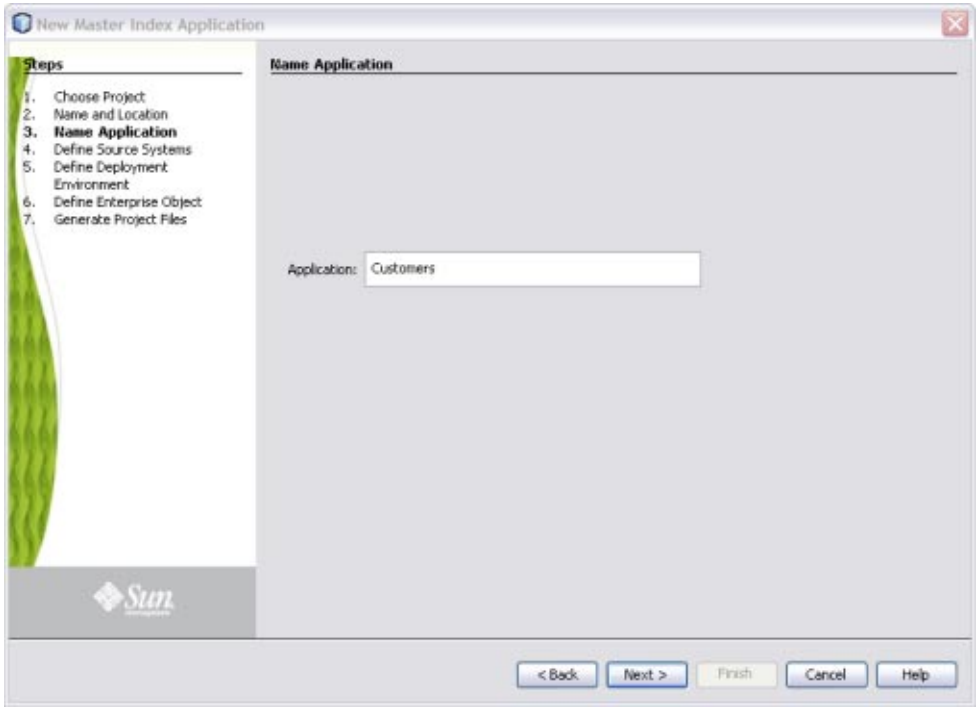

FIGURE 3 Name Application Window

▼ To Name the Master Index Application

1 **Complete "Step 1: Create a Project and Start the Wizard" on page 10.**

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

Note – Give the application the same name you will give to the parent object when you define the object structure.

The Define Source Systems window appears.

3 **Continue to "Step 3: Define Source Systems" on page 14.**

Step 3: Define Source Systems

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.

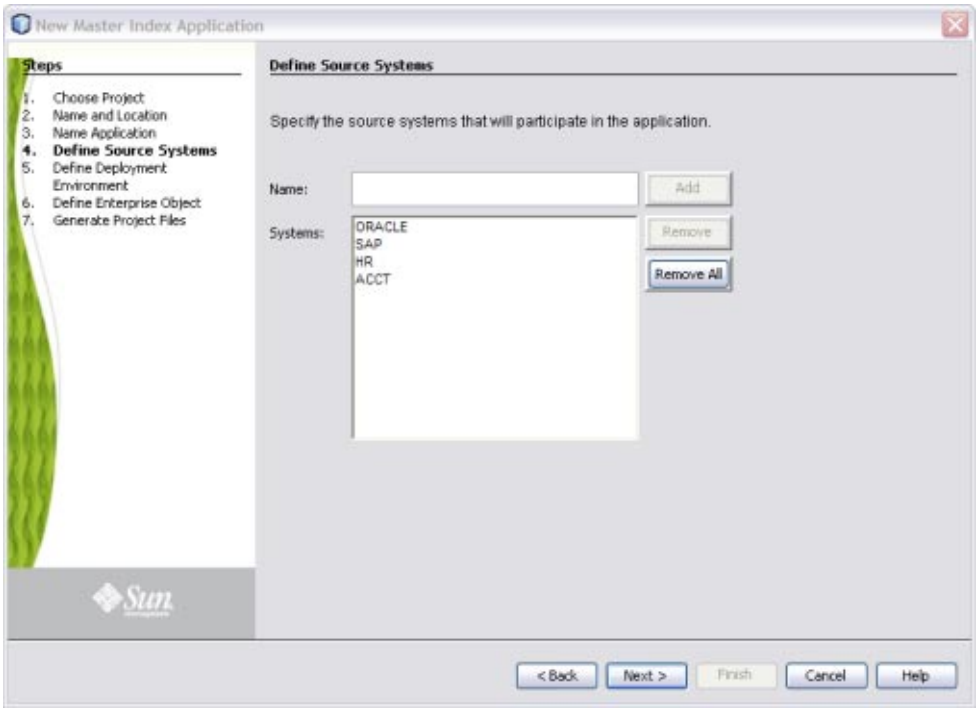

FIGURE 4 Define Source Systems

▼ To Define Source Systems

1. **Complete "Step 2: Name the Master Index Application" on page 12.**

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 update.xml 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" on page 15.**

Step 4: Define the Deployment Environment

Note – MySQL is only supported in Java CAPS 6 Update 1.

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

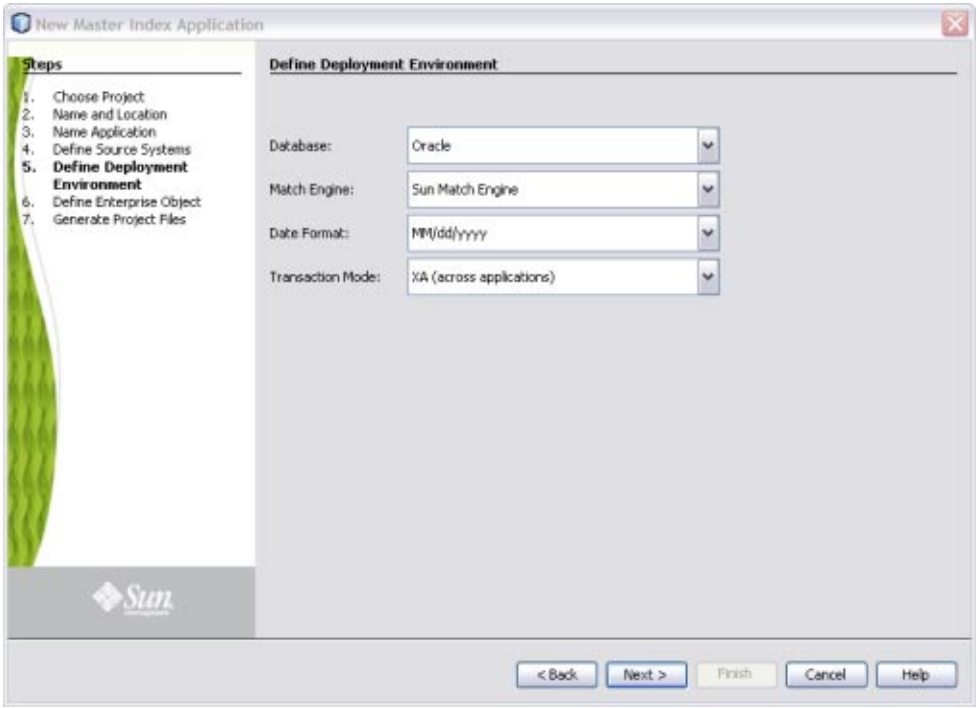

FIGURE 5 Define Deployment Environment

▼ To Define the Deployment Environment

1. **Complete "Step 3: Define Source Systems" on page 14.**

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 MySQL, 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 MIDM. You can select MM/dd/yyyy, dd/MM/yyyy, or yyyy/MM/dd.
   - **Transaction Mode** – An indicator of whether transactions are distributed. Select one of the following options:
     - **XA (across applications)**, if the transactions will be distributed across multiple applications

- **XA (within application only)**, if the transactions will be distributed just across the master index application.
- **Local**, if the transactions will not be distributed.

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

4 **Continue to "Step 5: Define Parent and Child Objects" on page 17.**

Step 5: Define Parent and Child Objects

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 17
- "Creating Objects from a Template" on page 19
- "Deleting an Object from the Structure" on page 20

Complete "Step 4: Define the Deployment Environment" on page 15 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 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.

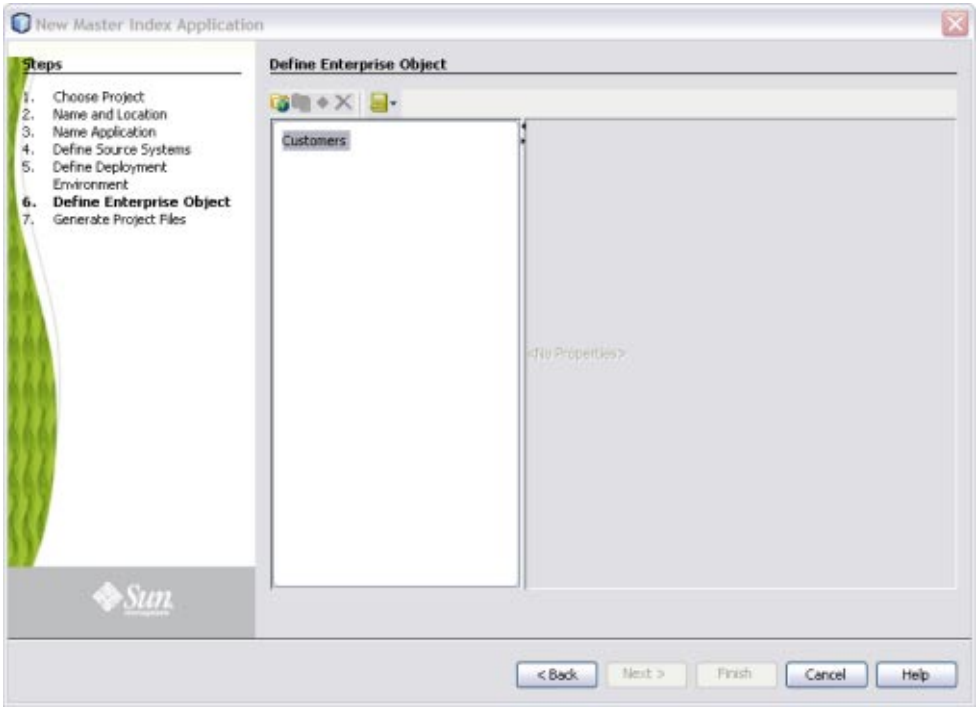

FIGURE 6 Define Enterprise Object

▼ 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" on page 12.

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" on page 20.**

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.

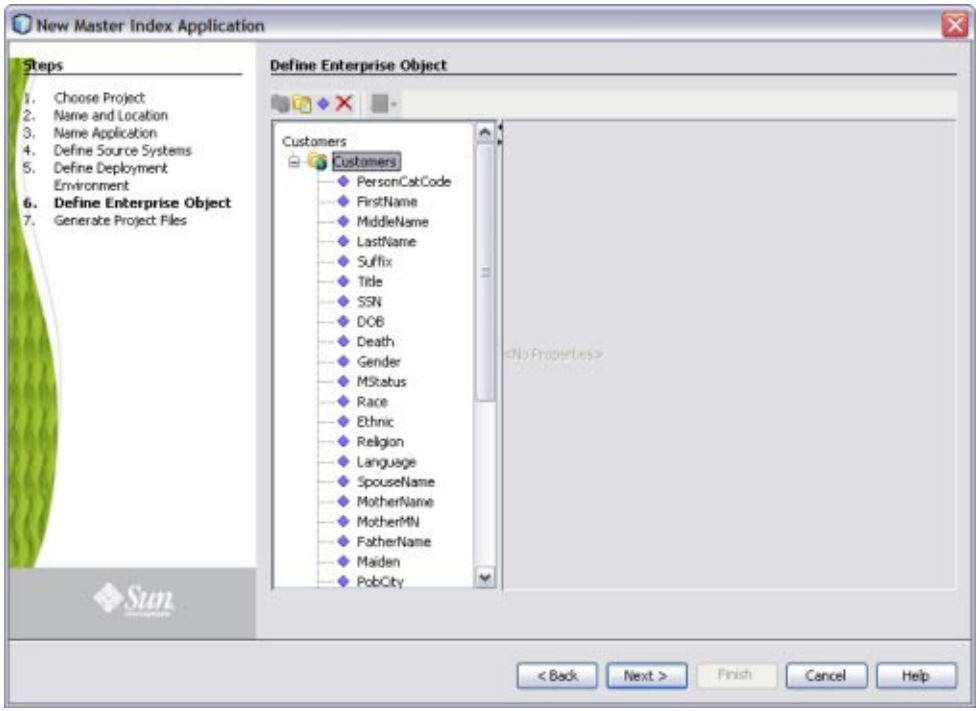

FIGURE 7 Define Enterprise Object

▼ 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" on page 20.**

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

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 21

Adding a Field

If you created an empty object in "Step 5: Define Parent and Child Objects" on page 17, 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" on page 17.**
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" on page 25.
4. **Continue to "Configuring Field Properties" on page 21.**

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 21.**
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" on page 26.**

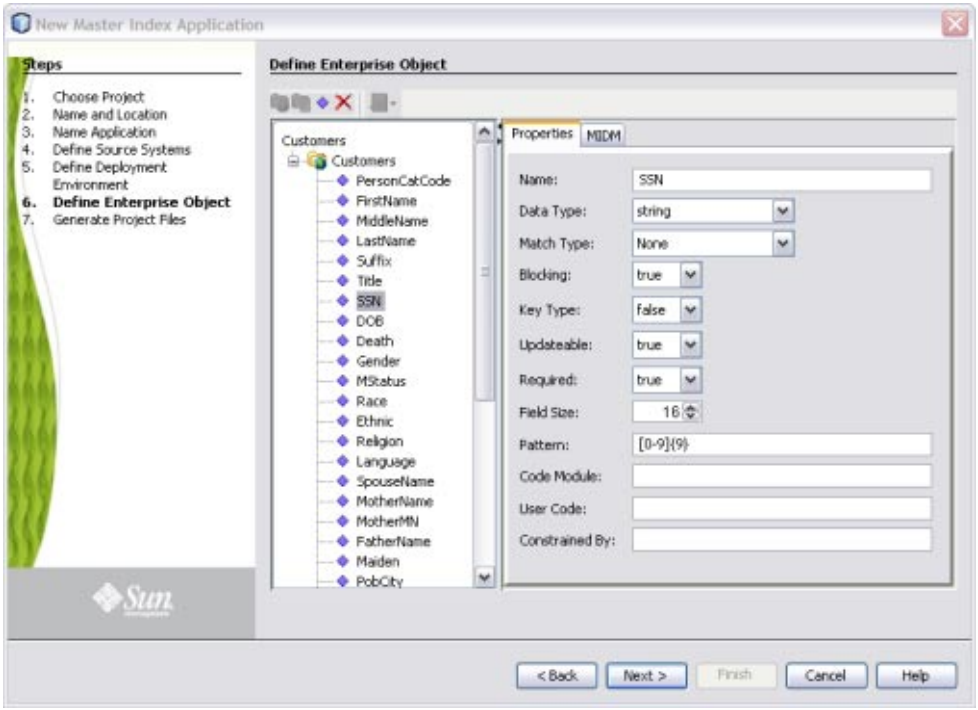

FIGURE 8 Field Properties

4 **On the right side of the window, click the MIDM tab, and then modify the value of any of the properties listed in "Master Index Wizard MIDM Field Properties" on page 28.**

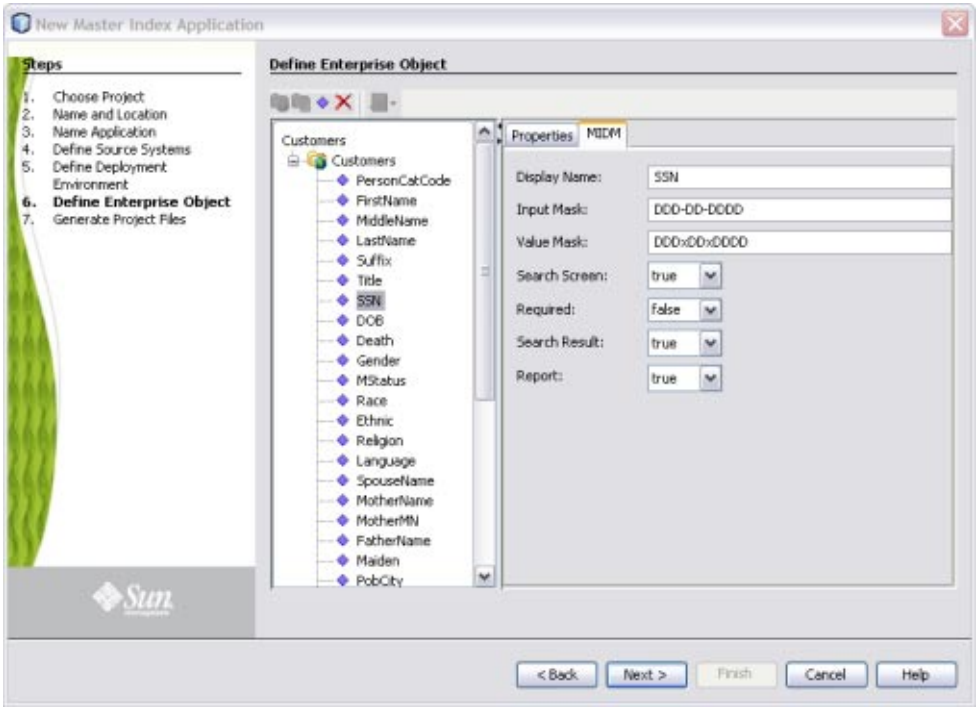

FIGURE 9 Field MIDM Properties

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" on page 24.**

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

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. 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 Sun Master Indexes* .

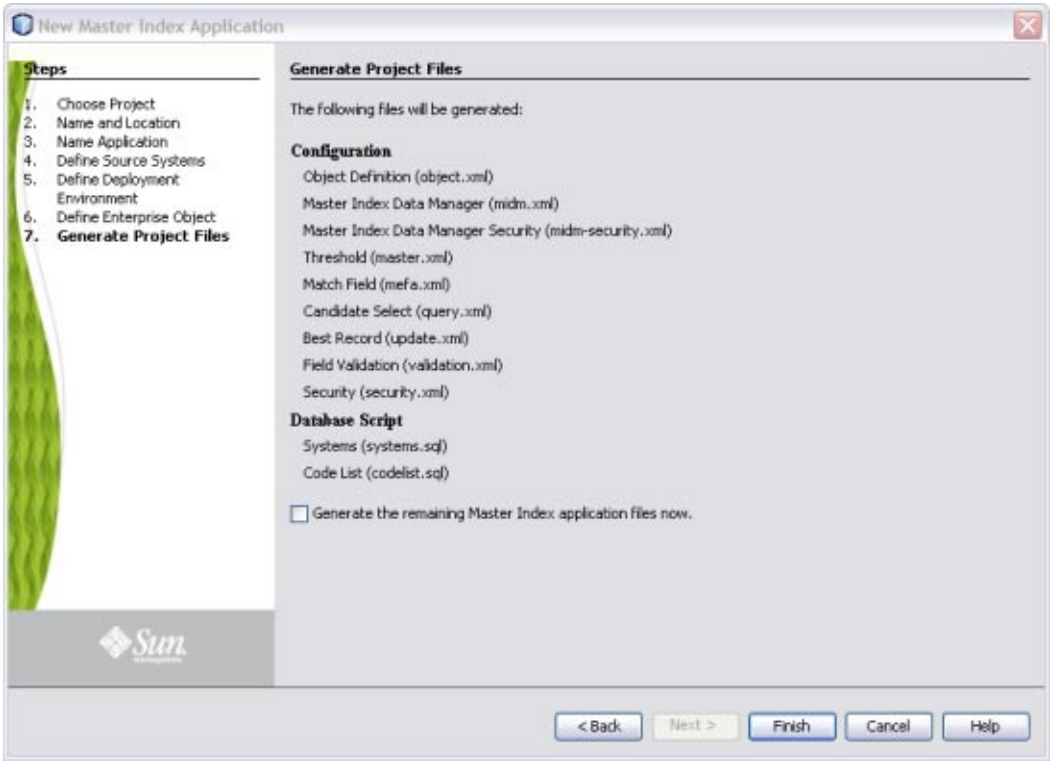

FIGURE 10 Generate Project Files

▼ To Generate the Configuration Files

1. **Complete "Step 6: Define the Fields for Each Object" on page 20.**
2. **Verify that all of the information you have entered is complete and correct.**
3. **On the Generate Configuration Files window, click Finish.**
   The configuration files are generated, and are stored in the eGate Repository.
4. **Continue to "Step 8: Review the Configuration Files" on page 25.**

Step 8: Review the Configuration Files

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 Sun Master Indexes* for information about configuration tasks. For information about the configuration files and configurable options, see *Understanding Sun Master Index Configuration Options* .

Master Index Wizard Field Properties and Name Restrictions

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 MIDM, 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

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.

- Sun 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, Sun 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 Sun Master Index Processing* .
- In field names, do not use characters or names restricted by Java, XML, or the database platform being used.

Master Index Wizard General Field Properties

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`**.
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 mefa.xml, including the match string. The match types in mefa.xml might differ from the wizard match types. See *Understanding Sun Master Index Configuration Options* for information about the available options for this field and how the wizard match types correlate to mefa.xml 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.

Property	Description
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 MIDM 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 MIDM.
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 MIDM, 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.
Code Module	The identification code for the drop-down list that appears for this field in the MIDM. **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.

Property	Description
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 MIDM Field Properties

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

Property	Description
Display Name	The name of the field as it will appear on the MIDM.
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" on page 26). The following character types can be used.

Property	Description
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 MIDM. 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 MIDM 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 MIDM 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 MIDM. 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 MIDM. Specify **`true`** to display the field on the reports; otherwise specify **`false`**.

Custom Plug-ins for Master Index Custom Transaction Processing

You can add custom processing to the master index application by adding your own code to new Java classes in the EJB project for the master index application. These classes are called custom plug-ins. 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 Sun Master Index Configuration Options* .

- "Master Index Update Policy Plug-ins" on page 30 – Define custom processing logic to perform against the resulting record of a transaction before it is stored in the database.

- "Master Index Field Validation Plug-ins" on page 31 – Define validations to perform against specific fields, such as checking the local ID length and format.
- "Master Index Field Masking Plug-ins" on page 32 – Define how the values for sensitive fields are hidden on the MIDM from users who do not have permission to view them.
- "Master Index Match Processing Logic Plug-ins" on page 32 – Define custom logic based on predefined decision points for how records are matched during a transaction.
- "Master Index Custom Plug-in Exception Processing" on page 33 – Define how exceptions are handled by the custom plug-ins you create.

Master Index Update Policy Plug-ins

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

Update policies are specified in the *UpdatePolicy* section of update.xml, and there are several different types. Each policy modifies an enterprise object (class `com.sun.mdm.index.objects.EnterpriseObject`) and must implement `com.sun.mdm.index.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.sun.mdm.index.master.UserException` and `com.sun.mdm.index.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 update.xml, 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 update.xml, 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 update.xml, 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 update.xml, 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 update.xml, 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 update.xml, 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 update.xml, enter the fully qualified name of this custom plug-in.

Master Index Field Validation Plug-ins

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

- The custom validation classes must implement `com.sun.mdm.index.objects.validation.ObjectValidator`.
- The exception thrown is `com.sun.mdm.index.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 Sun Master Index Configuration Options* .

Master Index Field Masking Plug-ins

There might be instances where you want to mask certain data in records from general users of the Master Index 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 MIDM. 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 midm.xml).

Master Index Match Processing Logic Plug-ins

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

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

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 master.xml, 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.sun.mdm.index.master`. For information about where the decision points are reached in the processing logic and how to change the logic, see *Understanding Sun Master Index Processing* . 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.sun.mdm.index.objects.SystemObject`
- `com.sun.mdm.index.objects.EnterpriseObject`
- `com.sun.mdm.index.objects.exception.ObjectException`
- `com.sun.mdm.index.master.ExecuteMatchLogics`
- `com.sun.mdm.index.master.CustomizationException`

Custom Match Processing Configuration

If you create a custom plug-in that defines custom processing logic for the execute match methods, you must specify those custom plug-ins in master.xml in the master index project. If you create a plug-in for customizing logic in the execute match methods used by the back-end, specify the name of that class in the *logic-class* element. If you create a plug-in for the MIDM, specify the name of that class in the *logic-class-gui* element. For example:

```
<logic-class>com.sun.mdm.index.user.CustomCollaboration</logic-class>
<logic-class-gui>com.sun.mdm.index.user.CustomMIDM</logic-class-gui>
```

For more information about master.xml, see *Understanding Sun Master Index Configuration Options* .

Master Index Custom Plug-in Exception Processing

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

For more information about these exception classes, see the Javadocs for Sun Master Index.

Custom Plug-Ins for Master Index Custom Components

Sun 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" on page 34 – Define how the single best record (SBR) is generated.
- "Master Index Query Builder Plug-ins" on page 34 – Define how queries are performed against the master index database.
- "Master Index Block Picker Plug-ins" on page 35 – Define how the blocks of fields in a blocking query are selected during a query.
- "Master Index Pass Controller Plug-ins" on page 35 – Define how the master index application determines whether to perform additional match passes against a record.
- "Match Engine Plug-ins" on page 36 – Define logic that connects to a match engine other than the Master Index Match Engine.
- "Standardization Engine Plug-ins" on page 36 - Define logic that connects to a standardization engine other than that provided by the Master Index Standardization Engine.
- "Phonetic Encoders Plug-ins for a Master Index" on page 36 – Define logic that connects to phonetic encoders other than those provided.

Master Index Survivor Calculator Plug-ins

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

Master Index Query Builder Plug-ins

The query builder defines the different types of queries that can be used in the master index application. You can implement custom queries using custom plug-ins. To create a new query builder, you must define a class that extends the base abstract `com.sun.mdm.index.querybuilder.QueryBuilder` and then specify that class in a

query-builder element in query.xml. The exception thrown is `com.sun.mdm.index.querybuilder.QueryBuilderException`. The following methods must be implemented.

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

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

Master Index Block Picker Plug-ins

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

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

Master Index Pass Controller Plug-ins

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

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

Match Engine Plug-ins

You can define classes to connect to a custom match engine instead of the Master Index Match Engine. Specify the names of the custom classes you create in the *matcher-api* and *matcher-config* elements of mefa.xml. Follow these guidelines when implementing custom match engine classes.

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

You can also define custom comparison functions to plug in to the Master Index Match Engine. For more information, see *Understanding the Master Index Match Engine* .

Standardization Engine Plug-ins

You can define classes to connect to a custom standardization engine instead of the Master Index Standardization Engine. Specify the names of the custom classes you create in the *standardizer-api* and *standardizer-config* elements of mefa.xml. Follow these guidelines when implementing custom standardization engine classes.

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

You can also define custom standardization rules to plug in to the Master Index Standardization Engine. For more information, see *Understanding the Master Index Standardization Engine*.

Phonetic Encoders Plug-ins for a Master Index

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

Implementing Master Index Custom Plug-ins

Custom plug-ins allow you to incorporate custom code into your master index application. When you create a custom plug-in for a master index application, you define the package and class names. To incorporate the plug-ins into the master index application, you then specify the fully qualified class name of the class in the appropriate configuration file. For example, if you create a custom plug-in named `MergePolicy` in the package `com.sun.mdm.index.update`, the value to enter for the class in update.xml is `com.sun.mdm.index.update.MergePolicy`.

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

- "Custom Plug-ins for Master Index Custom Transaction Processing" on page 29
- "Custom Plug-Ins for Master Index Custom Components" on page 34

You create a custom plug-in by composing Java code in a source file using the NetBeans editor. Create the file in the Source Package folder of the EJB project associated with the master index application. Before you begin, determine the name of the package and class for the plug-in.

▼ To Create Custom Plug-ins

1. **In the NetBeans Projects window, expand *project_name*-ejb (where *project_name* is the name of the main master index project.**
2. **To create a new package for the custom plug-in files, do the following:**
   a. **Right-click Source Packages, point to New, and then click Java Package.**
   b. **Enter the package name and location, and then click Finish.**
3. **To create the plug-in, do the following for each source file:**
   a. **Under Source Packages, right-click the name of the package to which the new class will belong.**
   b. **Point to New, and then click Java Class.**
   c. **Enter the class name, verify the remaining fields, and then click Finish.**
      The new file appears in the NetBeans editor.
   d. **Create the custom processing rules using Java code.**

e. **When you are finished coding, save and close the file.**

f. **In the Projects window, right-click the file and then click Compile.**

The output and status of the compiling process appears in the bottom panel.

Note – If you do not compile a custom plug-in file, building the master index project will compile the files.

---

4 **Specify the name of the custom plug-in in the appropriate master index configuration file.**

Note – If you modify a custom plug-in, be sure to recompile the file and regenerate the application to incorporate the changes. If the project has already been built, rebuild the project to add the custom plug-in to the EJB .jar file.

Generating the Master Index Application

Before you generate the application, review the configuration files and make any necessary modifications (see *Configuring Sun Master Indexes* 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 Master Index Files.**

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

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

▼ To Regenerate the Application

1. **Save any configuration changes to the master index project.**
2. **If you are using the command-line reports, back them up to a different directory. Generating the application overwrites the existing report client.**
3. **Right-click the master index application in the Projects window.**
4. **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.
5. **On the NetBeans toolbar, click Save.**

Master Index Database Scripts and Design

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

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

Master Index Database Scripts

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 `create.sql`, `systems.sql`, `codelist.sql`, and `drop.sql`. 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

Note – MySQL and Microsoft SQL Server 2008 are only supported in Java CAPS 6 Update 1 or later.

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 database platform documentation. The person responsible for the database configuration should be a 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 MySQL Enterprise Server, SQL Server 2005 or 2008, or on Oracle 9*i*, 10*g* or 11*g*. 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 documentation that came with your database server 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 database vendors for a typical installation. Depending on the size of the database and expected volume, you should increase these recommendations as needed. See the documentation for your database for more information and for supported operating systems.

MySQL Database

For information and tips about installing a MySQL database for Sun Master Index, see the Chapter 2 of the MySQL 5.1 Reference Manual (`http://dev.mysql.com/doc/refman/5.1/en/installing.html`).

Oracle Database

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

- Windows 2000 SP3 or later, Windows XP SP2, or Windows Server 2003
- Pentium 266 or later

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

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

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

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

Microsoft SQL Server

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

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

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

Master Index Database Structure

The master index database contains some common tables that are created for all implementations and some that are customized for each implementation. The common tables include standard database system 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 Sun Master Index Processing* .

Designing the Master Index Database

Note – MySQL is only supported in Java CAPS 6 Update 1 or later.

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 MySQL, 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 MIDM, 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 MIDM. 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 a database instance using the provided configuration tools or command line functions. Use the tools provided by the database vendor 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 MIDM.

```
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

Note – MySQL is only supported in Java CAPS 6 Update 1 or later.

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 MySQL, 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 MIDM fields and to create drop-down lists for MIDM fields. System information is required in order to add records to the master index application.

Follow these steps to create the database.

- "Step 1: Analyze the Master Index Database Requirements" on page 45
- "Step 2: Create a Master Index Database and User" on page 46
- "Step 3: Define Master Index Database Indexes" on page 47
- "Step 4: Define Master Index External Systems" on page 48
- "Step 5: Define Master Index Code Lists" on page 50
- "Step 6: Define Master Index User Code Lists" on page 55
- "Step 7: Create Custom Master Index Database Scripts" on page 56
- "Step 8: Create the Master Index Database Structure" on page 57
- "Step 9: Specify a Starting EUID for a Master Index" on page 57

You can also delete the database for testing purposes using the supplied script. See "Dropping Master Index Database Tables" on page 58 for more information.

Step 1: Analyze the Master Index Database Requirements

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

A database administrator who is familiar with your data and processing requirements should perform this task. After this task is complete, continue to "Step 2: Create a Master Index Database and User" on page 46.

For additional information and guidelines about how to set up your database, see "Master Index Database Scripts and Design" on page 39.

Step 2: Create a Master Index Database and User

Note – MySQL is only supported in Java CAPS 6 Update 1 or later.

Before beginning this step, complete "Step 1: Analyze the Master Index Database Requirements" on page 45. After you create the database instance and user, continue to "Step 3: Define Master Index Database Indexes" on page 47 if you want to define additional database indexes; otherwise skip to "Step 4: Define Master Index External Systems" on page 48.

For this step you need to create a database in which the master index database instance will be created. Use the tools provided by your database vendor 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 MIDM 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.

For MySQL, you can grant the user access to all permissions, or you can assign the individual permissions listed below. For example:

```
CREATE USER 'username'@'server_name' IDENTIFIED BY 'password';
GRANT ALL ON database TO 'username'@'server_name';
GRANT SELECT, INSERT ON database TO 'username'@'server_name';
```

where *username* is the login ID for the user, *server_name* is the name of the database server, *password* is the login password, and *database* is the database name.

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

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

Step 3: Define Master Index Database Indexes

To optimize data processing in the master index application, you can define additional indexes for the database tables that store object data. Best practice is to define indexes for each field used

for searching, blocking, or matching. You can define these indexes in `create.sql` or create a new script. Before you begin this step, complete "Step 2: Create a Master Index Database and User" on page 46.

▼ To Define an Index

1. **Do one of the following:**
   - **Under the master index project in the Projects window, expand Database Scripts and then open `create.sql` in the NetBeans editor.**
   - **On your computer, navigate to *project_home*`/src/DatabaseScript`, where *project_home* is the location of the master index project files. Open `create.sql` in a text editor.**
2. **Do any of the following:**
   - **Remove an existing index definition (not recommended).**
   - **Create new index definitions for the required fields.**
   - **Modify an existing index definition.**
3. **Save and close the file.**
4. **Continue to "Step 4: Define Master Index External Systems" on page 48.**

Step 4: Define Master Index External Systems

A SQL script is automatically created to insert the systems you specified in the wizard. These statements are provided in `systems.sql` in the master index project. Before you begin this step, complete "Step 2: Create a Master Index Database and User" on page 46 and, optionally, "Step 3: Define Master Index Database Indexes" on page 47.

▼ To Define an External System

1. **Do one of the following:**
   - **Under the master index project in the Projects window, expand Database Scripts and then open `systems.sql` in the NetBeans editor.**
   - **On your computer, navigate to *project_home*`/src/DatabaseScript`, where *project_home* is the location of the master index project files. Open `systems.sql` in a text editor.**

2 **For each system, create an INSERT statement using the column descriptions in "Master Index Database Table Description for sbyn_systems" on page 49 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', 8, '[0-9]{8}', 'DD-DDD-DDD',
'DD^DDD^DDD', sysdate, 'admin');
```

3 **Delete any default INSERT statements you do not need.**

4 **Save and close the file.**

5 **Continue to "Step 5: Define Master Index Code Lists" on page 50.**

Master Index Database Table Description for sbyn_systems

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

Column	Description
systemcode	The unique processing code of the system. This field accepts any of the following characters: ■ ! _ ~ ( ) { } + \Q # $ % & : ; - / ■ a-z ■ A-Z ■ 0-9 ■ þ ÿ Þ ß 'à-ö ø-ý À-Ö Ø-Ý
description	A brief description of the system, or the system name.
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.

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

Step 5: Define Master Index Code Lists

Note – MySQL is only supported in Java CAPS 6 Update 1 or later.

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`codelist.sql` (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" on page 48.

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 MySQL

1 **Do one of the following:**

- **Under the master index project in the Projects window, expand Database Scripts and then open `codelist.sql` in the NetBeans editor.**
- **On your computer, navigate to *project_home*`/src/DatabaseScript`, where *project_home* is the location of the master index project files. Open `codelist.sql` in a text editor.**

2 **Scroll to the first line that begins with "— ****".**

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 tCodeList values('L', 'PHONTYPE', 'TELEPHONE TYPE');
```

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

```
insert into tCodeList 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 Master Index Data Manager windows. For example:

```
-- ****         PHONTYPE   ****
insert into tCodeList values('L', 'PHONTYPE', 'TELEPHONE TYPE');
insert into tCodeList values('V', 'H', 'HOME');
insert into tCodeList values('V', 'C', 'CELL');
insert into tCodeList values('V', 'F', 'FAX');
insert into tCodeList values('V', 'O', 'OFFICE');
insert into tCodeList 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 object.xml and midm.xml after the database scripts were generated, add a new stanza for each new code type.**

7 **Save and close the file.**

8 **Do one of the following:**

- **If you need to define user code lists, continue to "Step 6: Define Master Index User Code Lists" on page 55.**
- **If you need to create a custom database script, skip to "Step 7: Create Custom Master Index Database Scripts" on page 56.**
- **If you are ready to create the master index database structure, skip to "Step 8: Create the Master Index Database Structure" on page 57.**

▼ To Customize Common Table Data for Oracle

1 **Do one of the following:**

- **Under the master index project in the Projects window, expand Database Scripts and then open `codelist.sql` in the NetBeans editor.**
- **On your computer, navigate to *project_home*`/src/DatabaseScript`, where *project_home* is the location of the master index project files. Open `codelist.sql` in a text 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 Master Index 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 object.xml and midm.xml 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" on page 55.**
- **If you need to create a custom database script, skip to "Step 7: Create Custom Master Index Database Scripts" on page 56.**
- **If you are ready to create the master index database structure, skip to "Step 8: Create the Master Index Database Structure" on page 57.**

▼ To Customize Common Table Data for SQL Server

1 **Do one of the following:**

- **Under the master index project in the Projects window, expand Database Scripts and then open `codelist.sql` in the NetBeans editor.**

- **On your computer, navigate to** *project_home*`/src/DatabaseScript`**, where** *project_home* **is the location of the master index project files. Open** `codelist.sql` **in a text 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 Master Index 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 object.xml and midm.xml, 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" on page 55.**
- **To create a custom database script, skip to "Step 7: Create Custom Master Index Database Scripts" on page 56.**

- **To create the master index database structure, skip to "Step 8: Create the Master Index Database Structure" on page 57.**

Step 6: Define Master Index User Code Lists

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" on page 55. Complete "Step 5: Define Master Index Code Lists" on page 50 before beginning this step.

▼ To Define a User Code List

1. **Create a new SQL file to contain the user code SQL statements.**
2. **Use the above sample to define a value for the user code drop-down list and the required format for the dependent fields.**
3. **Repeat the above step for each drop-down list value and type (for example you might have one list for credit cards and another for postal codes and their corresponding cities).**
4. **Save and close the file.**
5. **Continue to "Step 7: Create Custom Master Index Database Scripts" on page 56 or to "Step 8: Create the Master Index Database Structure" on page 57 if you do not need to create any custom database scripts.**

Master Index Database Table Description for sbyn_user_code

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

Column Name	Description
code_list	The code list name of the user code type (using the credit card example above, this might be similar to "CREDCARD"). This column links the values for each list.
code	The processing code of each user code element.
description	A brief description or name for the user code element. This is the value that appears in the drop-down list.
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 MIDM to add punctuation to the constrained field. For example, the input mask **DD-DDD-DDD** inserts a hyphen after the second and fifth characters in an 8-digit ID. If you use an input mask, you should also use a value mask to strip the punctuation for database storage (see below). The following character types can be used. ■ **D** – Numeric character ■ **L** – Alphabetic character ■ **A** – Alphanumeric character
value-mask	A mask used to strip any extra characters that were added by the input mask for database storage. The value mask is the same as the input mask, but with an "x" in place of each punctuation mark. Using the input mask described above, the value mask is **DDxDDDxDDD**. This strips the hyphens before storing the ID.

Step 7: Create Custom Master Index Database Scripts

You can insert additional information into the database by creating a custom script. For information about the structure of the master index database, see *Understanding Sun Master Index Processing* .

▼ To Create a Custom Script

1. **Create a new text file to contain the SQL statements.**
2. **In the text editor, create the SQL script to insert the custom data.**
3. **Save and close the file.**
4. **Continue to "Step 8: Create the Master Index Database Structure" on page 57.**

Step 8: Create the Master Index Database Structure

Note – MySQL is only supported in Java CAPS 6 Update 1 or later.

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. **Open a standard SQL editor or SQL command line and connect to the master index database using the user you created in "Step 2: Create a Master Index Database and User" on page 46. For MySQL, you can run the files from the mysql prompt.**

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

   - **Open `create.sql` in the NetBeans editor, copy and paste the entire text of the file into the SQL editor, and run the script against the database.**
   - **From the SQL editor or command line, run `create.sql` directly. The file is located at** *project_home*`/src/DatabaseScript`**.**

     Tip – To run this script for Oracle or SQL Server, the command is `@create.sql`. For MySQL, the command is `source create.sql`.

3. **Repeat the previous two steps for `systems.sql`, `codelist.sql`, and any custom scripts you created.**

Step 9: Specify a Starting EUID for a Master Index

By default, the EUIDs assigned by the master index application start with "0", with padded zeroes added to the left to make the EUID number the correct length (for more information, see *Understanding Sun Master Index Configuration Options*). 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';
```

Dropping Master Index Database Tables

Scripts are provided to drop the default database tables and indexes created in "Step 8: Create the Master Index Database Structure" on page 57. This is useful while testing the master index application implementation.

▼ To Delete Database Tables

1. **Open a standard SQL editor and connect to the master index database using the user you created in "Step 2: Create a Master Index Database and User" on page 46.**
2. **Open `drop.sql` in the NetBeans editor.**
3. **Copy and paste the entire text of `drop.sql` into the SQL editor.**
4. **Run the script against the database.**
5. **If the database is running on the Oracle 10*g* platform, open a SQL prompt and run the following command.**

```
PURGE RECYCLEBIN;
```

Defining the Database Connection Pools

Each master index application requires two database connection pools; one for the master controller and one for the sequence manager. To set up the connection pools, you need to create the connection pools and then define a JDBC resource for each. 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 Sun Java System Application Server Admin Console.

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

- "Step 1: Add the MySQL or Oracle Driver to the Application Server" on page 59
- "Step 2: Create two JDBC Connection Pools" on page 59
- "Step 3: Create the JDBC Resources" on page 62

Step 1: Add the MySQL or Oracle Driver to the Application Server

Note – MySQL is only supported in Java CAPS 6 Update 1 or later.

If you are using a MySQL or 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.

For Oracle, you can either install the 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 Sun Java System Application Server.

For MySQL, download and extract the latest MySQL Connector/J — for connecting to MySQL from Java. You can access the driver at the MySQL downloads page (`http://dev.mysql.com/downloads`). Copy `mysql-connector-java-5.1.6-bin.jar` to *app_server_home*`\lib`.

Once the driver is installed or copied, continue to "Step 2: Create two JDBC Connection Pools" on page 59.

Step 2: Create two JDBC Connection Pools

Note – MySQL is only supported in Java CAPS 6 Update 1. You only need to create the sequence connection pool if you are using Update 1.

The JDBC connection pools provide connections to the master index database. You need to create two connection pools that are configured in the same way.

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 or MySQL database, add the database driver to the application server environment, as described in "Step 1: Add the MySQL or Oracle Driver to the Application Server" on page 59.

1. **Log in to the Sun Java System Application Server Admin Console.**

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

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

3. **On the Create Connection Pool page, click New.**
4. **In the Name field, enter a name for the connection pool.**
5. **In the Resource Type field, select the Java class for the type of transactions the master index application processes.**
   - `javax.sql.DataSource` – Use this class if the master index application is using local transactions only.
   - `javax.sql.XADataSource` – Use this class if the master index application transactions are distributed, either within the application or across applications.
   - `javax.sql.ConnectionPoolDataSource` – Use this class if the master index application is using local transactions only. This class provides possible performance improvements.
6. **In the Database Vendor field, select the database platform used by the master index database.**
7. **Click Next.**
8. **In the DataSource Classname field, accept the default class or enter a new one to use.**
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 according to whether the application supports transactional processing.**

Note – Make sure you configure Transaction properties to match the transaction mode you specified for the master index application and the connection pool Resource Type you selected.

---

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).**
    - **For Oracle:**
      - **URL** – The URL that points to the database. The syntax of the URL is `jdbc:oracle:thin:@`*host*`:`*port*`:`*database_name*.

Note – If you are using a third-party JDBC driver, refer to the documentation for that driver for information about the URL.

- **user** – The login ID for the user you created in "Step 2: Create a Master Index Database and User" on page 46.
- **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`**.

- **For MySQL:**
  - **URL** – The URL that points to the database. The syntax of the URL is `jdbc:mysql://`*server*`:`*port*`/`*database_name*.

    Note – If you are using a third-party JDBC driver, refer to the documentation for that driver for information about the URL.

  - **user** – The login ID for the user you created in "Step 2: Create a Master Index Database and User" on page 46.
  - **password** – The password for the above user.
  - **DatabaseName** – The name of the database.

- **For Microsoft SQL Server:**
  - **URL** – The URL that points to the database. The syntax of the URL is `jdbc:mysql://`*server*`:`*port*`:`*database_name*.

    Note – If you are using a third-party JDBC driver, refer to the documentation for that driver for information about the URL.

  - **user** – The login ID for the user you created in "Step 2: Create a Master Index Database and User" on page 46.
  - **password** – The password for the above user.
  - **SendStringParametersAsUnicdoe** – An indicator of whether string parameters are sent to the database in Unicode or in the default character encoding of the database. Set this property to **`false`**.
  - **MaxPooledStatements** – The maximum number of prepared statements in the cache. Set this property to **`1000`**.

13 **Follow the previous steps again to create another connection pool for the sequence manager.**

14 **Continue to "Step 3: Create the JDBC Resources" on page 62.**

Step 3: Create the JDBC Resources

Note – MySQL is only supported in Java CAPS 6 Update 1. You only need to create the sequence connection pool if you are using Update 1.

A JDBC resource (also known as a data source) gives the master index application the ability to connect to the database. Two JDBC resources are required.

▼ To Create the JDBC Resources

Before You Begin Create the JDBC connection pools, as described in "Step 2: Create two JDBC Connection Pools" on page 59.

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 two JDBC Connection Pools" on page 59.**

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 previous steps to create a JDBC resource for the sequence manager with these guidelines:**

- In the Pool Name field, enter the name of the *second* JDBC connection pool you created.
- The name of the JDBC 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`**.