Oracle® Communications Billing and Revenue Management

LDAP Manager

Release 15.2
G35880-01
January 2026

Oracle Communications Billing and Revenue Management LDAP Manager, Release 15.2

G35880-01

Contents

3 Managing the Directory Server Organization

About This Content

This guide describes how to integrate Oracle Communications Billing and Revenue Management (BRM) LDAP Manager with your LDAP directory server.

Audience

This guide is intended for developers and system administrators.

1

About LDAP Manager

Learn how to integrate Oracle Communications Billing and Revenue Management (BRM) LDAP Manager with your LDAP directory server.

Topics in this document:

How LDAP Manager Works

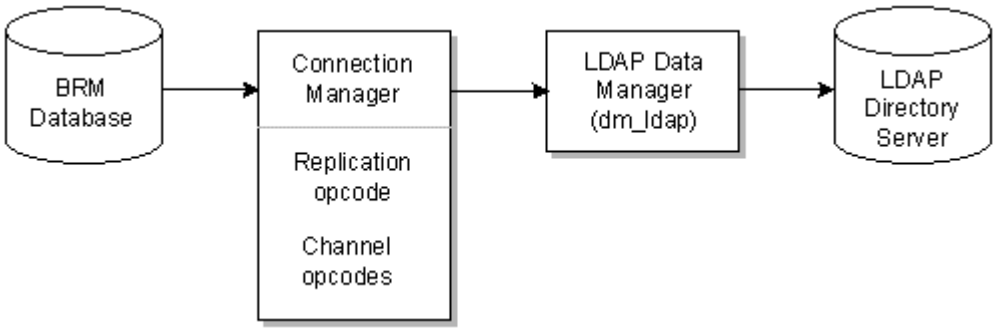
Use LDAP Manager to integrate your LDAP directory server with BRM. LDAP Manager replicates account and service data in the BRM database to the LDAP database. Changes made to the LDAP database are not replicated in the BRM database.

To send data to the LDAP directory server, LDAP Manager uses the following components:

- **Replication Opcode**: This opcode determines which BRM data is sent to the LDAP directory and how that data is structured. For more information, see "Understanding the Replication Module".
- **Channel Framework**: The channel framework sends data to the LDAP directory server either serially or in parallel, depending on your configuration. It also helps manage directory server downtime and fulfill auditing requirements. For more information, see "Understanding the Channel Framework".
- **LDAP Data Manager**: The LDAP DM translates the data into the LDAP database format and establishes a connection between BRM and the LDAP database. For more information, see "About the LDAP Data Manager".
- **pin_channel_export**: This export application transfers changes from BRM to the directory server, ensuring that the data in the BRM channel stays in sync with the external directory server. It also includes reporting and error management utilities: **pin_channel_report** and **pin_channel_clear_errors**. For more information, see "Configuring the pin_channel_export Utility".

Figure 1-1 shows how changes to data in the BRM database are sent to the LDAP directory server.

Figure 1-1 BRM Database Updates to LDAP Directory Server



About Setting Up Your LDAP Integration

To set up an LDAP integration, you perform these tasks:

- Install and configure BRM, along with your LDAP directory server.
- Install and configure LDAP Manager. For example, you need to configure the connections between the CM and the LDAP Data Manager. See "Installing LDAP Manager".
- Define the mapping between BRM data and LDAP data. For example, you need to specify how an account object in BRM is stored in the LDAP database. See "Mapping Data Between LDAP Data Manager and Your Directory Server".
- Set up your directory server. To set up your directory server with attributes that BRM can understand, such as Portal Object ID (POID), names, addresses, currency, login, and service information, you must create a BRM object-type definition called the replicate user or **r_user** object class in your directory server. See "Managing the Directory Server Organization".

2

Mapping Data Between LDAP Data Manager and Your Directory Server

Learn about Oracle Communications Billing and Revenue Management (BRM) LDAP Data Manager, its API and mapping files, its predefined mapping schemes, the BRM LDAP object model, the channel framework, and the replication module.

Topics in this document:

About the LDAP Data Manager

The LDAP Data Manager translates the BRM object model to the directory server object model. It implements a subset of the standard BRM object API.

You can think of the mapping task between BRM and your directory server in these terms:

- At the lowest level, you map BRM fields to directory server attributes.
- At the next level, you map BRM classes to directory server objects.
- At the highest level, you determine the mapping scheme to use for your directory server, for example, the user mapping scheme or the one-to-one mapping scheme.

LDAP Data Manager API and Mapping Files

The next sections describe the LDAP Data Manager API, in particular, the data types, the LDAP mapping files, class and subclass mapping restrictions, required fields, base class attributes, and LDAP Manager supported and unsupported operations.

LDAP Data Manager Data Types

BRM accesses all attribute values as UTF8 strings in the directory server. You can encode the following BRM field values as strings:

- PIN_FLDT_INT, PIN_FLDT_ENUM, and PIN_FLD_DECIMAL are encoded as the decimal representation of their values, with each decimal digit represented by its character equivalent. For example, the number **1234** is represented by the character string **1234**.
- PIN_FLDT_POID and PIN_FLDT_STR are encoded as is.
- PIN_FLDT_TSTAMP is encoded as the string representation of the UTC time value. By default, time stamps are encoded in their decimal format.

> **Note**
>
> – You can also represent time stamps as printable strings in a more readable format. For more information, see "Configuring the Connection Manager for LDAP Manager".
>
> – By default, if a field of type TSTAMP is not present in the input flist, its value is set to 0. However, this default does not apply to the following TSTAMP fields, which have special meanings: PIN_FLD_EFFECTIVE_T, PIN_FLD_CREATED_T, and PIN_FLD_MOD_T. These fields require valid timestamp values and should not be left unset or defaulted to 0.

- PIN_FLDT_ARRAY fields are mapped to multivalue attributes.

> **Note**
>
> The array elements must each have only one field and the element-id is not significant. Arbitrary BRM arrays are not supported.

- PIN_FLDT_BINSTR fields are encoded using base-64 encoding and stored as binary values.
- PIN_FLDT_SUBSTRUCT, PIN_FLDT_BUF, PIN_FLDT_OBJ, and PIN_FLDT_ERR type fields are not supported.

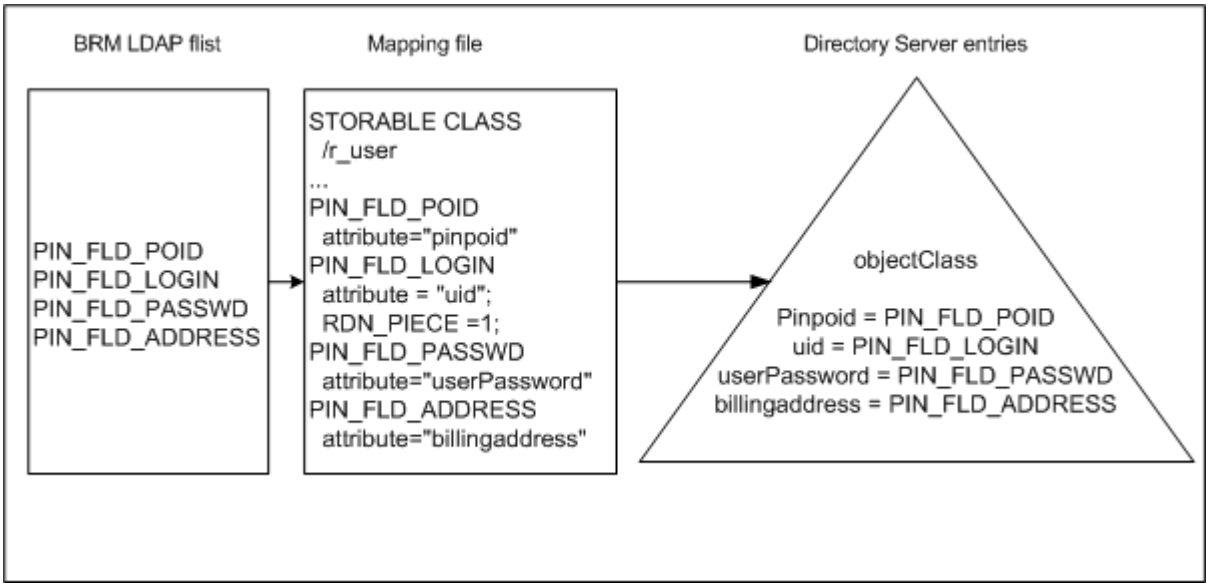
LDAP Data Manager Mapping Files

The LDAP Data Manager mapping file specifies the BRM data elements to replicate in your directory server. The mapping file maps:

- BRM classes to directory server classes.
- BRM objects to directory server entries.
- BRM fields to directory server attributes.

Figure 2-1 shows snippets of the BRM LDAP flist, the mapping file, and the directory schema and how the components correspond to each other:

Figure 2-1 BRM LDAP Flist, Mapping File, and Directory Server Entries



When you install LDAP Manager, a sample mapping file (*BRM_home*/**sys/ldap.idl**) and the LDAP Data Manager configuration file (*BRM_home*/**sys/dm_ldap/pin.conf**) are included with your installation. You use the **ldap.idl** mapping file to define your BRM LDAP mapping configuration.

The mapping file must match your directory server implementation. The mapping file specifies the following information for each object type:

- Mapping from BRM fields to LDAP attributes, permission for attributes (mandatory/optional/system), and modify permissions (readable/writable).
- Absolute distinguished name (DN) suffix, which determines the location of the entry for replication purposes.

  For more information on how BRM composes the DN when it creates objects, see "About Managing Directory Server Entries".

  In some cases, after you set up your mapping file, you might want to override the location value in the mapping file by using BRM LDAP DN qualifiers or location parameters. For more information, see "Specifying Directory Tree Entries".
- BRM field for the relative distinguished name (RDN) of the entry. The LDAP Data Manager uses the RDN when you do not specify a DN during creation of the entry.

  > **Note**
  >
  > BRM does not support RDNs composed of multiple attributes within the entry. Only one top-level, nonarray field in the base class can be tagged as an RDN component.

- ObjectClass of the entry that corresponds to the object type.
- Ordered list of LDAP attributes with explicit values that are passed along when you create the entry. This is useful in LDAP environments that use inherited object classes. For

example, in a Netscape environment with **/r_user** class implementation, the object class attribute is predefined with the following values:

- **person**
- **inetOrgPerson**
- Other values used during entry creation

In addition to the mapping file you create, you must configure the following directory server information in the LDAP Data Manager **pin.conf** file:

- Directory server port and location
- Directory server bind information
- Directory server password information
- Queue-based daemon parameters

The LDAP Data Manager authenticates itself by using a clear password (simple authentication). BRM binds to the User ID and password that you set up for your directory server. For more information on configuring BRM and LDAP components to communicate with each other, see "Installing LDAP Manager".

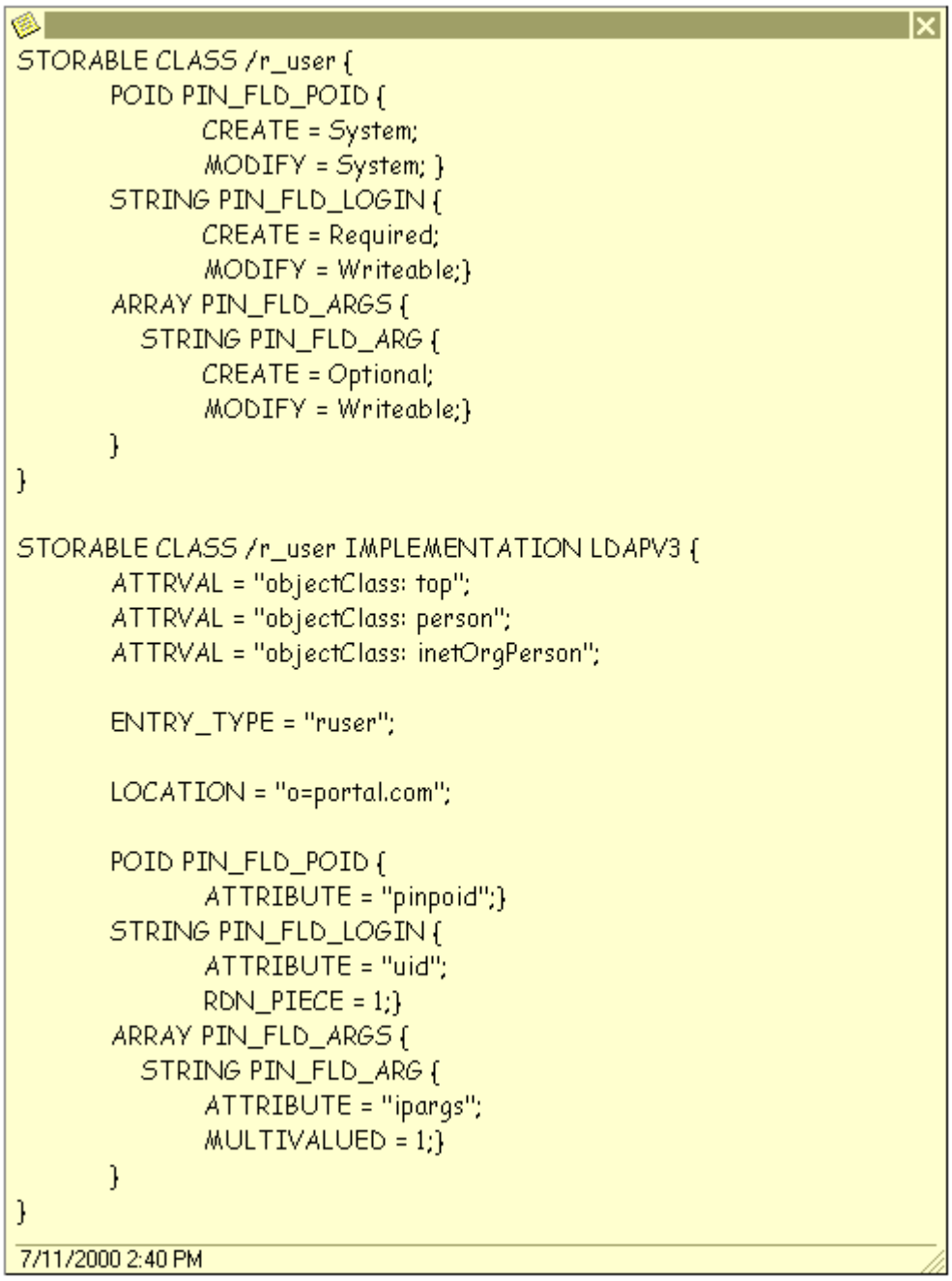
Sample Mapping File

The LDAP Data Manager uses the interface description language (idl) format to map BRM classes, subclasses, and fields to directory server entries.

The LDAP Data Manager parses the mapping file when it starts. BRM determines the location of directory server entries from the mapping file for replication purposes.

Figure 2-2 shows parts of the class definitions you can create and their corresponding implementation values:

Figure 2-2 Class Definitions and Corresponding Implementation Values



```
STORABLE CLASS /r_user {
        POID PIN_FLD_POID {
                CREATE = System;
                MODIFY = System; }
        STRING PIN_FLD_LOGIN {
                CREATE = Required;
                MODIFY = Writeable;}
        ARRAY PIN_FLD_ARGS {
          STRING PIN_FLD_ARG {
                CREATE = Optional;
                MODIFY = Writeable;}
        }
}

STORABLE CLASS /r_user IMPLEMENTATION LDAPV3 {
        ATTRVAL = "objectClass: top";
        ATTRVAL = "objectClass: person";
        ATTRVAL = "objectClass: inetOrgPerson";

        ENTRY_TYPE = "ruser";

        LOCATION = "o=portal.com";

        POID PIN_FLD_POID {
                ATTRIBUTE = "pinpoid";}
        STRING PIN_FLD_LOGIN {
                ATTRIBUTE = "uid";
                RDN_PIECE = 1;}
        ARRAY PIN_FLD_ARGS {
          STRING PIN_FLD_ARG {
                ATTRIBUTE = "ipargs";
                MULTIVALUED = 1;}
        }
}
7/11/2000 2:40 PM
```

This has two sections:

- **Class definitions** specify the required and optional fields, the permissions for fields (read/write), and the length for string fields.
- **Implementation definitions** specify directory server object classes, base location in the directory tree, attribute names for fields, and how to compose a Distinguished Name (directory's object id).

LDAP Data Manager Class and Subclass Mapping Restrictions

The following restrictions apply to the classes and subclasses you define in the mapping file:

- Only one nonarray required field in the base class can be tagged as RDN_PIECE.
- Only a field within an array can be tagged MULTIVALUED.
- Required fields in the base class:
  - PIN_FLD_POID
  - PIN_FLD_MOD_T
  - PIN_FLD_CREATED_T
- Attributes that apply only to the base class:
  - ENTRY_TYPE
  - ATTRVAL

LDAP Data Manager Supported Operations

The LDAP Data Manager uses LDAP-specific inputs and outputs to opcodes that perform operations on a base BRM system. Therefore, if you have experience customizing BRM, calling the LDAP Data Manager opcodes is similar to calling the Oracle Data Manager (**dm_oracle**) opcode.

Use the BRM and LDAP Data Manager API opcodes to perform the following data propagation tasks:

- Create entries in the directory server.
- Modify entries in the directory server.
- Delete entries in the directory server.
- Delete attributes from the directory server entry.
- Read entries from the directory server.
- Read attributes from an entry in the directory server.
- Search for entries in the directory server.
- Verify that the LDAP Data Manager and the directory server daemon/service processes are up and running and communicating with each other.

For more detailed information on these operations, including their inputs and outputs, see "LDAP Base Opcodes" in *BRM Opcode Guide*.

For more information on working with directory server entries, see "Managing the Directory Server Organization".

LDAP Data Manager Unsupported Operations

LDAP Data Manager does *not* support transaction operations (PCM_OP_TRANS_*).

LDAP Data Manager Predefined Mapping Schemes

The purpose of the mapping scheme is to replicate BRM data to your directory server. The LDAP Data Manager provides you with the following predefined mapping schemes:

- **user mapping scheme**: Maps BRM **/account** and **/service** objects to one entry in the directory server.
- **one-to-one mapping scheme**: Maps BRM **/account** and **/service** objects to separate entries in the directory server.

For a procedure overview of how to use the user mapping scheme, see "Defining the User Mapping Scheme". For a procedure overview of how to use the one-to-one mapping scheme, see "Defining the One-to-One Mapping Scheme".

Understanding the BRM Object Model

Before you map BRM objects to directory server entries, you need to understand the BRM object model. The next sections describe the BRM object model and how BRM translates its data elements to directory server entries.

Replicatable User Objects (/r_user)

Both the user mapping scheme and one-to-one mapping scheme use a replicatable user object (**/r_user**) class to propagate BRM information to the directory server. The **/r_user** object is not an actual BRM object, but rather, an object type definition that allows the LDAP Data Manager to translate directory server information into a format that BRM can understand.

BRM objects that are replicated to the directory server are called *replicatable* objects. These can be *virtual* objects defined just for replication; these objects do not necessarily exist in the BRM database.

The only requirements for these replicatable objects are that they have a unique Portal object ID (POID) and that they are replicated intact to external sites. Reasons for the *whole* replication are:

- To preserve the object interface
- To simplify mapping to and from external implementations

For more information on BRM data types and the POID, see "Understanding the BRM Data Types" in *BRM Developer's Guide*.

Replica User Objects

A *replica* object is a subset of the corresponding BRM **/account** and **/service** objects that you define. By default, the **/r_user** replica object includes a number of the fields that make up the **/account** and **/service** objects as defined in the **ldap.idl** mapping file.

The replica contains enough information to point to the corresponding replicatable object, but with object IDs that are different from those of the replicatable objects. The object types must be the same.

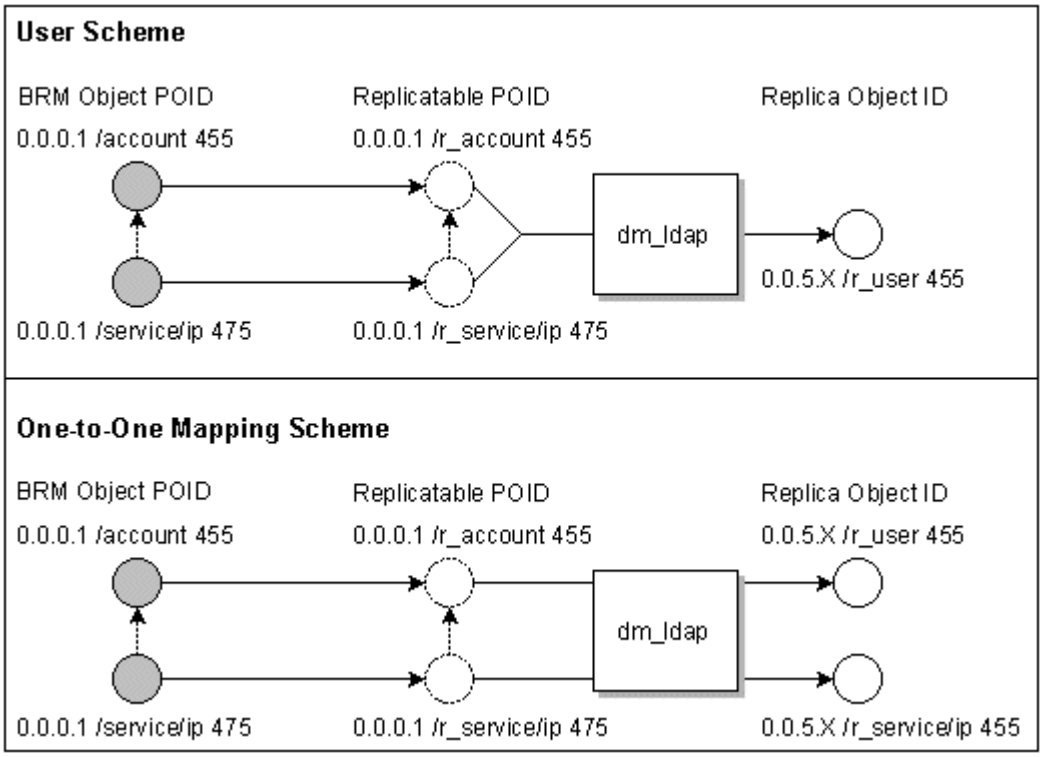
BRM maps replicas to entries in the directory server, and these entries have a **pinpoid** directory attribute that holds the replica object ID.

Because you cannot represent the entire **/account** object in the directory server, it cannot serve as a replicatable object. To locate the **/account** object easily, BRM creates a replicatable object with the same POID as the **/account** object, but with a different type.

For example, if you use the one-to-one mapping scheme for translation purposes, the POID of the replicatable object, such as **0.0.0.1 /r_account 455,** is used to create a replica with the object id **0.0.5.X /r_account 455**, where the first part of the POID is the LDAP Data Manager database number and **X** is the number of the Oracle database.

Figure 2-3 shows the BRM replication scenario for both the user scheme and the one-to-one mapping scheme.

Figure 2-3 BRM Replication Scenarios

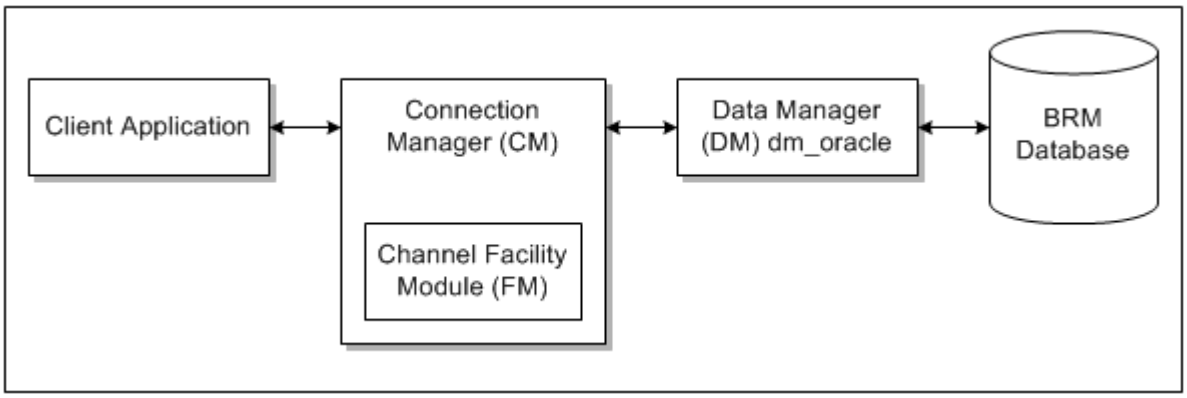


Understanding the Channel Framework

By default, BRM propagates changes to the directory server by using the channel framework. You can configure BRM to export data serially or in parallel. The default is serially.

Figure 2-4 shows the BRM channel framework data flow.

Figure 2-4 BRM Channel Framework Data Flow



As changes, such as the creation of an account or the modification of a service, take place in BRM through client (driver) applications such as Billing Care, or various custom applications, transactions occur in BRM and are captured as events. *After* you commit the changes to the BRM database, BRM pushes the changes to the specified LDAP Data Manager in near real time.

About Channels and Data Propagation

BRM uses the **/channel** object to store configuration information for BRM channel events. A channel determines how a set of events gets published to an LDAP directory server. Channels also organize what data is transferred between the BRM database and the LDAP database by defining *suppliers* and *consumers*:

- **Suppliers** (PIN_FLD_SUPPLIER_OBJ) contain information about the BRM object that triggered the creation of the **/channel_event** object. A channel definition can contain any number of suppliers.

  Suppliers include the BRM database number in which the event occurred and the BRM object name. For example:

  ```
  0.0.0.1 /account
  ```

  When an event occurs in the BRM database, the PCM_OP_CHANNEL_PUSH opcode searches for all suppliers and returns a list of the channels that contain them. For each channel retrieved, it creates one **/channel_event** object in the BRM database. This handles the case in which the same supplier is associated with multiple channels.

- **Consumers** (PIN_FLD_CONSUMER_OBJ) accept change information from suppliers, translate the change information, and call the LDAP Data Manager to propagate the changes to the directory server.

  Consumers include the LDAP Data Manager (DM) to which the Connection Manager (CM) connects and the destination object name. For example:

  ```
  0.0.5.1 /r_account -1
  ```

  Each consumer is coupled with a reference to an opcode (PIN_FLD_PUSH_OPCODE field) that publishes the object data to the directory server. For example, **745** is the opcode reference to the PCM_OP_REPL_POL_PUSH policy opcode, which is the default push opcode.

Channels act as reliable message queues and provide retry capabilities.

Channel Object Composition

The **/channel** object contains the following information:

- Channel ID
- Description
- Channel family
- Channel order
- List of suppliers
- List of consumers and a per-consumer **push** opcode

For an example channel definition, see "Example channel_config.xml File".

Channel Event Composition

BRM uses **/channel_event** objects to track changes to a specific consumer. For example, BRM creates a **channel_event** entry to indicate that an accounting type change was made.

The **/channel_event** object contains the following information:

- Channel ID
- Channel event ID
- Source ID
- Supplier ID
- Status

Each **/channel_event** represents a unit of work that needs to be propagated. It contains a pointer to the object that generates the change (SOURCE_OBJ).

About Channel Families

A channel *family* is a channel attribute that enables you to create independent groups of channels that get handled separately by the channel framework. You define a family by setting the PIN_FLD_FAMILY_ID value in the **/channel** object.

To retrieve channel events for a channel family, run the **pin_channel_export** utility with the **-f** parameter and specify the family ID. When the **pin_channel_export** utility runs, it retrieves the channels containing that ID and then determines how to process the channel events based on their channel definitions.

> **Note**
>
> A channel definition can have only one channel family value. Within that family, all channels are published according to the channel priority, if defined. For more information, see "About Channel Order".

For more information on assigning a channel family, see "Configuring Channel Definitions".

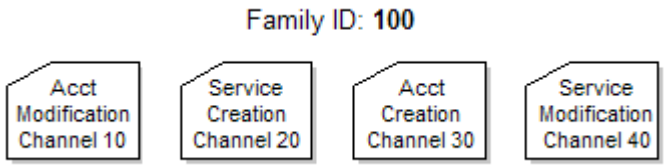
For information on how the **pin_channel_export** utility works, see "How Channel Events are Published".

About Channel Order

The *channel order* defines the sequence in which channels with the same family ID are processed. The publishing order is ascending, with **1** being the highest priority. You define the channel order by setting the PIN_FLD_CHANNEL_ORDER value in the **/channel** object.

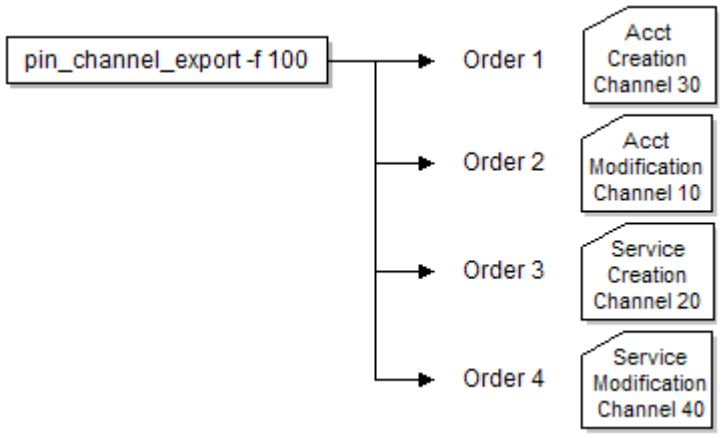
When the "pin_channel_export" utility runs, it reads the channel definitions in that family and prioritizes them based on their channel order. For example, consider the channel IDs **10**, **20**, **30**, and **40** in family **100** shown in Figure 2-5.

Figure 2-5 Example Channel IDs for Family ID 100



Because channel 30 is an account creation channel event, its order is set to **1** so it will be created in the LDAP database before any modification events associated with the account are published. Channel 10, which contains account modification suppliers, is published next. Channel 20, the service creation channel, is ordered after the account creation channel but is published before the service modification channel 40 as depicted in Figure 2-6.

Figure 2-6 Example Publishing Order for Family ID 100



When the **pin_channel_export** utility runs, it first retrieves batches of **/channel_event** objects belonging to channel 30 and publishes them, then it does the same for channels 10, 20, and 40, in that order.

> **Note**
>
> You can also set the publishing priority to **0**, which is the default. In such cases, the channels are published in increasing order of their channel IDs.

For information on assigning a channel order, see "Configuring Channel Definitions".

For more information on how channel events are published, see "How Channel Events are Published".

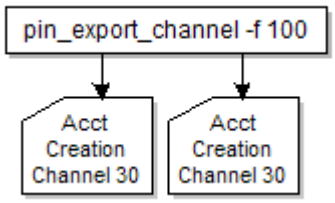
About Channel Publishing Mode

You can define whether channel events are published to an LDAP server serially (**0**, the default) or in parallel by setting the PIN_FLD_MULTI_THREADED value of the **/channel**

object. When you set the value to **1**, all channel events with that channel ID are published concurrently.

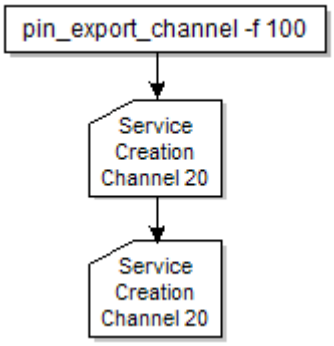
For example, say channel ID 30 has a PIN_FLD_MULTI_THREADED value of **1** and channel ID 20 has a PIN_FLD_MULTI_THREADED value of **0**. When the **pin_channel_export** utility runs, it processes the channel events for channel ID 30 in parallel because the publishing mode is set to multithreaded (**1**) as shown in Figure 2-7.

Figure 2-7 Multithreaded Publishing Example for Family ID 100



It processes channel events for channel ID 20 one after another because the publishing mode is set to serial (**0**) as shown in Figure 2-8.

Figure 2-8 Serial Publishing Example for Family ID 100



For information on assigning a publishing order, see "Configuring Channel Definitions".

For more information on how channel events are published, see "How Channel Events are Published".

About Defining Channels

By default, you define channels in the *BRM_home*/**sys/data/config/channel_config.xml** file. This configuration file contains one or more channel definitions that get stored in the **/channel** object. Each channel definition is defined in the file as a **Channel** child element of the **ChannelConfig** parent element.

Table 2-1 describes the elements and attributes in the **Channel** child element and shows their corresponding fields in the **/channel** object.

Table 2-1 Channel Elements and Attributes with Corresponding Fields in the /channel Object

XML Field	/channel Object Field	Description	Possible Values
FldChannelId	PIN_FLD_POID	A number that identifies the */channel* object in the BRM database.	Any positive integer less than **1000**. Channel IDs **100**-**103** are pre-defined but can be customized.
FldName	PIN_FLD_NAME	An optional character string that provides a descriptive name for the channel.	The name must be unique within your BRM system. Minimum length is 1 character; maximum length is 255.
FldFamilyId	PIN_FLD_FAMILY_ID	A number that identifies the family ID. Channels with the same family ID are processed as an independent group.	Any positive integer. **0** - Disables the family functionality.
FldOrder	PIN_FLD_CHANNEL_ORDER	The publishing order of the channel in a family, in ascending order. If two channels in the same family have the same channel order, the behavior is considered *undefined*.	Any positive integer. **1** - The highest priority. **0** - The default, which sets the order as *undefined*. Channels are published in increasing order of their channel IDs.
FldMultithread	PIN_FLD_MULTI_THREADED	Specifies if the channel should be published serially or in parallel.	**0** - Publish channels serially. **1** - Publish channels in parallel.
FldSupplierObj	PIN_FLD_SUPPLIERS	An array that holds information about the suppliers.	The array element ID must be **1**.
FldDatabase	PIN_FLD_SUPPLIER_OBJ	The BRM database number.	The BRM database number (FldDatabase) and the supplier event name (FldSupplierName). The format is *database_number*/*object_name*.
FldSupplierName	PIN_FLD_SUPPLIER_OBJ	The name of the supplier object.	The BRM database number (FldDatabase) and the supplier event name (FldSupplierName). The format is *database_number*/*object_name*.
FldConsumerObj	PIN_FLD_CONSUMERS	An array that holds information about the consumers.	The array element ID is not significant.
FldDatabase	PIN_FLD_CONSUMER_OBJ	The LDAP database number.	The LDAP database number (FldDatabase) and the consumer name (FldConsumerName). The format is *database_number*/*object_name*.
FldConsumerName	PIN_FLD_CONSUMER_OBJ	The name of the replicatable object in the LDAP database.	The consumer object is comprised of the LDAP database number (FldDatabase) and the consumer name (FldConsumerName). The format is *database_number*/*object _name*.

Table 2-1 (Cont.) Channel Elements and Attributes with Corresponding Fields in the /channel Object

XML Field	/channel Object Field	Description	Possible Values
FldPushOpcode	PIN_FLD_PUSH_OPCODE	The number of the opcode that is used to publish information to the LDAP database. By default, this points to the PCM_OP_REPL_POL_PUSH policy opcode.	Any positive integer.

After editing the contents of the XML file, you use the "load_channel_config" utility to load the contents of the file into the **/channel** object in the database. Before loading the contents of the file, the utility validates the contents against the file's schema definition. If the contents do not conform to the schema definition, the load operation fails.

By default, the schema definition is located in the *BRM_home***/xsd/channel_config.xsd** file.

> **Note**
>
> The **channel_config.xml** file must follow all standard XML formatting rules.

Example channel_config.xml File

The following example shows a typical **channel_config.xml** file. It defines two channel families, **100** and **200**, and eight channels: **10**, **20**, **30**, **40** in family **100**, and **50**, **60**, **70**, **80** in family **200**.

```
<?xml version="1.0" encoding="UTF-8"?>
<ChannelConfig xmlns="http://www.portal.com/schemas/BusinessConfig"  xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://www.portal.com/schemas/BusinessConfig channel_config.xsd">

  <Channel FldChannelId="30" FldFamilyId="100" FldOrder="1" FldMultithread="1" FldName="Account Creation
Channel">
     <ConsumersArray>
          <FldConsumerObj> "/r_account" </FldConsumerObj>
          <FldPushOpcode> 745 </FldPushOpcode>
     </ConsumersArray>
     <SuppliersArray>
     <FldSupplierObj> "/account" </FldSupplierObj>
     </SuppliersArray>
  </Channel>

  <Channel FldChannelId="10" FldFamilyId="100" FldOrder="2"  FldMultithread="1" FldName="Account
Modification Channel">
     <ConsumersArray>
          <FldConsumerObj> "/r_account" </FldConsumerObj>
          <FldPushOpcode> 745 </FldPushOpcode>
     </ConsumersArray>
     <SuppliersArray>
          <FldSupplierObj> "/event/customer/nameinfo" </FldSupplierObj>
          <FldSupplierObj> "/event/customer/product_status" </FldSupplierObj>
          <FldSupplierObj> "/event/customer/status" </FldSupplierObj>
          <FldSupplierObj> "/event/customer/billinfo"</FldSupplierObj>
     </SuppliersArray>
  </Channel>
```

```
  <Channel FldChannelId="20" FldFamilyId="100" FldOrder="3"  FldMultithread="0" FldName="Service
Creation Channel">
     <ConsumersArray>
          <FldConsumerObj> "/r_service" </FldConsumerObj>
          <FldPushOpcode> 745 </FldPushOpcode>
     </ConsumersArray>
     <SuppliersArray>
          <FldSupplierObj> "/service" </FldSupplierObj>
     </SuppliersArray>
  </Channel>

  <Channel FldChannelId="40" FldFamilyId="100" FldOrder="4"  FldMultithread="1" FldName="Service
Modification Channel">
     <ConsumersArray>
          <FldConsumerObj> "/r_service" </FldConsumerObj>
          <FldPushOpcode> 745 </FldPushOpcode>
     </ConsumersArray>
     <SuppliersArray>
          <FldSupplierObj> "/event/customer/login" </FldSupplierObj>
          <FldSupplierObj> "/event/customer/password" </FldSupplierObj>
          <FldSupplierObj> "/event/notification/service" </FldSupplierObj>
          <FldSupplierObj> "/event/customer/status" </FldSupplierObj>
          <FldSupplierObj> "/event/billing/product" </FldSupplierObj>
     </SuppliersArray>
  </Channel>

  <Channel FldChannelId="50" FldFamilyId="200" FldOrder="1"  FldMultithread="1" FldName="Account
Creation Channel">
     <ConsumersArray>
          <FldConsumerObj> "/r_account" </FldConsumerObj>
          <FldPushOpcode> 746 </FldPushOpcode>
     </ConsumersArray>
     <SuppliersArray>
          <FldSupplierObj> "/account" </FldSupplierObj>
     </SuppliersArray>
  </Channel>

  <Channel FldChannelId="70" FldFamilyId="200" FldOrder="2"  FldMultithread="0" FldName="Account
Modification Channel">
     <ConsumersArray>
          <FldConsumerObj> "/r_account" </FldConsumerObj>
          <FldPushOpcode> 746 </FldPushOpcode>
     </ConsumersArray>
     <SuppliersArray>
          <FldSupplierObj> "/event/customer/nameinfo" </FldSupplierObj>
          <FldSupplierObj> "/event/customer/status" </FldSupplierObj>
          <FldSupplierObj> "/event/customer/billinfo" </FldSupplierObj>
     </SuppliersArray>
  </Channel>

  <Channel FldChannelId="60" FldFamilyId="200" FldOrder="3"  FldMultithread="1" FldName="Broadband
Creation Channel">
     <ConsumersArray>
          <FldConsumerObj> "/r_service" </FldConsumerObj>
          <FldPushOpcode> 746 </FldPushOpcode>
     </ConsumersArray>
     <SuppliersArray>
          <FldSupplierObj> "/service/broadband" </FldSupplierObj>
     </SuppliersArray>
  </Channel>
```

```
  <Channel FldChannelId="80" FldFamilyId="200" FldOrder="4"  FldMultithread="1" FldName="Broadband Usage
Channel">
     <ConsumersArray>
          <FldConsumerObj> "/r_service" </FldConsumerObj>
          <FldPushOpcode> 746 </FldPushOpcode>
     </ConsumersArray>
     <SuppliersArray>
          <FldSupplierObj> "/event/broadband" </FldSupplierObj>
          <FldSupplierObj> "/event/broadband/usage" </FldSupplierObj>
     </SuppliersArray>
  </Channel>
</ChannelConfig>
```

How Channel Events are Published

The "pin_channel_export" utility publishes channel events to an LDAP database according to their channel ID (PIN_FLD_POID) and family ID (PIN_FLD_FAMILY_ID), if specified.

> **Note**
>
> You must have one instance of the **pin_channel_export** utility running for each family ID in your system.

Each **pin_channel_export** instance does the following:

1. If specified, uses the family ID value in the **-f** parameter to retrieve a batch of channels with that family ID. The channels are retrieved in increasing order of the channel IDs.
2. Reads the channel configurations that were retrieved and processes channel events by:
   - Prioritizing the channels based on the PIN_FLD_CHANNEL_ORDER value.
   - Determining whether channels should be published serially or in parallel based on the PIN_FLD_MULTI_THREADED value.
   - Reading the PIN_FLD_CONSUMER_OBJ value to determine which LDAP DM to send the results to.
3. Calls the PCM_OP_REPL_POL_PUSH policy opcode to publish the channel events to respective LDAP databases.

After the "pin_channel_export" utility has cycled through all necessary channels, it sleeps for the period specified in its **pin.conf** file and then starts the publishing cycle again.

For information on the channel configuration values, see "About Defining Channels".

Configuring How Channels are Published

You can define any number of LDAP directory servers to which you publish data. Before publishing data, make certain you:

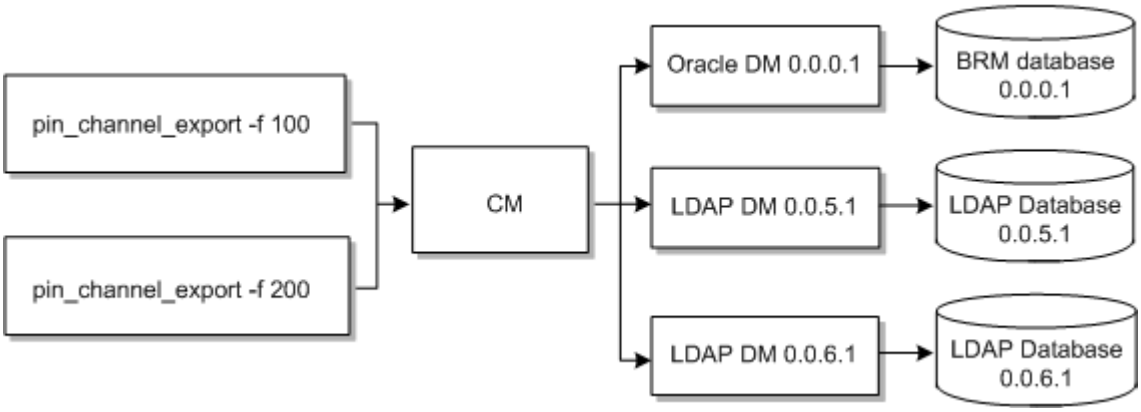
- Configure the CM **pin.conf** file, including the following entries:
  - Set the **-ldap_db** entry for each LDAP database in your system.
  - Set the **-dm_pointer** entry for each LDAP DM in your system.

  See "Configuring the Connection Manager for LDAP Manager".

- Configure each **/channel** object to contain the LDAP database number in the PIN_FLD_CONSUMER_OBJ field. The PCM_OP_REPL_POL_PUSH policy opcode retrieves this value when publishing data. See "Configuring Channel Definitions".
- Set up one instance of the **pin_channel_export** utility for each family ID in your system and make sure they both point to the same CM.

For example, Figure 2-9 shows two instances of the **pin_channel_export** utility (one for family 100 and one for family 200) which publish channels to the BRM database and two LDAP databases. Each database is connected to its own DM.

Figure 2-9 Double pin_channel_export Instance Publishing



The channel IDs and channel family IDs must be unique across each LDAP database. This is a requirement for multidatabase publishing. For example, say you have the configuration described in Table 2-2 defined in your system.

Table 2-2 Example Channel ID and Family Configuration

Database	Family ID	Channel ID	Validity	Description
0.0.5.1	100	10,20,30,40	Valid	These are the first unique family ID and channel IDs defined in the system.

Table 2-3 lists the scenarios that are either invalid or valid, depending on the database number, the family ID, and the channel IDs defined.

Table 2-3 Validity

Database	Family ID	Channel ID	Validity	Description
0.0.6.1	200	50,60,70,80	Valid	The family ID and channel IDs are unique across the databases.
0.0.6.1	100	10,20,30,40	Invalid	The family ID and channel IDs are already defined in database **0.0.5.1**.
0.0.6.1	100	50,60,70,80	Invalid	The family ID is already defined in database **0.0.5.1**.
0.0.6.1	200	10,20,30,40	Invalid	The channel IDs are already defined in database **0.0.5.1**.

Example of Publishing a Channel Family

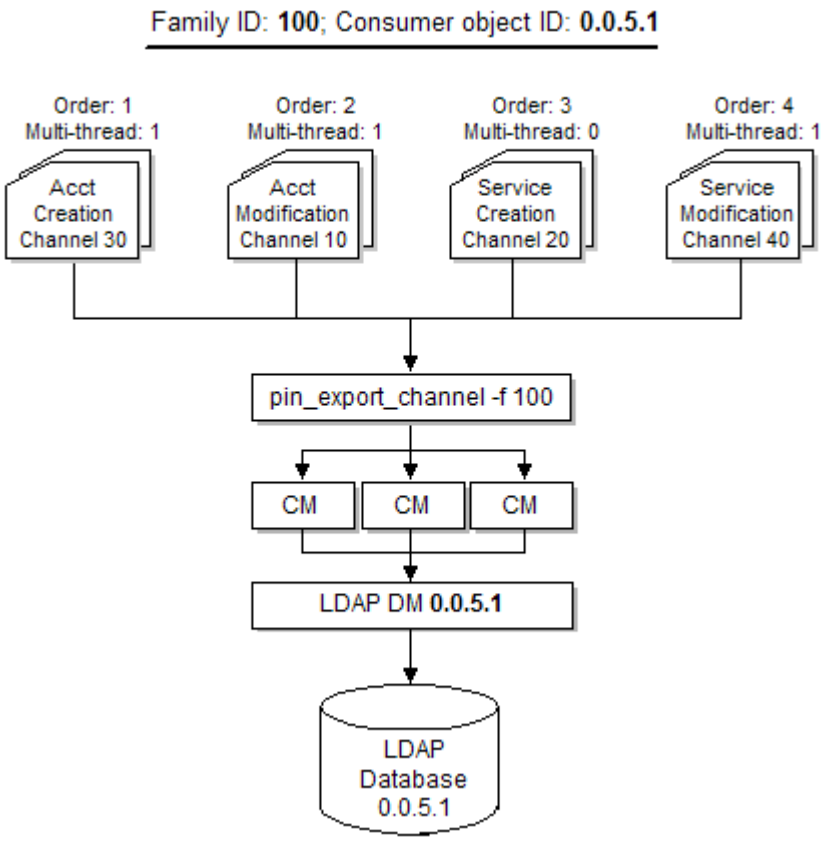
This example shows how the family ID is used to publish channels to an LDAP database. Channel IDs 10, 20, 30, and 40 have a family ID value of **100**. In addition, they all have the same PIN_FLD_CONSUMER_OBJ value of **0.0.5.1**, which means they are all published to the same LDAP database.

When you run the **pin_channel_export** utility with the **-f** parameter value set to **100**, the channels for that family ID are first prioritized by order, and, for each channel, the **pin_channel_export** utility does the following:

- Searches for **/channel_event** objects in the BRM database.
- Calls the PCM_OP_REPL_POL_PUSH policy opcode (defined in the PIN_FLD_PUSH_OPCODE field) to read the LDAP database number from the PIN_FLD_CONSUMER_OBJ field.
- Reads the CM **pin.conf** file to determine the LDAP DM for the LDAP database retrieved.
- Reads the channel's PIN_FLD_MULTI_THREADED value to determine whether to publish the channel events serially or in parallel.

The result is that all channel events for channel IDs 10, 20, 30, and 40 are pushed to LDAP DM **0.0.5.1** using three threads as shown in Figure 2-10.

Figure 2-10 Example of Publishing a Channel Family



> **Note**
>
> To connect to the BRM database, the **load_channel_config** utility needs a configuration file in the directory from which you run the utility. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.

About Setting Replicatable Objects as Consumers

When you set up replicatable objects as consumers, BRM uses the supplier/consumer/channel paradigm. BRM predefines several **/channel** configuration objects in the *BRM_home***/sys/dd/data/init_objects.source** file.

Each channel object contains two array fields:

- Consumer array (PIN_FLD_CONSUMERS)
- Supplier array (PIN_FLD_SUPPLIERS)

You can register a list of suppliers for a particular **/channel** object by adding new array elements to the PIN_FLD_SUPPLIERS array. The consumers for each of the **/channel** objects are the replicated objects that the replication module and the LDAP Manager Data Manager create.

The next sections describe the BRM LDAP channel configuration objects and the logical data that these objects track.

Tracking New Account Creation

The **/channel 100** channel object tracks new account creation (see Table 2-4).

Table 2-4 /channel 100 Channel Object

Consumer Array	Supplier Array
$DB_NO /r_account -1 PCM_OP_REPL_POL_PUSH	$DB_NO /account -1

Tracking Modifications to Accounts

The **/channel 101** channel object tracks changes to **/account** objects (see Table 2-5). In particular, it tracks modifications to customer billing information, name information, and whether the charge offer status is active or closed or whether a credit card is valid or invalid.

Table 2-5 /channel 101 Channel Object

Consumer Array	Supplier Array
$DB_NO /r_account -1 PCM_OP_REPL_POL_PUSH	$DB_NO /event/customer/billinfo -1 $DB_NO /event/customer/nameinfo -1 $DB_NO /event/customer/product_status -1 $DB_NO /event/customer/status -1

Tracking Service Creation

The **/channel 102** object tracks changes to **/service** objects (see Table 2-6).

Table 2-6 /channel 102 Channel Object

Consumer Array	Supplier Array
$DB_NO /r_service -1 PCM_OP_REPL_POL_PUSH	$DB_NO /service -1

Tracking Modifications to Services

The **/channel 103** object propagates changes to **/service** objects (see Table 2-7).

Table 2-7 /channel 103 Channel Object

Consumer Array	Supplier Array
$DB_NO /r_service -1 PCM_OP_REPL_POL_PUSH	$DB_NO /event/customer/login -1 $DB_NO /event/customer/password -1 $DB_NO /event/notification/service -1 $DB_NO /event/customer/status -1

Understanding the Replication Policy Push Operation

The channel framework uses the PCM_OP_REPL_POL_PUSH opcode to implement the push operation.

This opcode implements the translation logic for **/account** and **/service** objects.

This policy opcode performs one of the following mapping functions. The mapping function for PCM_OP_REPL_POL_PUSH is determined by the Connection Manager (CM) **pin.conf user_scheme** entry.

- If **user_scheme** is **0**, the opcode maps the **/account** and **/service** objects one-to-one to **/r_account** and **/r_service** objects respectively. There will be one entry in the directory for every object in BRM.
- If **user_scheme** is **1**, the opcode merges the **/account** and **/service** object fields to form the **/r_user** object.

By default, the Connection Manager **user_scheme pin.conf** entry is set to **1**, and the LDAP Data Manager implements a single-entry mapping operation.

If the PIN_FLD_POID field in the input flist indicates that the consumer is **/r_account -1**, this opcode does the following operations:

- If the supplier is **/account -1**, it searches for all accounts created within the time range passed to this opcode by the input fields PIN_FLD_INVOKE_T and PIN_FLD_LAST_INVOKE_T. For each account, it reads a subset of the account fields and forms a replicated object **/r_user**. It then invokes the PCM_OP_CREATE_OBJ operation on **dm_ldap** to create the directory entry, which serves as the replica of this **/r_user** object.
- For all other suppliers, it searches for all events created within the time range. For the account associated with each event, it reads the same subset of the account fields from

the previous step, and invokes the PCM_OP_WRITE_FLDS operation on **dm_ldap** to modify the directory entry.

If the PIN_FLD_POID field in the input flist indicates that the consumer is **/r_service -1**, this opcode does the following operations:

- If the supplier is **/service -1**, it searches for all services created within the time range passed to this opcode by the input fields PIN_FLD_INVOKE_T and PIN_FLD_LAST_INVOKE_T. For each service, it reads a subset of the service fields and invokes the PCM_OP_WRITE_FLDS operation on **dm_ldap**, and modifies the directory entry that corresponds to the **/r_user** object. The modified directory entry relates to the **/account** object to which the service belongs.
- For all other suppliers, this opcode searches for services modified within the time range. For each service, it reads the same subset of the service fields as in the previous step, and invokes the PCM_OP_WRITE_FLDS operation on **dm_ldap** to modify the directory entry.

The default implementation for push operations is as follows:

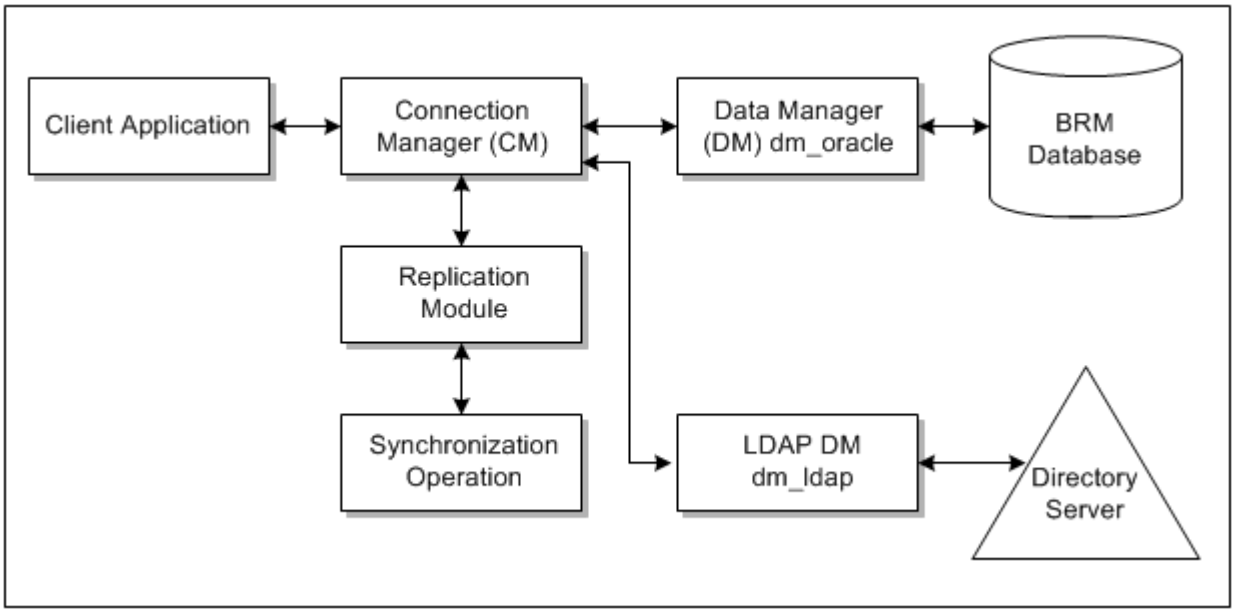
1. BRM searches for all events listed in the supplier list and retrieves a list of the changed **/account** objects (sources).
2. For each **/account** object, BRM calls PCM_OP_REPL_POL_PUSH operations with the corresponding replicatable object PID ID.
3. BRM sends the result of the push operation to the LDAP Data Manager based on the LDAP database specified in the PIN_FLD_CONSUMER_OBJ value and the family ID (PIN_FLD_FAMILY_ID), if specified. See "How Channel Events are Published".

Understanding the Replication Module

The module that controls what BRM data gets pushed and how it is structured in the directory server is the replication module. BRM calls the replication module from the channel framework synchronization operation for each channel event object in the channel event table.

Figure 2-11 shows the replication module data flow.

Figure 2-11 Replication Module Data Flow

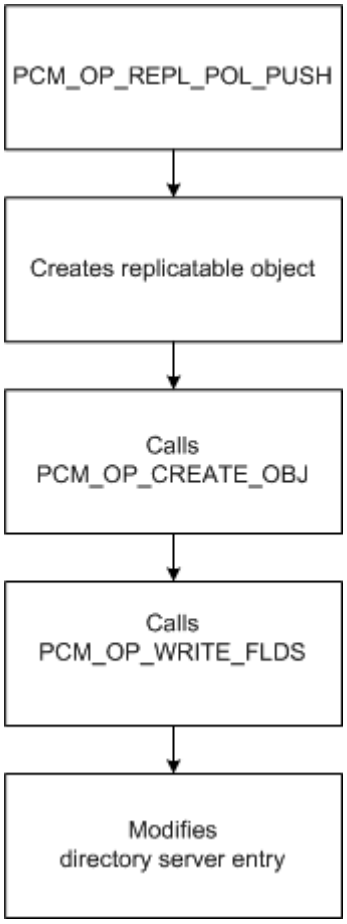


The replication module implements the translation logic for **/account** and **/service** objects, handling account and service creation and modification. By default, the replication module merges fields from **/account** and **/service** objects during the translation. For more information, see "Defining the User Mapping Scheme".

Replication Policy Default Implementation

The replication policy, PCM_OP_REPL_POL_PUSH, pushes data about the object whose POID is supplied in the input flist. It locates the account supplied in the input flist and performs the operations shown in Figure 2-12.

Figure 2-12 Replication Policy Default Implementation



The default implementation includes these high-level steps:

1. Reads a subset of the account fields, such as **bill_type** and **currency**.

   LDAP Data Manager derives this information from the field list in the mapping file.

2. Forms the replicatable **/r_user** object (assuming user scheme).
3. Calls the create object opcode, PCM_OP_CREATE_OBJ on **dm_ldap** to create the directory entry, which serves as the replica of this **/r_user** object.
4. Iteratively reads the same subset of account fields for all other suppliers (assuming events) and calls the PCM_OP_WRITE_FLDS operation on **dm_ldap** to modify the directory entry.

The opcode searches for the object whose POID was supplied in the input flist. For that service, it reads a subset of the service fields and calls the PCM_OP_WRITE_FLDS operation on **dm_ldap** to modify the directory entry that corresponds to the **/r_user** object.

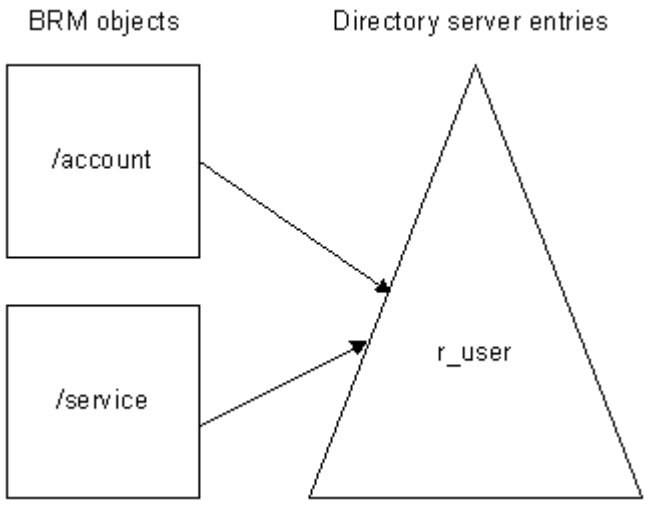
Defining the User Mapping Scheme

The *user mapping scheme* maps BRM's **/account** and **/service** objects to a single entry in the directory server, making it suitable for a single-sign-on environment.

The parameter that controls whether BRM uses this scheme is the **user_scheme** entry in the CM **pin.conf** file. If **user_scheme** is set to **1**, LDAP Data Manager uses the user mapping scheme.

Figure 2-13 illustrates how BRM maps the **/account** and **/service** objects to the **ruser** entry in the directory server.

Figure 2-13 /account and /service Object Mapping to ruser Entry



The replication policy uses fields from the **/account** and **/service** objects to create an **/r_user** object. LDAP Data Manager takes the data from the **/r_user** object and pushes it to the directory server. The replication policy determines whether to use this merged **/r_user** object by reading the **user_scheme** entry in the CM **pin.conf** file.

For more information on the CM **pin.conf** file, see "Configuring the Connection Manager for LDAP Manager".

The **/r_user** object uses fields from the BRM **/account** and **/service** objects. You map attributes in the mapping file to BRM fields. Some of these attributes are predefined for LDAP directory servers. However, the BRM-specific attributes cannot be created automatically. For more information, see "Determining the /r_user Object Class Attributes".

In the user scheme, the field name spaces can collide. For example, the status field in the **/account** and **/service** objects has different field and attribute names in the **/r_user** object (such as PIN_FLD_ACCOUNT_STATUS and PIN_FLD_SERVICE_IP_STATUS).

> **Note**
>
> With the user scheme, it is difficult for an account to have multiple services of a given type. BRM usually allows this.

The following example shows the default mapping file for the LDAP Data Manager (**ldap.idl**) for the user scheme:

```
# Class Definitions:
# All attributes need to be defined in the directory server
# All the entry_type objectclasses and attrval objectclasses need to be
# defined in the directory server.
#
STORABLE CLASS /r_user {
POID PIN_FLD_POID {
CREATE = System;
MODIFY = System;
}
TIMESTAMP PIN_FLD_CREATED_T {
CREATE = System;
MODIFY = System;
}
TIMESTAMP PIN_FLD_MOD_T {
CREATE = System;
MODIFY = System;
}
STRING PIN_FLD_NAME {
CREATE = Optional;
MODIFY = Writeable;
}
POID PIN_FLD_ACCOUNT_OBJ {
CREATE = Required;
MODIFY = Writeable;
}
STRING PIN_FLD_ACCOUNT_NO {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_FIRST_NAME {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_LAST_NAME {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_ACCOUNT_STATUS {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_BILL_TYPE_NAME {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_ADDRESS {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_CURRENCY_NAME {
```

```
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_LOGIN {
CREATE = Required;
MODIFY = Writeable;
}
STRING PIN_FLD_PASSWD {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_EMAIL_LOGIN {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_EMAIL_STATUS {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_IP_LOGIN {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_IP_STATUS {
CREATE = Optional;
MODIFY = Writeable;
}
INT PIN_FLD_MAX_MBOX_SIZE {
CREATE = Optional;
MODIFY = Writeable;
}
INT PIN_FLD_MAX_MSG_CNT {
                CREATE = Optional;
                MODIFY = Writeable;
}
INT PIN_FLD_MAX_MSG_SIZE {
                CREATE = Optional;
                MODIFY = Writeable;
}
        ARRAY PIN_FLD_ARGS {
                STRING PIN_FLD_ARG {
                        CREATE = Optional;
                        MODIFY = Writeable;
                }
        }
}

STORABLE CLASS /r_user IMPLEMENTATION LDAPV3 {
ATTRVAL = "objectClass : top";
ATTRVAL = "objectClass : person";
ATTRVAL = "objectClass : inetOrgPerson";

ENTRY_TYPE = "ruser";

LOCATION = "o=example.com";

POID PIN_FLD_POID {
ATTRIBUTE = "pinpoid";
}
TIMESTAMP PIN_FLD_CREATED_T {
ATTRIBUTE = "pincreatedt";
}
```

```
TIMESTAMP PIN_FLD_MOD_T {
ATTRIBUTE = "pinmodt";
}
STRING PIN_FLD_NAME {
ATTRIBUTE = "cn";
}
POID PIN_FLD_ACCOUNT_OBJ {
ATTRIBUTE = "billingid";
}
STRING PIN_FLD_ACCOUNT_NO {
ATTRIBUTE = "accountno";
}
STRING PIN_FLD_FIRST_NAME {
ATTRIBUTE = "givenname";
}
STRING PIN_FLD_LAST_NAME {
ATTRIBUTE = "sn";
}
STRING PIN_FLD_ACCOUNT_STATUS {
ATTRIBUTE = "billingstatus";
}
STRING PIN_FLD_BILL_TYPE_NAME {
ATTRIBUTE = "billingtype";
}
STRING PIN_FLD_ADDRESS {
ATTRIBUTE = "billingaddress";
}
STRING PIN_FLD_CURRENCY_NAME {
ATTRIBUTE = "currency";
}
STRING PIN_FLD_LOGIN {
ATTRIBUTE = "uid";
RDN_PIECE = 1;
}
STRING PIN_FLD_PASSWD {
ATTRIBUTE = "userpassword";
}
STRING PIN_FLD_EMAIL_LOGIN {
ATTRIBUTE = "mail";
}
STRING PIN_FLD_EMAIL_STATUS {
ATTRIBUTE = "mailstatus";
}
STRING PIN_FLD_IP_LOGIN {
ATTRIBUTE = "iplogin";
}
STRING PIN_FLD_IP_STATUS {
ATTRIBUTE = "ipstatus";
}
INT PIN_FLD_MAX_MBOX_SIZE {
ATTRIBUTE = "mailquota";
}
INT PIN_FLD_MAX_MSG_CNT {
ATTRIBUTE = "mailmaxmsgcount";
}
INT PIN_FLD_MAX_MSG_SIZE {
ATTRIBUTE = "mailmaxmsgsize";
}
        ARRAY PIN_FLD_ARGS {
                STRING PIN_FLD_ARG {
                        ATTRIBUTE = "ipargs";
                        MULTIVALUED = 1;
```

```
                    }
                }
            }
```

Related /account and /service Opcodes

Table 2-8 summarizes the opcodes that capture changes made to **/account** and **/service** objects so that fields in these objects can then be pushed to the directory server.

Table 2-8 Related /account and /service Opcodes

Opcodes	Action
PCM_OP_CUST_COMMIT_CUSTOMER PCM_OP_CUST_PREP_CUSTOMER	Accepts the DNs for the **/account** object in the form of an optional **/profile/ldap** object.
PCM_OP_ACT_POL_EVENT_NOTIFY	Calls CM_OP_CHANNEL_PUSH to create an entry in the **channel_event** table to track changes.
PCM_OP_CUST_POL_ENCRYPT_PASSWD	Disables or enables encryption of the **/service/ldap** password.
PCM_OP_CUST_CREATE_ACCT	Calls PCM_OP_CHANNEL_PUSH to create an entry in the **channel_event** table to track creation of accounts.
PCM_OP_CUST_CREATE_SERVICE	Calls PCM_OP_CHANNEL_PUSH to create an entry in the **channel_event** table to track creation of services.
PCM_OP_TRANS_POL_COMMIT	Defers PCM_OP_CHANNEL_SYNC automatically.

Determining the /r_user Object Class Attributes

BRM cannot automatically create directory server entries. Therefore, you must manually define the BRM data elements that you are interested in capturing with your own directory server tools.

For example, BRM-specific information such as the POID must map to a **pinpoid** attribute, which is not predefined the directory server. For this reason, you must create a replicatable user object for BRM to use in the directory server. BRM can then modify the schema of a particular entry based on how it is configured.

The next table describes the BRM **/r_user** object and how it is composed of BRM fields in the **/service** and **/account** objects as well as its corresponding directory attributes, and whether they are predefined in the LDAP directory server.

Refer to Table 2-9 when you create the **ruser** directory server entry and its corresponding BRM attributes for replication purposes. This table also lists the fields that BRM pushes to the directory server by default.

> **Note**
>
> - Case is significant for the directory server attributes. For example, you must enter **givenName**, not **givenname**. If you do not enter these attributes exactly as shown, you will encounter object class violations when you map BRM fields to directory server attributes.
> - Make sure that the object class name in the directory server matches the entry type name in the **ldap.idl** mapping file.

Table 2-9 Default Directory Server Fields

/r_user Field	Object Components and Comments	Directory Attribute	Predefined in LDAP
PIN_FLD_POID	**/account. ldap_db db_no + POID ID**	pinpoid	N
PIN_FLD_CREATED_T	Created time set by **dm_ldap**	pincreatedt	N
PIN_FLD_MOD_T	Modified time set by **dm_ldap**	pinmodt	N
PIN_FLD_NAME	**/account**. Composed by concatenating the first_name, middle_name, and last_name fields. PIN_FLD_BILLINFO [PIN_NAMEINFO_BILLING]	cn - LDAP	Y
PIN_FLD_ACCOUNT_OBJ	**/account.PIN_FLD_POID**	billingid	N
PIN_FLD_ACCOUNT_NO	**/account.PIN_FLD_ACCOUNT_NO**	accountno	N
PIN_FLD_FIRST_NAME	See PIN_FLD_NAME	givenName	Y
PIN_FLD_LAST_NAME	See PIN_FLD_NAME	sn	Y
PIN_FLD_ACCOUNT_STATUS	**/account.PIN_FLD_STATUS** Values: **Active**, **Inactive**, and **Closed**	billingstatus	N
PIN_FLD_BILL_TYPE_NAME	**/account.PIN_FLD_BILL_TYPE** Values: **Prepaid, Invoice, Debit, Credit Card, Direct Debit, Smart Card, Subordinate** (nonpaying)**, Internal, Guest, Cash, Check, Wire Transfer, Inter-Bank Payment Order, Postal Order, Unknown**	billingtype	N
PIN_FLD_ADDRESS	**/ account.PIN_FLD_NAMEINFO[PIN_NAMEINFO_BILLING]:** Street, city, state, ZIP code, and country concatenated using comma separators	billingaddress	N
PIN_FLD_CURRENCY_NAME	**/account.PIN_FLD_CURRENCY**. (String BEID name such as **US Dollar** from **/config/beid_balances**)	currency	Y
PIN_FLD_LOGIN	Login associated with **/service/ldap** for LDAP Manager	uid [RDN component] for LDAP Manager	Y
PIN_FLD_PASSWD	Cleartext password associated with **/service/ldap** for LDAP Manager	userPassword	Y
PIN_FLD_EMAIL_LOGIN	**/service/email. Login**	mail	Y
PIN_FLD_EMAIL_STATUS	**/service/email. PIN_FLD_STATUS** Values: **Active**, **Inactive**, and **Closed**	mailstatus	N
PIN_FLD_IP_LOGIN	**/service/ip**. Login	iplogin	Y
PIN_FLD_IP_STATUS	**/service/ip.PIN_FLD_STATUS** Values: **Active**, **Inactive**, and **Closed**	ipstatus	N
PIN_FLD_MAX_MBOX_SIZE	**/service/ email.PIN_FLD_SERVICE_EMAIL.PIN_FLD_MAX_MBOX_SIZE**	mailboxQuota	Y
PIN_FLD_MAX_MSG_CNT	**/service/ email.PIN_FLD_SERVICE_EMAIL PIN_FLD_MAX_MSG_CNT**	mailmaxmsgcnt	N

Table 2-9 (Cont.) Default Directory Server Fields

/r_user Field	Object Components and Comments	Directory Attribute	Predefined in LDAP
PIN_FLD_MAX_MSG_SIZE	**/service/emailad**	mailmaxmsgsize	N
PIN_FLD_PATH	**/service/email.**PIN_FLD_SERVICE_EMAIL.PIN_FLD_PATH	mailmessagestore	Y
PIN_FLD_ARGS[].PIN_FLD_ARG	**/service/ip.PIN_FLD_ARGS.** The PIN_FLD_NAME and PIN_FLD_VALUE are concatenated (using "=" separators)	ipargs	N

Creating the /r_user Object Class in the Directory Server

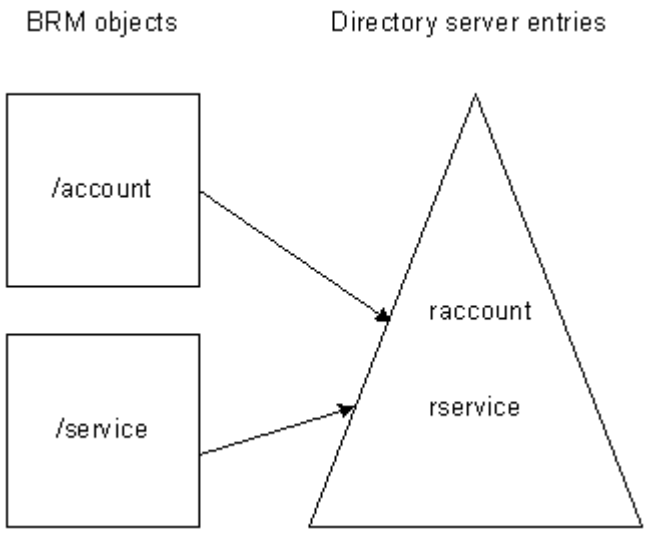
To replicate data from BRM to your directory server, you must set up the **ruser** object class in your directory server using your own LDAP directory server tools:

1. Start your Netscape directory server tools.
2. Define the **ruser** object class in the LDAP directory server or Membership directory.
3. Use the **/r_user** mapping table described in the previous section to define all the required BRM-specific attributes that you want to replicate to the list of allowable attributes in the **ruser** objectClass.
4. Save the object class in the directory server.
5. Change the value of the **LOCATION** key in the implementation section of the **/r_user** object definition in the **ldap.idl** mapping file.

Defining the One-to-One Mapping Scheme

The *one-to-one mapping scheme* maps one BRM object to one LDAP directory entry. Figure 2-14 shows how BRM maps the **/account** and **/service** objects to the **raccount** and **rservice** entries in the directory server.

Figure 2-14 BRM /raccount and rservice Object Mapping in Directory Server



To use the one-to-one mapping scheme, you must modify the replication policy C source code file. After you define the LDAP schema, you modify the replication policy (PCM_OP_REPL_POL_PUSH) to perform multiple pushes; one for the account and one for each service.

> **Note**
>
> The default implementation for the one-to-one mapping scheme in the replication module exports only **/service/email** and **/service/ip** objects.

To use the one-to-one mapping scheme:

1. Set the **user_scheme** entry of the Connection Manager (CM) **pin.conf** file to **0**. See "Configuring the Connection Manager for LDAP Manager".
2. Map the following classes in the LDAP Data Manager mapping file (**ldap.idl**). You can find the mapping file in *BRM_home***/sys/dm_ldap**.
   - **/r_account**
   - **/r_service**
   - **/r_service/ip**
   - **/r_service/email**

   See "One-to-One Mapping File Example" for the details.

   > **Tip**
   >
   > You can define the mappings in a new file and point to it from the LDAP Data Manager **pin.conf** file.
3. Define **raccount** and **rservice** object classes in the directory server.
4. For each of the **/r_account** and **/r_service**, **/r_service/ip** and **/r_service/email** fields you want to export, define a corresponding attribute and add to the list of allowed attributes of the **raccount** or **rservice** object classes.
5. Set up the **rservice** object class to allow all attributes mapped to **/r_service** and its subtypes (**/r_service/email** and **/r_service/ip**).

   > **Note**
   >
   > The LDAP Data Manager mapping scheme requires you to map to one **ENTRY_TYPE** per class and its subtypes.
6. Modify the replication policy C source code file, **fm_repl_pol_translate.c.**

   This file implements the replication policy PCM_OP_REPL_POL_PUSH. See "Changing the Replication Policy for the One-to-One Mapping Scheme" for details.

One-to-One Mapping File Example

This example shows a subset of the mapping file (the implementation section) as an example for the one-to-one mapping scheme.

```
##
## /r_account (to represent the /account storable class)
##
STORABLE CLASS /r_account {
# list all the /r_account fields (similar to the /r_user fields in the user scheme
# without the service related fields like PIN_FLD_EMAIL_LOGIN, etc.

#PIN_FLD_POID
#PIN_FLD_CREATED_T
#PIN_FLD_MOD_T
#PIN_FLD_NAME
#PIN_FLD_ACCOUNT_OBJ
#PIN_FLD_ACCOUNT_NO
#PIN_FLD_FIRST_NAME
#PIN_FLD_LAST_NAME
#PIN_FLD_ACCOUNT_STATUS
#PIN_FLD_BILL_TYPE_NAME
#PIN_FLD_ADDRESS
#PIN_FLD_CURRENCY_NAME
#PIN_FLD_GUID
#PIN_FLD_LOGIN
#PIN_FLD_PASSWD
}

#
STORABLE CLASS /r_account IMPLEMENTATION LDAPV3 {

# ENTRY_TYPE = "raccount";
# LOCATION = "o=xyz, c=US";

# provide mapping for /r_account fields
}

##
## /r_service (to represent the /service storable class)
##
STORABLE CLASS /r_service {
# list all the /r_service fields
# PIN_FLD_POID
# PIN_FLD_CREATED_T
# PIN_FLD_MOD_T
# PIN_FLD_ACCOUNT_OBJ
# PIN_FLD_LOGIN
# PIN_FLD_PASSWD
# PIN_FLD_STATUS
}

## /r_service/ip (to represent the /service/ip storable class)
STORABLE CLASS /r_service/ip {
# list all the /r_service/ip related fields
# PIN_FLD_ARGS
#PIN_FLD_ARG
}
## /r_service/email (to represent the /service/email storable class)
STORABLE CLASS /r_service/email {
# list all the /r_service/email related fields
# PIN_FLD_EMAIL_INFO
#PIN_FLD_MBOX_SIZE
#PIN_FLD_MAX_MSG_CNT
#PIN_FLD_MAX_MSG_SIZE
}
```

```
#
STORABLE CLASS /r_service IMPLEMENTATION LDAPV3 {

# ENTRY_TYPE = "rservice";
# LOCATION = "ou=services, o=xyz, c=US";

# provide mapping for /r_service fields
# MAKE PIN_FLD_POID as the RDN_PIECE.
}

#
STORABLE CLASS /r_service/ip IMPLEMENTATION LDAPV3 {
# provide mapping for /r_service/ip fields
}

#
STORABLE CLASS /r_service/email IMPLEMENTATION LDAPV3 {
# provide mapping for /r_service/email fields
}
```

Changing the Replication Policy for the One-to-One Mapping Scheme

Use this section to guide you in modifying the PCM_OP_REPL_POL_PUSH replication policy. You make all changes to the C source code file (*BRM_home*/**source/sys/fm_repl_pol/fm_repl_pol_translate.c**).

To change the replication policy:

1. Save a copy of **fm_repl_pol_translate.c**.
2. Package the input flist to LDAP Data Manager to match the mapping information you specified in the mapping file (**ldap.idl**) file associated with **dm_ldap** as described in "LDAP Data Manager Mapping Files".
3. In the **fm_repl_pol_prep_service_flds()** function, drop the PIN_FLD_STATUS and PIN_FLD_LOGIN associated with **/service/ldap** only if you are using the **user_scheme**.
4. In the **fm_repl_pol_prep_service_email_flds()** function, move the information in the PIN_FLD_SERVICE_EMAIL substruct to the PIN_FLD_EMAIL_INFO array field as shown here:

```
e_flistp = PIN_FLIST_SUBSTR_TAKE(rep_flistp,
                PIN_FLD_SERVICE_EMAIL, 0, ebufp);
if (user_scheme) {
   PIN_FLIST_CONCAT(rep_flistp, e_flistp, ebufp);
   PIN_FLIST_DESTROY(e_flistp, NULL);
} else {
   PIN_FLIST_ELEM_PUT(rep_flistp, e_flistp,
                PIN_FLD_EMAIL_INFO, 0, ebufp);
}
```

3

Managing the Directory Server Organization

Learn how to use the Oracle Communications Billing and Revenue Management (BRM) LDAP API to manage and manipulate the directory tree organization.

Topics in this document:

About Managing Directory Server Entries

You can create, delete, read, and write directory server entries and attributes in any part of a directory tree.

BRM uses Distinguished Names (DNs) for these operations in one of two ways:

- From the location value in the mapping file
- From the location in the input flist based on:
  - The location value and Relative Distinguished Name (RDN) piece
  - The DN field and DN qualifiers (complete and prefixed) used in the create operation. The create operation can also use the parent DN qualifier.

For more information, see "Specifying Directory Tree Entries".

The BRM LDAP Data Manager makes the following assumptions during create, delete, read, and write operations with the DN you supply at run time:

- The directory server entry is an instance of the object defined in the mapping file.
- The location value matches the location you specified in the mapping file.

Semantics for the LDAP Modify Operation

You use the LDAP modify operation to manage directory server entries. This operation is used by the following opcodes:

- PCM_OP_DELETE_FLDS

- PCM_OP_DELETE_OBJ
- PCM_OP_WRITE_FLDS
- PCM_OP_CREATE_OBJ

The LDAP modify operation accepts a list of modifications to be performed, and performs the modifications in the order listed as a single atomic operation. The value that may be taken on by the operation field in each modification construct can have the following semantics:

- **Add**: Adds specified values to the given attribute and creates the attribute if necessary.
- **Delete**: Deletes specified values from the given attribute. Removes the entire attribute if no values are specified or if all existing values of the attribute are listed for deletion.
- **Replace**: Replaces all existing values of the given attribute with the specified values, and creates the attribute if it does not already exist. Using replace with no specified values deletes the entire attribute.

Distinguished Name Field and the DN Flags Field

Use the Distinguished Name field, PIN_FLD_DN, and the Distinguished Name flags field, PIN_FLD_DN_FLAGS with one of its values to specify the locations in the directory tree shown in Table 3-1.

Table 3-1 PIN_FLD_DN Settings

Value	Meaning
0	Complete DN. To specify a complete DN, pass the complete DN of the entry in the PIN_FLD_DN field. You can optionally set PIN_FLD_DN_FLAGS to **0**. For more detailed information and an example, see "Using a Complete Distinguished Name".
1	Prefixed DN. To specify a prefixed DN, pass the prefixed DN of the entry in the PIN_FLD_DN field and set the PIN_FLD_DN_FLAGS to **1**. A prefixed DN is the case when the module that calls the LDAP Data Manager (typically, PCM_OP_REPL_POL_PUSH) passes in the RDN as a prefix. For more detailed information and an example, see "Using a Prefixed Distinguished Name".
2	Parent DN. You cannot use the parent value as a flag to the following opcodes: • PCM_OP_DELETE_FLDS • PCM_OP_DELETE_OBJ • PCM_OP_WRITE_FLDS You can only use the parent value with PCM_OP_CREATE_OBJ. To specify a parent DN, pass the parent DN of the entry in the PIN_FLD_DN field, and set the PIN_FLD_DN_FLAGS to **2**. For more detailed information and an example, see "Using a Parent Distinguished Name (Create Operation Only)".

The Location Field

You can use the location field, PIN_FLD_LOCATION to override the base **LOCATION** value specified in the mapping file. The override is a one shot override for that particular operation only. This lets you specify a different tree root location (base DN) for the directory server entry. For more detailed information and an example, see "Overriding the Base Dn Location".

Creating Directory Server Entries

This section describes the default control logic of the LDAP create operation.

For information on specifying directory tree entries, see "Specifying Directory Tree Entries".

To create new directory server entries or reuse entries in the directory server for replication purposes, use the LDAP PCM_OP_CREATE_OBJ base opcode.

The LDAP PCM_OP_CREATE_OBJ base opcode performs the following operations:

- Makes entries compatible with BRM by adding these BRM fields:
  - Portal object ID (POID)
  - PIN_FLD_POID
  - CREATED_T
  - MOD_T
- Composes the Distinguished Name (DN).
- Accepts the Distinguished Name (DN) of the entry as an input.

PCM_OP_CREATE_OBJ requires the following inputs:

- PIN_FLD_POID. Use the replica POID for this field.
- PCM_OP_USE_POID_GIVEN opflag.
- Other fields depending on the object you create.

> **Note**
>
> When writing array elements, this opcode does not use the element ID.

If you specify a Distinguished Name for the PIN_FLD_DN field in the input flist, the LDAP Data Manager creates a directory entry with the DN that you provide. If the entry already exists, this opcode ignores the error and performs a modify directory operation instead of an add directory operation.

> **Note**
>
> The modify directory operation updates all attributes in an entry.

For more information see "Semantics for the LDAP Modify Operation".

Distinguished Name Control Logic for PCM_OP_CREATE_OBJ

You can specify the location in the directory tree by using the PIN_FLD_DN, PIN_FLD_DN_FLAGS as well as the PIN_FLD_LOCATION fields:

- If you use the PIN_FLD_DN in the input flist, the create opcode creates an entry with the value of the DN you provide.
- If you do not supply a DN for the PIN_FLD_DN field, the LDAP Data Manager uses the RDN_PIECE in the mapping file to compose the Relative Distinguished Name (RDN).
- If you use the PIN_FLD_DN field without the PIN_FLD_DN_FLAGS in the input flist, then the create opcode assumes you want to pass in a *complete* DN. This sets the PIN_FLD_DN_FLAGS field to **0**.
- If you use the PIN_FLD_DN field with the PIN_FLD_DN_FLAGS field set to **1**, then the create opcode appends the **LOCATION** value to construct the DN. This is the *prefixed* DN.
- If you use the PIN_FLD_DN field with the PIN_FLD_DN_FLAGS field set to **2**, then the create opcode appends the LOCATION to PIN_FLD_DN. It then appends this to the value specified for the attribute that is tagged as the RDN_PIECE in the mapping file. This is the *parent* DN case.

> **Note**
>
> You can specify only a parent DN for the create opcode.

If you use the PIN_FLD_LOCATION field in the input flist, the create opcode overrides the location value that you specified in the mapping file for this operation. If you do not use this field in the input flist, the create opcode assumes that you want to use the location value you specified in the mapping file.

Pre-Existing Distinguished Names

If you supply a DN in the input flist for PCM_OP_CREATE_OBJ, and an entry with the same DN exists in the directory server, BRM adds the information to the existing entry based on the information that you provide.

LDAP Data Manager does not treat pre-existing entries as errors.

Supplying Distinguished Names

If you supply a DN to the LDAP Data Manager for the PCM_OP_CREATE_OBJ operation, BRM uses it to create a directory server entry. This lets you use existing LDAP customer entries and also gives you flexibility in assigning DNs.

Not Supplying Distinguished Names

If you do not supply a DN to the LDAP Data Manager for the PCM_OP_CREATE_OBJ operation, BRM uses the attribute tagged with the key RDN_PIECE in the mapping file as the RDN.

For example, the LOCATION `"o=example.com"` is appended to the value specified for the attribute cn (tagged as RDN_PIECE) in the mapping file to compose the REAL DN, `"cn=john,o=example.com"`.

Understanding Matching Rules for Distinguished Names

You control the location of directory tree entries by using a combination of values in the mapping file and by using DN qualifiers, or location parameters at run time as inputs to the BRM LDAP create, delete, read, and write operations.

LDAP Data Manager uses static and dynamic matching rules for DNs to determine the location for directory server entries in the directory tree.

Using Static Controls for DNs

The LDAP Data Manager lets you specify the location of directory server entries statically for each object type by using the LOCATION key in the mapping file. You specify different LOCATION values for different object types as follows:

```
STORABLE CLASS /info1 IMPLEMENTATION LDAPV3 {
LOCATION = "ou=portal,c=US";
}
STORABLE CLASS /info2 IMPLEMENTATION LDAPV3 {
LOCATION = "ou=abc,o=xyz,c=US";
}
```

Using Dynamic Controls for DNs

LDAP Data Manager lets you manipulate the static controls in the mapping file using the control field for DNs, the PIN_FLD_DN field, and a flag for the DN PIN_FLD_FLAGS.

Rule matching for the given DN with the LOCATION field allows entries to be at arbitrary depths relative to the location instead of limiting entries to one location.

For example, assume the LOCATION field is set to the following:

```
LOCATION = "o=example.com"
```

In this case, the following locations are accepted:

```
PIN_FLD_DN = "cn=john, ou=dev, ou=engg, o=example.com"
```

Deleting Directory Server Entries

To delete directory server entries, use the LDAP PCM_OP_DELETE_OBJ base opcode to perform the delete object operation. This opcode invokes the delete entry semantics of the LDAP modify operation in the directory server.

To delete an entry from an existing directory server, create the PCM_OP_DELETE_OBJ input flist. Specify the complete POID or the DN of the entry in the flist.

- If you specify the DN, make sure that the POID is a type-only POID, for example:

  ```
  POID [0] 0.0.5.1 /r_user -1
  ```

- By default, the delete entry operation expects a complete DN. You can optionally supply a prefixed DN.
  - To set up a complete DN, see "Using a Complete Distinguished Name".
  - To set up a prefixed DN, see "Using a Prefixed Distinguished Name".

PCM_OP_DELETE_OBJ performs the following operations:

- Invokes the *delete* entry semantics of the LDAP *modify* operation in the directory server.
- If the *pinpoid* attribute does not form the Relative Distinguished Name (RDN), this opcode locates the directory server entry first using a search operation and then deletes the object.

You must supply the PIN_FLD_POID (replica POID ID) and the POID of the object that you want to delete in the input flist.

For more information see "Semantics for the LDAP Modify Operation".

Changing Directory Server Entries

You can manage changes to directory server entries by performing the following operations:

- Adding Attributes to an Existing Directory Server Entry
- Deleting Attributes from an Existing Directory Server Entry
- Renaming Directory Server Entries
- Creating Subclass Objects in the Directory Server
- Creating Related Entries Under One Node

The next sections describe how to accomplish these tasks.

For information on specifying directory tree entries, see "Specifying Directory Tree Entries".

Adding Attributes to an Existing Directory Server Entry

You can add an attribute that already exists in the directory server schema to an instance of an object defined in the mapping file. This lets you access and modify the schema of the directory server entry by using the BRM API.

Follow these rules when you add an attribute to the directory server:

- If an attribute belongs to an auxiliary class, specify the attribute's auxiliary class in the mapping file.
- An attribute can belong to only one auxiliary class.
- Attribute names must be unique across object classes.
- Attributes being added must already exist in the directory schema.

If you add an attribute that belongs to an auxiliary class to an existing directory server entry, the auxiliary class is automatically added to the entry.

To update or rename an entry, use the LDAP PCM_OP_WRITE_FLDS base opcode.

This opcode performs the following operations:

- Updates attributes by using the LDAP *modify* operation
- Uses the *replace* semantics of the LDAP *modify* operation
- Renames directory server entries

When you specify an array field, the entire array is replaced. The PCM_OPFLG_ADD_ENTRY opflag is used to invoke the *add* semantics, thereby adding new values to an existing entry.

You must supply the PIN_FLD_POID (replica POID ID) and at least one field that you want to write in the input flist. For rename operations, you can only specify the RDN piece as it is specified in the mapping file. You must also set the deleteOldRdn entry in the LDAP Data Manager **pin.conf** file to **1**.

For more information see "Semantics for the LDAP Modify Operation".

For the procedure on renaming a directory server entries, see "Renaming Directory Server Entries".

To add an attribute to an existing directory server entry:

1. Create the PCM_OP_WRITE_FLDS input flist:
   a. Add the complete POID or DN of the entry.

      If you specify the DN, make sure that the POID is a type-only POID, for example, **POID [0] 0.0.5.1 /r_user -1**.

   b. (Optional) Specify whether the DN is a complete or prefixed DN.

      By default, the write fields operation expects a complete DN. You can optionally supply a prefixed DN.

   To set up a complete DN, see "Using a Complete Distinguished Name".

   To set up a prefixed DN, see "Using a Prefixed Distinguished Name".
2. In the mapping file, define all possible attributes including the auxiliary class that the directory server entry might contain.
3. For the attribute you are adding:
   a. Set the CREATE string to **Optional**.
   b. Specify the auxiliary class that the attribute belongs to.

   In this sample, PIN_FLD_HTTP_URL corresponds to the attribute **labeleduri** for an auxiliary class called **labeleduriobject**.

```
STORABLE CLASS /r_user {
...
STRING PIN_FLD_HTTP_URL {
CREATE = Optional;
MODIFY = Writeable;
}
STRING PIN_FLD_TYPE_STR {
CREATE = Optional;
MODIFY = Writeable;
}
...
}
STORABLE CLASS /r_user IMPLEMENTATION LDAPV3 {
...
STRING PIN_FLD_HTTP_URL {
ATTRIBUTE = "labeleduri";
OBJECTCLASS = "labeleduriobject";
}
STRING PIN_FLD_TYPE_STR {
ATTRIBUTE = "memberurl";
OBJECTCLASS = "mylabeleduriobject";
}
.......
}
```

Deleting Attributes from an Existing Directory Server Entry

To delete an attribute from an existing directory server entry, follow these rules:

- Delete only those attributes that you have defined in the mapping file and are tagged as optional.
- Do not delete fields that you have used for the RDN or any fields tagged as required. You can delete an entire array by using PIN_ELEM_ID_ANY.

> **Note**
>
> Before you delete an attribute, make sure that doing so does not violate the schema constraints of the directory server entry. If you attempt to violate schema constraints, the LDAP Data Manager reports an object violation error.

For more information see "Semantics for the LDAP Modify Operation".

To delete values and attributes in an entry, use the LDAP PCM_OP_DELETE_FLDS base opcode. This opcode performs the delete operation by using the LDAP *modify* operation, which imposes delete semantics.

To delete an attribute from an existing directory server entry, create the PCM_OP_DELETE_FLDS input flist.

1. Specify the complete POID or DN of the entry.

   You must supply the PIN_FLD_POID (replica POID) and at least one field that you want to delete in the input flist.

   If you specify the DN, make sure that the POID is a type-only POID, for example:

   ```
   POID [0] 0.0.5.1 /r_user -1
   ```

2. (Optional) Specify whether the DN is a complete or prefixed DN.
3. By default, the delete fields operation expects you to supply a complete DN. You can optionally supply a prefixed DN.
   - To set up a complete DN, see "Using a Complete Distinguished Name".
   - To set up a prefixed DN, see "Using a Prefixed Distinguished Name".
4. Provide the names of the fields that you want to delete.

   ```
   0 PIN_FLD_<field to be deleted>
   ...
   ...
   ```

   The LDAP Data Manager deletes the attribute and the auxiliary object class if it is the last attribute belonging to that auxiliary class.

Renaming Directory Server Entries

You can rename a directory server entry in BRM, which is helpful when a customer's name or login name (the RDN part) changes. For example, if you need to update the entry for Jane Doe to reflect her new name, Jane Smith, you would change the directory entry from:

```
cn=Jane Doe; o=example.com; c=US
```

to this:

```
cn=Jane Smith; o=example.com; c=US
```

When renaming directory server entries, follow these guidelines:

- Do not rename the entry such that it moves to a different part of the directory tree.
- Do not specify any field other than the field tagged by the RDN_PIECE in the mapping file in the input flist on this operation.
- Rename only the left-most value in the DN entry. In the example above, this is the cn value.

> **Note**
>
> When using the write fields operation to rename entries, ensure that the input file list includes no other fields.

LDAP Data Manager uses the PCM_OP_WRITE_FLDS opcode to rename entries. It also references an entry in the LDAP Data Manager configuration file (*BRM_home***/sys/dm_ldap/pin.conf**) to determine whether to remove the old RDN value from the entry. For more information, see "Semantics for the LDAP Modify Operation".

To propagate a name change from BRM to the directory server entry:

1. Create the PCM_OP_WRITE_FLDS input flist by adding the POID or complete DN of the old entry and the new name of the entry:

```
0 PIN_FLD_POID POID [0]
0.0.5.1 /r_user <account_number>
0 PIN_FLD_LOGIN STR [0] "Jane Smith"
```

2. In the **pin.conf** file, uncomment the following entry and set the delete value to **1**.

```
- ldap_ds deleteOldRdn 1
```

> **Note**
>
> When you set the delete value to **1**, LDAP Data Manager deletes the old RDN (Jane Doe). To keep the old RDN, set the delete value to **0**.

BRM retrieves the new value of the attribute tagged as RDN_PIECE from the input flist in the mapping file and uses it to rename the entry.

Creating Subclass Objects in the Directory Server

You can extend, or *subclass,* an existing object class to add attributes to it. In this case, you use the create opcode to create subclass objects in any location in the directory server. You can create entries for a given BRM subclass object type in a particular part of the directory information tree independent of the location of objects belonging to the parent class. For example, **/service** objects can be located in the following location:

```
ou=Services, o=example.com, c=US
```

The **/service/email** subclass object can be located in a different location:

```
ou=ServiceEmail, o=example.com, c=US
```

To create a subclass object in the directory server independent of its parent's location:

1. Using your directory server tools, create the organizational unit (**ou**) for the subclass object in the directory server.

2. Create the corresponding subclass mapping for this attribute by specifying the LOCATION key in the implementation definition section of the mapping file.
3. Start LDAP Data Manager and make sure that the **dm_ldap.pinlog** file reports no errors.
4. Create the input flist that contains the complete POID or DN of the object that you want to create.

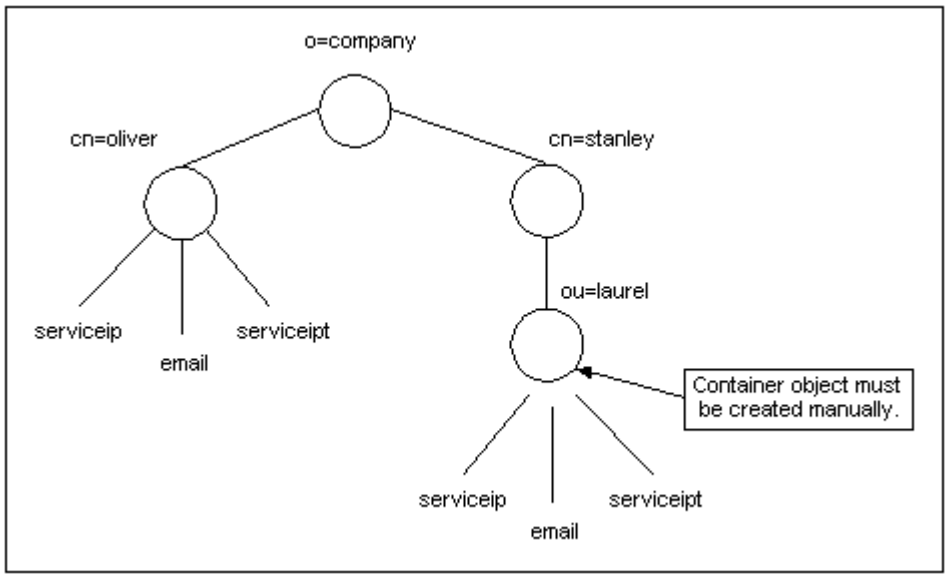
Creating Related Entries Under One Node

You can group *any* set of related entries under one node in the directory tree. To set up your BRM LDAP environment to replicate data in this type of structure, you use the link object attribute to define the related node entries in the mapping file.

For example, you can create all services associated with a particular account under the account entry in the directory tree.

If the creation of linked objects requires containers, this feature will not create the container objects automatically. You must create the containers for these objects with your directory server tools manually, as Figure 3-1 shows.

Figure 3-1 Manually Created Containers Example



In each top-level service, you must use a link attribute that has the POID of the account that the service is linked to.

To group all services under an account:

> **Note**
>
> This procedure assumes that your related services are linked.

1. In the mapping file, configure the link object for each class and link attribute for each top-level class as in this example:

> **Note**
>
> The link attribute must be a POID. Specify the types and link attributes for all classes.

```
STORABLE CLASS /r_user {
...
}
##
## /r_service (to represent the /service storable class)
##
STORABLE CLASS /r_service {
# list all the /r_service fields

...
       POID PIN_FLD_PARENT {
             CREATE = Optional;
             MODIFY = Writeable;
}
....
}

## /r_service/ip (to represent the /service/ip storable class)
STORABLE CLASS /r_service/ip {
# list all the /r_service/ip related fields

...
       POID PIN_FLD_PARENT {
             CREATE = Optional;
             MODIFY = Writeable;
}
....
}
## /r_service/email (to represent the /service/email storable class)
STORABLE CLASS /r_service/email {
# list all the /r_service/email related fields

...
       POID PIN_FLD_PARENT {
             CREATE = Optional;
             MODIFY = Writeable;
}
....
}

STORABLE CLASS /r_user IMPLEMENTATION LDAPV3 {
       ATTRVAL = "objectClass: top";
       ATTRVAL = "objectClass: person";
       ATTRVAL = "objectClass: inetOrgPerson";

       ENTRY_TYPE = "ruser";

       LOCATION = "o=example.com";
...
}
       # For LDAP Manager

STORABLE CLASS /r_service IMPLEMENTATION LDAPV3 {
       ATTRVAL = "objectClass: top";
       ENTRY_TYPE = "rservice";
```

```
        LINK_OBJECT = "/r_user";
...
        POID PIN_FLD_PARENT {
        ATTRIBUTE = "parent";
        LINK_ATTRIBUTE = 1;
}
...
}
#
STORABLE CLASS /r_service/ip IMPLEMENTATION LDAPV3 {
# provide mapping for /r_service/ip fields
LINK_OBJECT = "/r_user";
# Provide mapping for /r_service fields.
# The link attribute is inherited from the parent.
...
        POID PIN_FLD_PARENT {
        ATTRIBUTE = "parent";
        LINK_ATTRIBUTE = 1;
}
...
}
#STORABLE CLASS /r_service/email IMPLEMENTATION LDAPV3 {
# Provide mapping for /r_service/email fields
# The link attribute is inherited from the parent.
        LINK_OBJECT = "/r_user";
...
        POID PIN_FLD_PARENT {
        ATTRIBUTE = "parent";
        LINK_ATTRIBUTE = 1;
}
...
}
```

> **Note**
>
> To locate the parent **/r_user** object, the LDAP Data Manager uses the LOCATION attribute of the **/r_user** class as the base DN.

2. Specify one of the flexible DN assignment rules to dynamically manipulate the static behavior defined in the mapping file.
   - If the DN flag indicates that the DN is a complete DN, the DN supplied is used to create the **/r_service** entry.
   - If the DN flag indicates that the DN is a prefix, the DN of the located.
   - **/r_user** entry is appended to the DNs supplied to compose the DN of the **/r_service** entry.
   - If the DN flag indicates that the DN is a parent, the **/r_service** DN supplied is appended to the value of the attribute tagged as the RDN piece. The DN of the located **/r_user** entry is appended to this string.
   - If you do not supply a DN, then the value of the DN of the **/r_user** entry will be appended to the RDN of **/r_service**.

Specifying Directory Tree Entries

You specify directory server entries by using the DN field, the DN qualifier field flags, and the location parameter. The DN field and the DN qualifiers let you manipulate location of entries for your directory tree, while the location parameter lets you override the mapping file location.

This section describes the controls you can use as inputs to the create, delete, read, and write operations either to manipulate directory entries or to override the base location value.

Using a Complete Distinguished Name

A *complete* DN is the absolute root in the directory tree for the directory server entry. To specify a complete DN, pass the complete DN of the entry in the PIN_FLD_DN field. You can optionally set PIN_FLD_DN_FLAGS to **0**.

This example shows how to specify a complete DN in the input flist to the create, delete, read, and write operations at run time:

```
LOCATION = "o=example.com"
PIN_FLD_DN = "cn=john, ou=dev, ou=engg, o=example.com"
PIN_FLD_DN_FLAGS=0

REAL DN = "cn=john, ou=dev, ou=engg, o=example.com"
```

For a complete DN, PIN_FLD_DN must match the LOCATION. For example, if the LOCATION is `"o=example.com"`, then the last element of PIN_FLD_DN must be `"o=example.com"`. The PIN_FLD_DN is the REAL DN in this case.

Using a Prefixed Distinguished Name

The *prefixed* DN is the RDN location in the directory tree for the directory server entry. To specify a prefixed DN, pass the prefixed DN of the entry in the PIN_FLD_DN field and set the PIN_FLD_DN_FLAGS to **1**.

This example shows how to specify a prefixed DN in the input flist to the create, delete, read, and write input flists at run time:

```
LOCATION = "o=example.com"
PIN_FLD_DN = "cn=john, ou=dev, ou=engg"
PIN_FLD_DN_FLAGS=1

REAL DN = "cn=john, ou=dev, ou=engg, o=example.com"
```

The **LOCATION** `"o=example.com"` is appended to PIN_FLD_DN `"cn=john, ou=dev, ou=engg"` to compose the REAL DN `"cn=john, ou=dev, ou=engg, o=example.com"`.

Using a Parent Distinguished Name (Create Operation Only)

A *parent* DN is the location of the parent of the entry in the directory tree. You can pass a parent DN only in the create operation (PCM_OP_CREATE_OBJ).

To specify a parent DN, pass the parent DN of the entry in the PIN_FLD_DN field, and set the PIN_FLD_DN_FLAGS to **2**.

This example shows how to specify a parent DN in the input flist to PCM_OP_CREATE_OBJ at run time:

```
LOCATION = "o=example.com"
PIN_FLD_DN = "ou=dev, ou=engg"
PIN_FLD_DN_FLAGS=2

REAL DN = "cn=john, ou=dev, ou=engg, o=example.com"
```

The LDAP Data Manager appends the LOCATION `"o=example.com"` to PIN_FLD_DN `"ou=dev, ou=engg"` and appends the value `"ou=dev, ou=engg, o=example.com"` to the value "**john**" (**cn**) tagged as RDN_PIECE in the mapping file to compose the REAL DN `"cn=john, ou=dev, ou=engg, o=example.com"`.

Overriding the Base Dn Location

A *base* DN is the root location in the directory tree for the directory server entry. You can use the location field, PIN_FLD_LOCATION to dynamically override the **LOCATION** value specified in the mapping file for the create, delete, delete field, and write field operations. Overrides affect the immediate operation only.

For example, if you defined this location in the mapping file (**ldap.idl**):

```
ou=engg, o=example, dc=US
```

To create this entry:

```
mail=login1@example.com.us, ou=engg, o=example, dc=US
```

And you want to create another entry in a different location such as:

```
ou=techdocs, o=mycompany, dc=US
```

You can override the value specified for the location in the mapping file to create the following entry:

```
mail=login2@example.com.us, ou=techdocs, o=mycompany, dc=US
```

To do this, you override the mapping file location by using the PIN_FLD_LOCATION field as an input to the create operation. This lets you create accounts in a different tree without LDAP Data Manager checking the **LOCATION** value in the mapping file.

To specify a new base location for a directory server entry, enter the new base location in the input flist:

```
0 PIN_FLD_LOCATION STR[0] = "mail=login2@example.com.us, ou=techdocs, o=mycompany, dc=US"
```

When subsequent operations are performed on this entry, you should override the mapping file location by using the PIN_FLD_LOCATION field as an input.

Reading and Searching for Directory Server Entries

You can do these operations on directory server entries:

- Reading Objects from the Directory Server
- Reading Attributes from the Directory Server Entry

- Searching the Directory Server for Entries

Reading Objects from the Directory Server

You can read objects from the directory server and return an flist containing fields and values for each attribute in the directory server entry.

To read fields in a directory server entry, use the LDAP PCM_OP_READ_FLDS base opcode. This opcode reads attributes from a directory server entry from the database using the LDAP search operation.

The LDAP Data Manager parses the mapping file (**ldap.idl**). After this file is parsed, the LDAP Data Manager creates a mapping class that contains a mapping of the **/r_user** class attributes to BRM fields.

The read fields opcode accepts the POID of the object that you want to read. If the POID is a type-only POID, and you provide a DN, the LDAP Data Manager uses the DN to locate the object in the directory server.

You supply the list of the fields that you want to read from the object. These fields correspond the attributes of the directory server object. For each entry that you provide in the input flist, the LDAP Data Manager queries the mapping class for the corresponding attribute name to generate the list of attributes names.

> **Note**
>
> You can read only those objects that you have defined in the mapping file.

To read an object:

1. Create the PCM_OP_READ_OBJ input flist.
2. Specify the complete POID or DN of the directory server entry.

> **Note**
>
> If you supply a DN then you must also supply a type-only POID. A type-only POID is a POID with a value set to -1.

Object Read examples

These examples explain how to read objects using a complete POID and a DN:

Read Object Using a Complete POID

```
0 PIN_FLD_POID           POID [0] 0.0.5.1 /r_user 71131 0
0 PIN_FLD_LOCATION       STR [0] "o=example.com"
```

Read Object Using a DN

```
0 PIN_FLD_POID           POID [0] 0.0.5.1 /r_user -1
0 PIN_FLD_LOCATION       STR [0] "o=example.com"
0 PIN_FLD_DN             STR [0] "uid=link51, o=example.com"
0 PIN_FLD_DN_FLAGS       INT [0] 0
```

Reading Attributes from the Directory Server Entry

The steps to read attributes from the directory server are similar to reading objects. However, you need to pass the fields corresponding to the attributes of the directory server object that you want to read in the input flist to PCM_OP_READ_FLDS as opposed to the objects.

To read attributes from the directory server entry:

1. Create the PCM_OP_READ_FLDS input flist.
2. Specify the complete POID or DN of the entry.

> **Note**
>
> If you supply a DN then you must also supply a type-only POID. A type-only POID is a POID with a value set to **-1**.

3. Pass the fields of the attributes that you want to read in the input flist.

Attribute Read Examples

These examples show how to read attributes by using a complete POID and a DN:

Read Attributes Using a Complete POID

```
0 PIN_FLD_POID            POID [0] 0.0.5.1 /r_user 71148
0 PIN_FLD_LOCATION         STR [0] "o=example.com"
0 PIN_FLD_ACCOUNT_NO       STR [0] 0.0.0.1 /r_account 71145 13
0 PIN_FLD_ACCOUNT_OBJ     POID [0] 0.0.0.1 /r_account 71145
0 PIN_FLD_ACCOUNT_STATUS   STR [0] "Active"
0 PIN_FLD_LOGIN            STR [0] "link45"
0 PIN_FLD_NAME             STR [0] "link45"
0 PIN_FLD_FIRST_NAME       STR [0] "link45"
0 PIN_FLD_LAST_NAME        STR [0] "link45"
```

Read Attributes Using a DN

```
0 PIN_FLD_POID            POID [0] 0.0.5.1 /r_user    -1
0 PIN_FLD_LOCATION         STR [0] "o=example.com"
0 PIN_FLD_DN               STR [0] "uid=link45, o=example.com"
0 PIN_FLD_DN_FLAGS         INT [0] 0
0 PIN_FLD_ACCOUNT_NO       STR [0] 0.0.0.1 /r_account 71145 13
0 PIN_FLD_ACCOUNT_OBJ     POID [0] 0.0.0.1 /r_account 71145
0 PIN_FLD_ACCOUNT_STATUS   STR [0] "Active"
0 PIN_FLD_LOGIN            STR [0] "link45"
0 PIN_FLD_NAME             STR [0] "link45"
0 PIN_FLD_ADDRESS          STR [0] "sdsdfs, sfsdfsd, CA 12345, USA"
```

Searching the Directory Server for Entries

You can search the directory server for entries that match a specified search filter. For example, you can search for an entry that has a **login** attribute. You can search all entries with the **login** attribute within a tree or sub tree.

> **Note**
>
> *Only* those objects and attributes that you define in the mapping file can be returned by the LDAP Data Manager in the output flist.

To find LDAP objects, use the LDAP PCM_OP_SEARCH base opcode. This opcode searches the directory server based on a specified search criteria that you supply as a template in the input flist.

> **Note**
>
> *Only* those objects and attributes that you define in the mapping file can be returned by the LDAP Data Manager in the output flist.

The LDAP Data Manager parses the mapping file (**ldap.idl**). After this file is parsed, the LDAP Data Manager creates a mapping class that contains a mapping of the **/r_user** class attributes to BRM fields.

You must supply the POID of the object for the search operation. The search opcode uses the base Distinguished Name (DN) of the POID type as a base for the search operation. You supply a search filter to this opcode as a template in PIN_FLD_TEMPLATE. The template is the filter expression with attribute names and literal values.

For each entry that matches the given search criteria, the search call returns the value of the attributes in the attribute list, if they exist in the directory entry. For multi-valued attributes an array of values is returned. This call does not return an error if it is queried for an attribute that does not exist in the directory entry.

For each attribute value or values, the class map is queried for the BRM field name corresponding to the attribute name and an entry is added to the output flist with the corresponding value for each attribute. For array attributes that are multi-valued, an array entry is created for each value.

1. Select a search filter. See "Using the Sample LDAP Search Filters".
2. Create the PCM_OP_SEARCH input flist.
   a. Add the POID for the PIN_FLD_POID field.
   b. Select the search filter to use as a template for the PIN_FLD_TEMPLATE field.
   c. Set PIN_FLD_ARGS to specify the arguments and value of arguments to be substituted in the search filter.
   d. Set the search scope. See "Setting the Search Scope".
3. Call the opcode to perform the search.

   The search opcode replaces the search template criterion with attribute names (An) and values (Vn). Attribute names and values are specified in pin_fld_args[n]. In the example below, the filter translates into this search:

   ```
   (&(cn-*link*)(ipstatus r=Active))
   ```

Setting the Search Scope

You can set the search scope on the directory by using the PIN_FLD_SCOPE field. The PIN_FLD_SCOPE field can have the values listed in Table 3-2.

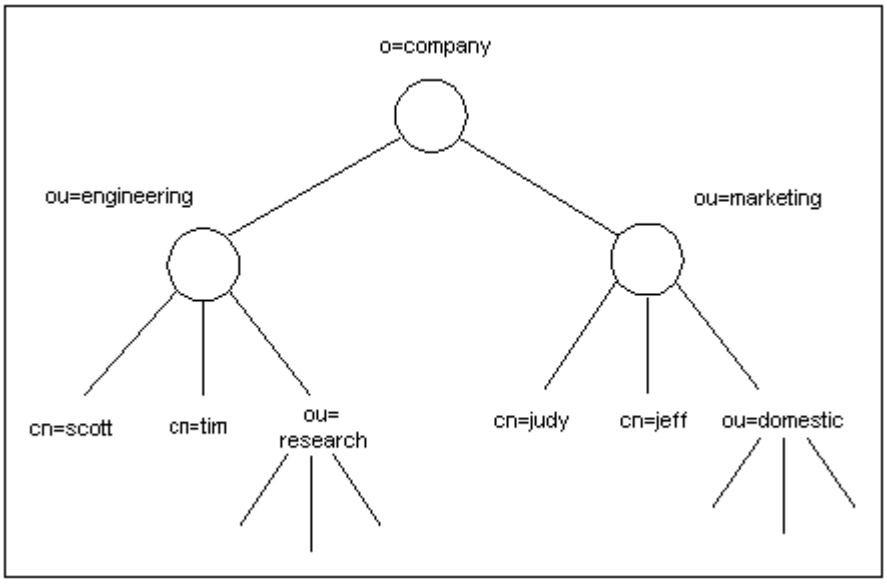
Table 3-2 PIN_FLD_SCOPE Settings

Value	Meaning
LDAP_SCOPE_SUBTREE	Default value. Searches the entire directory tree.
LDAP_SCOPE_ONELEVEL	Searches one level down the directory tree.

The search opcode returns each attribute value or values as a result in the output flist.

Figure 3-2 shows a sample directory tree.

Figure 3-2 Sample Directory Tree



If the search scope is 0 PIN_FLD_SCOPE ENUM [0] 2, entries in all branches beneath the location are searched. In this example, the searched entries include **cn=scott**, **cn=tim**, and all entries in branches under **ou=research**. If the search scope is 0 PIN_FLD_SCOPE ENUM [0] 1, only entries immediately below the location are searched. In this example, these include only **cn=scott** and **cn=tim**.

Specifying the Base DN

The object type (**/r_user**) determines the base DN of a search. The value specified for the location for **/r_user** in the IDL file is used as the base DN. You can override this by specifying the value for PIN_FLD_LOCATION in the input flist. The results of the search are mapped to this object class.

Searching from Different Locations

If all base class and subclass entries are under the same location, you can perform a single search from that location to find entries that are instances of the base class and subclasses that satisfy the search criteria. If the entries are under different locations, you must perform a separate search under each location to find the instances.

Example Service Storable Class Tree and Search

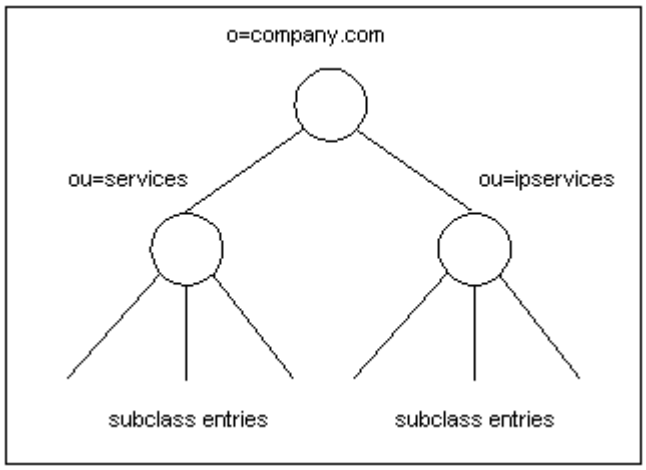
This is a sample service storable class definition from an IDL file:

```
STORABLE CLASS /r_service IMPLEMENTATION LDAPV3 {
ATTRVAL = "objectClass: top";
ENTRY_TYPE = "rservice";
LOCATION = "ou=services, o=example.com";
.....}

STORABLE CLASS /r_service/ip IMPLEMENTATION LDAPV3 {
# provide mapping for /r_service/ip fields
LOCATION = "ou=ipservices, o=example.com";
.....}
```

Figure 3-3 shows the directory tree.

Figure 3-3 Directory Tree



To locate base class and subclass entries, you must perform a search in each of the locations:

```
0 PIN_FLD_POID           POID [0] 0.0.5.1 /r_service -1
0 PIN_FLD_TEMPLATE        STR [0] "(A1 = V1)"
0 PIN_FLD_SCOPE          ENUM [0] 2
0 PIN_FLD_ARGS          ARRAY [1] allocated 20, used 1
1 PIN_FLD_LOGIN           STR [0] "*john*"
Summary of LDAP API operations

0 PIN_FLD_POID           POID [0] 0.0.5.1 /r_serviceip -1
0 PIN_FLD_TEMPLATE        STR [0] "(A1 = V1)"
0 PIN_FLD_SCOPE          ENUM [0] 2
0 PIN_FLD_ARGS          ARRAY [1] allocated 20, used 1
```

```
1 PIN_FLD_LOGIN           STR [0] "*john*"
Summary of LDAP API operations
```

Using the Sample LDAP Search Filters

Table 3-3 shows sample search filters that you use as templates to search the directory server entries. For information on how to search the directory server entries, see "Searching the Directory Server for Entries".

Table 3-3 LDAP Search Filters

Filter Example	Arguments	Modified Filter	Matches
(F1=V1)	0 PIN_FLD_ARGS ARRAY [1] allocated 10, used 1 1 PIN_FLD_NAME STR [0] "*bert*"	(cn=*bert*)	All entries with the string "bert" somewhere in the name.
(F1>=Fred)	0 PIN_FLD_ARGS ARRAY [1] allocated 10, used 1 1 PIN_FLD_NAME STR [0] "Fred"	(cn>=Fred)	All entries with a common name that is lexicographically greater than "Fred".
(&(objectclass=person) (F1=V1))	0 PIN_FLD_ARGS ARRAY [1] allocated 10, used 1 1 PIN_FLD_EMAIL_LOGIN STR [1] "*"	(&(objectclass=person) (maillogin=*))	All people with an email address.
(F1~=V1)	0 PIN_FLD_ARGS ARRAY [1] allocated 10, used 1 1 PIN_FLD_LAST_NAME STR [0] "Jensin"	(sn~=Jensin)	All entries with a surname approximately equal to Jensin.
(! (F1=V1))	0 PIN_FLD_ARGS ARRAY [1] allocated 10, used 1 1 PIN_FLD_EMAIL_LOGIN STR [0] "*"	(! (mail=*))	Entries without a mail attribute.
(F1=V1)	0 PIN_FLD_ARGS ARRAY [1] allocated 10, used 1 1 PIN_FLD_POID_STR [0] "0.0.6.1 / r_user 9999 0"	(pinpoid=0.0.6.1 /r_user 9999 0)	Entries that have a *pinpoid* equal to the value specified.
(F1=V1)	0 PIN_FLD_ARGS ARRAY [1] allocated 10, used 1 1 PIN_FLD_DN [0] "cn=joe, ou=People, o=microsoft"	(dn= cn=joe, ou=People, o=microsoft)	Entries that have a DN equal to the value specified.

LDAP Search Limitations

The search does not return objects that are not defined in the mapping file. LDAP Data Manager treats directory server objects that are not defined in the mapping file as an error condition.

Testing Directory Server Connections

To test directory server connections, use the LDAP PCM_OP_TEST_LOOPBACK base opcode. This opcode verifies that the LDAP Data Manager and the directory server daemon/service processes are running and communicating with each other.

BRM LDAP Profile Object

For convenience, the **/profile/ldap** object holds the DN of the directory server entries corresponding to the **/account** objects. This lets you pass in a DN when you create the BRM account. BRM uses the PCM_OP_CUST_COMMIT_CUSTOMER opcode to do this.

This is useful when you create BRM accounts based on customer information already in the directory server. For example, when a new entry is created for a customer in the directory server, the DN of the entry is passed to BRM while creating the corresponding **/account**.

To avoid empty **/profile** objects, the **/profile** object is created *only* if it is passed in during account creation.

Example of how input is passed in when you use a profile object:

```
/profile
          PIN_FLD_POID               MS       POID        profile POID
          PIN_FLD_ACCOUNT_OBJ_POID   MR       POID        Owning Account
          PIN_FLD_NAME               MR       STR[255]    Name("LDAP Information")
          /profile/ldap
                 PIN_FLD_LDAP_INFO   MR       SUBST
                 PIN_FLD_DN          MR       TR[1024]    DN of the Event Object
```

4

Installing LDAP Manager

Learn how to install the Oracle Communications Billing and Revenue Management (BRM) LDAP Manager software.

Topics in this document:

Installing LDAP Manager

> **Note**
>
> If you already installed the product, you must uninstall its features before reinstalling them.

To install LDAP Manager:

1. If you have a previous release of LDAP Manager installed, back up the existing *BRM_home*/**sys/dm_ldap/pin.conf** configuration file. If you have made changes to the *BRM_home*/**sys/ldap.idl** sample mapping file, back up this file also.

   > **Caution**
   >
   > The installation overwrites any pre-existing LDAP Manager **pin.conf** and **ldap.idl** files. If you do not back up the original files, you must manually re-create the settings in the new files to restore your original configuration.

2. Install LDAP Manager. For instructions, see "Installing Individual BRM Components" in *BRM Installation Guide*.
3. If you backed up the original *BRM_home*/**sys/dm_ldap/pin.conf** file, merge the contents of the backup copy into the new **pin.conf** file.
4. If you backed up the original *BRM_home*/**sys/ldap.idl** sample mapping file, replace the newly installed file with your backup file.

Your LDAP Manager installation is now complete.

Uninstalling LDAP Manager

To uninstall LDAP Manager, see "Uninstalling Optional Components" in *BRM Installation Guide*.

5

Configuring LDAP Manager

Learn how to configure the Oracle Communications Billing and Revenue Management (BRM) LDAP Manager.

Topics in this document:

Configuring the LDAP Data Manager

To configure the LDAP DM:

- Set up the **ldap.idl** mapping file.
- Set up the directory server to use a replicatable user object (**/r_user**) for BRM.
- Edit the LDAP DM **pin.conf** file.

Setting Up the Mapping File

When you install the LDAP Manager, a default interface language definition (**.idl**) mapping file is created for you. Use the **ldap.idl** mapping file.

You can find the mapping file in the *BRM_home***/sys/dm_ldap** directory.

The mapping file needs to match your directory server implementation. For more information on how to set up this file, see "LDAP Data Manager Mapping Files".

Setting Up the Directory Server

To set up your directory server with attributes that BRM can understand, such as Portal Object ID (POID), names, addresses, currency, login, and appropriate service information (email and other IP network services), you must create a BRM object-type definition called the replicate user (**/r_user**) object class in your directory server. For more information on setting up this object class in the directory server, see "Determining the /r_user Object Class Attributes".

Editing the LDAP Data Manager Configuration File

1. Open the LDAP DM configuration file (*BRM_home***/sys/dm_ldap/pin.conf**).
2. Edit the standard memory, connection, debugging, and log file entries. See "Using Configuration Files to Connect and Configure Components" in *BRM System Administrator's Guide*.

   The **hostname** and **port** entries identify the machine when the LDAP directory server runs:

```
- ldap_ds       hostname        my_company.com
- ldap_ds       port            port_number
```

3. To specify the mapping file, set the **mapping_file** entry to **ldap.idl**:

```
- ldap_ds       mapping_file    ldap.idl
```

4. To set the Bind Distinguished Name (DN) for authenticating BRM to the directory server, set the bind entry to the Distinguished Name (DN) of the entry you want to use for binding to the directory server.

   For example:

```
- ldap_ds bind uid=admin,ou=Administrators,ou=TopologyManagement, o=NetscapeRoot
```

> **Note**
>
> You can check the directory bind DN by using your directory server tools or you can ask your directory server system administrator for this information.

5. Set the bind password for your LDAP directory server host:

```
- ldap_ds       password        password
```

6. To specify how the LDAP Manager outputs timestamps, edit the **encodeTimestamp** entry.
   - Use **UTCTIMESTRING** to specify a readable format; for example, 20021207135225 (**yyyymmddhhmmss**).
   - Use **UTCTIMEVALUE** to specify a decimal format; for example, **962246667**. This is the default.

```
- ldap_ds    encodeTimestamp    UTCTIMESTRING
```

7. Use the **appendZToTimestamp** entry to append a Z to the end of the time stamp. The default (**0**) does not append a Z.

```
- ldap_ds      appendZToTimestamp   1
```

8. Use the **deleteOldRdn** entry to specify whether to rename distinguished names. See "Renaming Directory Server Entries". By default, the old name is deleted.

```
- ldap_ds       deleteOldRdn         1
```

9. Use the **ops_fields_extension_file** entry to specify the file that contains the definitions of custom field lists.

> **Note**
>
> Include this entry in your LDAP Data Manager **pin.conf** file *only* when you create custom fields for your directory server implementation.

```
- dm_ldap ops_fields_extension_file my_ldap_implementation
```

   For information on how to create custom fields, see "Creating Custom Fields" in *BRM Developer's Guide*.

Configuring the Connection Manager for LDAP Manager

Make sure the LDAP entries have been added to your CM configuration file and edit them as necessary (the **fm_module** entries are preconfigured, but you must *uncomment* them if they are commented out).

To configure the CM for LDAP Manager:

1. Open the Connection Manager configuration file (*BRM_home*/**sys/cm/pin.conf**).
2. Set the **dm_pointer** entry to point to your LDAP Data Manager.

   The default database number for the **dm_pointer** entry is **0.0.5.***x*, where *x* is the number of the BRM database.

   ```
   - cm dm_pointer      0.0.5.X      dm_ldap_host       dm_ldap_port
   ```

3. Do one of the following:
   - For the user mapping scheme, leave the mapping scheme entry (**user_scheme)** set to **1**. This is the default.
   - For the one-to-one mapping scheme, set **user_scheme** to **0**.

   ```
   - fm_repl_pol      user_scheme    1
   ```

Configuring the LDAP Data Manager for Multiple Schemas

You can configure a BRM system to use multiple database schemas as well as multiple LDAP Data Managers. For example:

```
- cm            dm_pointer      0.0.0.1 ip 198.51.100.13 56971   # Oracle
- cm            dm_pointer      0.0.0.2 ip 198.51.100.13 56972   # Oracle
- cm            dm_pointer      0.0.0.3 ip 198.51.100.13 56973   # Oracle
- cm            dm_pointer      0.0.5.1 ip 198.51.100.13 56981   # DM LDAP
- cm            dm_pointer      0.0.5.2 ip 198.51.100.13 56982   # DM LDAP
- cm            dm_pointer      0.0.5.3 ip 198.51.100.13 56983   # DM LDAP
```

In addition to setting these entries, you must define the LDAP DM's database number in the PIN_FLD_CONSUMER_OBJ field in the **/channel** object. The PCM_OP_REPL_POL_PUSH policy opcode retrieves this database number and sends the data to that LDAP DM. For more information, see "About Channels and Data Propagation".

Configuring the LDAP Data Manager with Different LDAP Data Manager Pointers

You can set different LDAP Data Manager pointers to reference the same host and port combination. For example:

```
- cm            dm_pointer      0.0.5.1 ip 198.51.100.13 56981   # DM LDAP
- cm            dm_pointer      0.0.5.2 ip 198.51.100.13 56981   # DM LDAP
- cm            dm_pointer      0.0.5.3 ip 198.51.100.13 56981   # DM LDAP
```

Configuring Event Notification for LDAP Manager

To trigger updates to the LDAP database, BRM uses event notification.

Before you can use LDAP Manager, you must configure the event notification feature as follows:

1. If your system has multiple configuration files for event notification, merge them. See "Merging Event Notification Lists" in *BRM Developer's Guide*.
2. Ensure that the merged file includes the entire event notification list in the *BRM_home***/sys/data/config/pin_notify.ldap** file.
3. (Optional) If necessary to accommodate your business needs, add, modify, or delete entries in your final event notification list. See "Editing the Event Notification List" in *BRM Developer's Guide*.
4. (Optional) If necessary to accommodate your business needs, create custom code for event notification to trigger. See "Triggering Custom Operations" in *BRM Developer's Guide*.
5. Load your final event notification list into the BRM database. See "Loading the Event Notification List" in *BRM Developer's Guide*.

For more information, see "Using Event Notification" in *BRM Developer's Guide*.

Loading the LDAP Price List into Pricing Center

LDAP Manager includes a price list that includes a plan that uses the LDAP service. Use Pricing Center to add the LDAP plans to your price list, or use the **loadpricelist** utility as shown below:

```
loadpricelist -v -cf BRM_home/setup/scripts/LdapPlan.xml
```

For information about the **loadpricelist** utility, see "loadpricelist" in *BRM Setting Up Pipeline Pricing*.

Configuring the Channel Framework

This section includes the following channel framework configuration tasks:

- Configuring the pin_channel_export Utility
- Configuring Channel Definitions
- Loading Channel Definitions into the BRM Database
- Saving Channel Definitions to a File

Configuring the pin_channel_export Utility

The "pin_channel_export" utility collects channel events accumulated in the BRM database since the last run of this utility. After collecting all necessary information and translating the BRM object model to the directory server object model, the utility pushes the data to the LDAP directory servers.

This utility runs as a process under Linux. Once you start it, it runs continuously in the background until you end the process or until the BRM connection goes down. To keep the **pin_channel_export** utility running even when BRM goes down, configure the **mta_retry_srch** entry in the utility's **pin.conf** file.

You can edit the **pin_channel_export** configuration file in *BRM_home***/apps/exportapps** to configure the following options:

- To specify whether to delete channel events that have been pushed, set the **delete_channel_entry** entry. The default is **1**, which deletes the objects. A value of **0** keeps them.
- To specify the interval time running the utility, set the **sleep_interval** entry. By default, the **pin_channel_export** utility publishes data every 60 seconds.
- To specify the number of worker threads spawned to perform the specified work, set the **children** entry. The default is **5**.
- To specify the number of channel events processed by each worker thread in batch mode, set the **per_batch** entry. The default is **5000**.
- To specify the number of channel events returned by each search step in the BRM database, set the **per_step** entry. The default is **500**.
- To specify the number of channel events received from the BRM database in a block and cached in system memory for processing, set the **fetch_size** entry. The default is **1000**.

To specify a channel family, run the **pin_channel_export** utility with the **-f** parameter and specify the channel family ID. For example:

```
pin_channel_export -f 100
```

> **Note**
>
> - To publish channel events for different channel families, you need a separate **pin_channel_export** instance for each family ID. The channel events are published to their respective LDAP servers by the PCM_OP_REPL_POL_PUSH policy opcode based on the PIN_FLD_CONSUMER_OBJ value in the **/channel** object.
> - When channel events are not deleted from the **channel_event_table**, the table can grow rapidly and reduce performance.

For more information on publishing channels, see "How Channel Events are Published".

Configuring Channel Definitions

This procedure describes how to set the following attributes of a channel, which determine how channel events are published to the LDAP servers:

- The channel family
- The publishing order
- The publishing method: serially or in parallel

In addition to setting these attributes, you must set the general channel attribute values, including the channel name, consumer array information, and supplier array information. For more information on all channel attributes, see "About Defining Channels".

1. Open the sample **channel_config.xml** file in the *BRM_home*/**sys/data/config** directory with an XML editor or text editor.
2. Do the following for each channel definition in the file:
   - Set the **FldChannelId** value to assign a channel ID.

> **Note**
>
> Channel IDs must be less than **1000**.

- Set the **FldFamilyId** value to assign a channel family. For more information, see "About Channel Families".
- Set the **FldOrder** value to define a publishing order for a channel inside a family. For more information, see "About Channel Order".
- Set the **FldMultithread** value to set whether the channel is published serially or in parallel:

  0 = Serially

  1 = In parallel

  For more information, see "About Channel Publishing Mode".

3. Save the **channel_config.xml** file. You can save this configuration file with any name and in any location.
4. Load the channel definitions into the BRM database. See "Loading Channel Definitions into the BRM Database".

Loading Channel Definitions into the BRM Database

To load channel definitions, edit the sample **channel_config.xml** file, then run the "load_channel_config" utility to load the contents into the **/channel** object in the BRM database:

> **Note**
>
> To connect to the BRM database, the **load_channel_config** utility needs a configuration file in the directory from which you run the utility. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.

> **Caution**
>
> When you run the **load_channel_config** utility, it overwrites the existing channel definitions in the **/channel** object in the BRM database. If a channel definition exists but is not included in the **channel_config.xml** file, the database definition is not overwritten.

1. Define the channels for your database in the channel configuration XML file and save the file. For more information, see "About Defining Channels".
2. Use the following command to load the **channel_config** file:

   ```
   load_channel_config channel_config.xml
   ```

   where *channel_config* is the name of the channel configuration file.

   If the channel configuration XML file is not in your working directory, use the full path to the file. For example:

```
load_channel_config BRM_home/sys/data/config/channel_config.xml
```

3. Stop and restart the **pin_channel_export** utility.

To verify that the **channel_config.xml** file was loaded, you can display the **/channel** object by using the Object Browser, or use the **robj** command with the **testnap** utility. See "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

Saving Channel Definitions to a File

To save channel definitions stored in your BRM database to an XML file, run the **load_channel_config** utility with the **-r** parameter:

> **Note**
>
> To connect to the BRM database, the **load_channel_config** utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

```
load_channel_config -r channel_config.xml
```

To export the channel definitions to an XML file not in your working directory, use the full path for the file:

```
load_channel_config -r BRM_home/data/config/channel_config.xml
```

For more information on the **channel_config.xml** file, see "About Defining Channels".

When creating a new supplier for a channel, the PCM_OP_ACT_POL_EVENT_NOTIFY policy opcode formerly checked if the input event type was of a particular sub-type and, if so, set the PIN_FLD_SUPPLIER_OBJ field to the event POID and sent it to PCM_OP_CHANNEL_PUSH. Now, it sets the PIN_FLD_SUPPLIER_OBJ field to the **subtype** value of the supplier. For details, see step 4 in "Tracking Additional Changes to /account or /service Objects".

Your LDAP Manager configuration is now complete.

6

Customizing Your BRM LDAP Environment

Learn how to extend and modify the list of objects that Oracle Communications Billing and Revenue Management (BRM) LDAP Manager exports.

Topics in this document:

- Exporting Additional Data to the Directory Server
- Exporting New Service Types

For the default list of objects, see "Determining the /r_user Object Class Attributes".

Exporting Additional Data to the Directory Server

You can export more data to your LDAP directory server than the data exported by default by LDAP Manager. However, you must make sure that the LDAP Data Manager can *update* data in the directory server after the data is created.

Exporting Additional Fields from Objects

This section explains how to export additional fields from **/account**, **/service/ip**, or **/service/email** objects to the directory server. For the list of fields that are pushed to the directory server by default, see "Determining the /r_user Object Class Attributes".

Assume that you want to push some accounting information, specifically the accounting cycle day of the month (PIN_FLD_ACTG_DOM) field from the bill unit (**/billinfo** object). Whenever you modify this field in the **/billinfo** object (via PCM_OP_CUST_SET_BILLINFO), an **/event/customer/billinfo** event is generated.

You must perform the following steps in the PCM_OP_REPL_POL_PUSH opcode:

1. Read PIN_FLD_ACTG_CYCLE_DOM from the **/billinfo** object in addition to other fields that are already being read in the function.

   fm_repl_pol_get_account_flds() is located in the file **fm_repl_pol_translate.c** as follows:

   ```
   PIN_FLIST_FLD_SET(read_flistp, PIN_FLD_ACTG_CYCLE_DOM, NULL, ebufp);
   ```

2. Recompile the policy and restart the CM.
3. Define an attribute with integer syntax in the directory server. Make sure the **objectClass** of the entry that holds the account **/service** information in the directory server allows this attribute. For example, you could define the attribute **actgdom** by editing the **ruser** object class using the directory server manager and add **actgdom** to the list of allowed attributes.
4. Add the accounting day of the month field in the LDAP Data Manager mapping file (**ldap.idl**).
5. Add the following code to the **/r_user** *class definition* of the mapping file:

   ```
   INT PIN_FLD_ACTG_CYCLE_DOM {
        CREATE = Optional;
        MODIFY = Writeable;
   }
   ```

6. Add the following code to the **/r_user** *class implementation* section of the mapping file to define the new attribute:

```
INT PIN_FLD_ACTG_CYCLE_DOM {
     ATTRIBUTE = "actgdom";
}
```

Tracking Additional Changes to /account or /service Objects

The **/account** object information is pushed to the directory server when any of the following events are generated (as listed in the PIN_FLD_SUPPLIERS array of **/channel 101**):

- **/event/customer/nameinfo**
- **/event/customer/product_status**
- **/event/customer/status**
- **/event/customer/billinfo**

You can track additional changes to the **/account** and **/service** objects. This section provides an example of how to track an additional change in the **/account** object so that it can be pushed to the directory server.

Pushing the **/account** object information to the directory server when you update account information creates an **/event/customer/billinfo** event:

1. Add **/event/customer/billinfo** to the PIN_FLD_SUPPLIERS array in **/channel 100**. You can do this by running the PCM_OP_WRITE_FLDS opcode from **testnap**.

   For example, assume the following flist is stored in a file called **add.flist**:

```
0 PIN_FLD_POID          POID  [0] $DB_NO /channel 101
0 PIN_FLD_SUPPLIERS     ARRAY [5]
1 PIN_FLD_SUPPLIER_OBJ  POID  [0] $DB_NO
/event/customer/billinfo -1
```

2. Run **testnap** and use this **flist** as input to the write fields operation:

```
sh> testnap
testnap> r add.flist 1
testnap> wflds 1
testnap> q
```

3. To trigger BRM to call the policy opcode PCM_OP_ACT_POL_EVENT_NOTIFY (opcode number 301) whenever an **/event/customer/billinfo** event is generated, add the following line to your system's event notification list:

```
301    0 /event/customer/billinfo
```

   See "Implementing Event Notification" in *BRM Developer's Guide*.

4. Set up the account modification channel to be updated when an **/event/customer/billinfo** event is generated by adding the following to the **fm_act_pol_event_notify_ldap** function in the PCM_OP_ACT_POL_EVENT_NOTIFY policy opcode:

```
if (fm_utils_is_subtype(e_pdp,PIN_OBJ_TYPE_EVENT_CREATE_BILLINFO) == 1){
    type = PIN_OBJ_TYPE_EVENT_CREATE_BILLINFO;
    push_account_modify = 1;

if (push_account_modify || push_service_modify) {
    flistp = PIN_FLIST_CREATE(ebufp);
    PIN_FLIST_FLD_PUT(flistp, PIN_FLD_POID, (void *)ch_pdp, ebufp);
    db_id = PIN_POID_GET_DB(e_pdp);
    id = PIN_POID_GET_ID(e_pdp);
```

```
    e_pdp1 = PIN_POID_CREATE(db_id, type, id, ebufp);
    PIN_FLIST_FLD_PUT(flistp, PIN_FLD_SUPPLIER_OBJ,
(void*)e_pdp1, ebufp);
}
```

> **Note**
>
> This policy opcode is implemented in the *BRM_home***/source/sys/fm_act_pol/fm_act_pol_event_notify.c** file.

The PCM_OP_REPL_POL_PUSH policy opcode searches for all channels that have **/event/customer/billinfo** as one of its PIN_FLD_SUPPLIER_OBJ values and creates **/channel_event** objects for each of those channels.

5. Recompile the policy and restart the CM.

Exporting New Service Types

Assume that you want to export the information from a custom service object called **/service/web**. BRM updates the service creation and service modification channel objects (**/channel 102** and **/channel 103**) when the **/service/web** objects are created or modified. If you use the user scheme (where service and account object information are made available from a single entry in the directory server), you do the following:

1. Define the attributes that correspond to this service object in the LDAP directory server. For example, **web-login**, **web-password**, and **web-size-limit**.
2. Add the attributes that correspond to the **ruser** service object to the list of allowed attributes in the object class.
3. Define the mapping for these attributes in the LDAP Data Manager mapping file (**ldap.idl**). For example:
   - PIN_FLD_WEB_LOGIN maps to web-login
   - PIN_FLD_WEB_PASSWORD maps to web-password
   - PIN_FLD_WEB_LIMIT maps to web-size-limit
4. Enhance the PCM_OP_REPL_POL_PUSH policy opcode to handle the **/service/web** object. Since each creation and modification is tracked individually, any newly created service or service modification is automatically picked up.

> **Tip**
>
> Refer to the code in the policy file **fm_repl_pol_translate.c**, which implements the functionality that reads from the service object and pushes it to **dm_ldap**. You can manage the fields of the **/service/web** object similarly to how you handle the fields in **/service/email**.

7

Troubleshooting Your BRM LDAP Environment

Learn how to fix common problems in your Oracle Communications Billing and Revenue Management (BRM) LDAP environment.

Topics in this document:

Checking for Event Errors and Recovering from Failure

Periodically check for and correct event errors by doing the following:

1. Run **pin_channel_report** to print a list of objects whose changes could not be exported.
2. Check the **dm_ldap.pinlog** file for errors that prevent events from being pushed.
3. Correct each of the errors in the **dm_ldap.pin.log** file.

> **Note**
>
> Errors are unique to specific configurations.

4. After the errors are corrected, clear them by running:

   ```
   pin_channel_clear_error type poid
   ```

   where:

   - *type* is one of the following values:

     -a: Clears the status of all the **channel_event** objects with errors.

     -i: Clears the status of a particular **channel_event** object.

   - *poid* is the particular **channel_event** object containing an error.
5. To verify that the errors are cleared, run **pin_channel_report**. The utility returns an empty list if all errors have been cleared.
6. After the errors are cleared, push the changes to the directory server by running **pin_channel_export**.

If the **pin_channel_report** utility does not return any errors and the changes are still not exported, the problem must be elsewhere. Run **testnap** to find the problem.

Verifying Event Creation by Running testnap

You can verify that channel event objects were created by running the **testnap** utility:

1. Create and name a test file. For example, **stest**.

2. Enter the following flist into the test file:

```
r stest

1 nap(13860)> d 1
# number of field entries allocated 20, used 6
0 PIN_FLD_POID          POID [0] 0.0.0.1 /search/pin 0 0
0 PIN_FLD_FLAGS          INT [0] 256
0 PIN_FLD_TEMPLATE       STR [0] "select X from /channel_event where F1 = V1 or F2 =
V2 "
0 PIN_FLD_ARGS         ARRAY [1] allocated 20, used 1
1 PIN_FLD_STATUS        ENUM [0] 0
0 PIN_FLD_ARGS         ARRAY [2] allocated 20, used 1
1 PIN_FLD_STATUS        ENUM [0] 4
0 PIN_FLD_RESULTS      ARRAY [0] allocated 20, used 5
1 PIN_FLD_POID          POID [0] NULL poid pointer
1 PIN_FLD_OBJECT        POID [0] NULL poid pointer
1 PIN_FLD_CHANNEL_OBJ   POID [0] NULL poid pointer
1 PIN_FLD_SUPPLIER_OBJ  POID [0] NULL poid pointer
1 PIN_FLD_STATUS        ENUM [0] 0
```

3. Run **testnap** and read the test file into a buffer:

```
testnap
r file_name buffer_number
```

4. Search for the list entries:

```
search buffer_number
```

A list is returned, similar to this sample:

```
# number of field entries allocated 5, used 5
0 PIN_FLD_POID           POID [0] 0.0.0.1 /search/pin 0 0
0 PIN_FLD_RESULTS       ARRAY [0] allocated 4, used 4
1 PIN_FLD_POID           POID [0] 0.0.0.1 /channel_event 8645 0
1 PIN_FLD_CHANNEL_OBJ    POID [0] 0.0.0.1 /channel 100 0
1 PIN_FLD_SUPPLIER_OBJ   POID [0] 0.0.0.1 /account -1 0
1 PIN_FLD_STATUS         ENUM [0] 0
0 PIN_FLD_RESULTS       ARRAY [1] allocated 4, used 4
1 PIN_FLD_POID           POID [0] 0.0.0.1 /channel_event 9669 0
1 PIN_FLD_CHANNEL_OBJ    POID [0] 0.0.0.1 /channel 102 0
1 PIN_FLD_SUPPLIER_OBJ   POID [0] 0.0.0.1 /service -1 0
1 PIN_FLD_STATUS         ENUM [0] 0
0 PIN_FLD_RESULTS       ARRAY [2] allocated 4, used 4
1 PIN_FLD_POID           POID [0] 0.0.0.1 /channel_event 10693 0
1 PIN_FLD_CHANNEL_OBJ    POID [0] 0.0.0.1 /channel 102 0
1 PIN_FLD_SUPPLIER_OBJ   POID [0] 0.0.0.1 /service -1 0
1 PIN_FLD_STATUS         ENUM [0] 0
0 PIN_FLD_RESULTS       ARRAY [3] allocated 4, used 4
1 PIN_FLD_POID           POID [0] 0.0.0.1 /channel_event 11717 0
1 PIN_FLD_CHANNEL_OBJ    POID [0] 0.0.0.1 /channel 102 0
1 PIN_FLD_SUPPLIER_OBJ   POID [0] 0.0.0.1 /service -1 0
1 PIN_FLD_STATUS         ENUM [0] 0
```

About Status Values for Channel Events

The **pin_channel_export** utility initiates synchronization every 60 seconds by default. All **/channel_events** ready to be pushed (those with a status of **0** or **4**) are exported to the Connection Manager (CM). The **pin_channel_export** utility calls PCM_OP_REPL_POL_PUSH to export the data.

Table 7-1 shows the status values that appear in **channel_event_table**.

Table 7-1 Status Entries in channel_event_table

Numeric Value	Value	Description
0	PIN_STATUS_NOT_PUSHED	Needs to be pushed.
2	PIN_STATUS_PUSHED	Channel event has been pushed. By default, pushed **/channel_events** are deleted from the channel event table. To save a record of a pushed **/channel_event**, set **delete_channel_events** to **0** in the CM **pin.conf** file.
3	PIN_STATUS_ERROR	Requires user intervention to fix. These errors usually occur during configuration and are less likely to happen in a stable, operating environment.
4	PIN_STATUS_LINK_ERROR	The directory service is down. Does not require user intervention, since errors are resolved automatically.

Verifying the Mapping Between Object Classes and Entries

You can avoid object class violation problems in the BRM LDAP environment by making sure that BRM object class type definitions map to directory server schema entries correctly.

The following sections describe three common object class violations and provide guidance on how to resolve them.

Mismatches Between the Mapping File and Directory Server Entries

There are two common mismatches between the mapping file and directory server entries:

- The object class type definition in the mapping file does not match the corresponding directory server name.
- The attribute name in the mapping file does not match the corresponding directory server attribute name.

Entry Class Type Definition Contains a Typo

Problem

The name in the mapping file (*BRM_home***/sys/ldap.idl**) does not match the name of the entry in the directory server. For example, the directory server entry is named **ruesr** (a typo) while the mapping file has an object type of **ruser**.

Solution

Verify that the object class name matches the entry type name.

Case or Spelling Mismatches in Attribute Names

Problem

There is a mismatch between the attribute name in the directory server and the name supplied in the mapping file. Sometimes, the mismatch is due to a difference in case or spelling. The most common errors are with the attributes shown in Table 7-2.

Table 7-2 Common Errors with Attribute

Attribute Name in the Directory Server	Attribute Name in the Mapping File
userpassword	userPassword
maxmsgcount	maxmsgcnt

Solution

Verify that the attribute names match (check for case).

Object Classes or Attributes are Missing in the Directory Server

BRM cannot create directory server schema elements. Therefore, you must manually define the BRM data elements that you are interested in capturing with your own directory server tools. If you do not define these object classes and attributes in the directory server, you will encounter BRM object class errors. For example, to capture BRM account object data, you must create a directory entry called **r_user** with an attribute called **pin-poid-id**, which is relevant only to BRM.

Object Class Attributes Undefined in the Directory Server

Problem

None of the attributes belonging to the object class were created in the directory server, added to the object class, or both. For example, the **/r_user** object class must be modeled correctly. That is, it must contain all of the required BRM attributes to push data to the directory server.

Solution

Verify that the attributes were created and added to the object class.

For more information on the **/r_user** directory server attributes, see "Determining the /r_user Object Class Attributes".

Attribute Used in Mapping File is Undefined in the Directory Server

Problem

An attribute specified in the mapping file is not defined in the directory server and has not been added to the corresponding object class. For example, if the attribute *userPassword* is referenced in the mapping file but was never defined in the directory server, any attempts to add or modify this attribute fail. This type of error may be reported as "Undefined Attribute Type" by some directory servers.

Solution

Verify that an attribute with this name exists in the directory server. If it does not exist, create it and add it to the object class. Additionally, double-check that the spelling and case of the attribute match exactly.

Directory Server Object Class Created Without Required Attributes

Problem

The object class is created without including all the required attributes. A common issue arises when all attributes of the class are designated as required, but only a subset is provided during object creation.

LDAP Data Manager verifies that all attributes marked as required in the mapping file are included when creating an object. However, suppose an attribute is marked as optional in the directory server but is needed in the mapping file. In that case, it is your responsibility to ensure that the definitions of the object class in the directory server and the mapping file are aligned.

Solution

Verify that only the attributes that are genuinely required are marked as such for the class. Additionally, ensure that the object class definitions in both the directory server and the mapping file are consistent with each other.

No Such Object Errors

Table 7-3 shows the most common no such object errors.

Table 7-3 No Such Object Errors

Problem	Solution
You get a "no such object error".	The location specified in the mapping file does not exist in the directory server. For example, the location in the mapping file was **o=example**, and the directory server had a tree rooted at **o=example.com**. Verify that the tree with the correct root exists in the directory. You should also check for typos.
The *ou* specified in the PIN_FLD_DN or in the Location in the mapping file does not exist. For example, if the PIN_FLD_DN has **uid=user1, ou=ipservices, o=example.com**, but the directory server has **ou=serviceip**, any attempt to create or modify the entry will return this error.	Some directory servers return a constraint violation if this error is present along with a duplicate RDN_PIECE. For example, if you have the same DN and *user1* already exists in the directory server, you receive a constraint violation. After you fix that problem, you will see the "No such object error". Verify that the *ou* specified exists in the directory server. You should also check for typos.
Already exists.	The value of the RDN_PIECE already exists in the DS. Check for duplicates.
Not all attributes in the DS entry are returned when an entry is read or searched from the DS.	The mapping file (**ldap.idl**) does not specify a mapping for all attributes in the directory server entry. Specify a mapping for all the fields that you are interested in reading from the entry.

Table 7-3 (Cont.) No Such Object Errors

Problem	Solution
Not all DS entries are returned when the DS is searched using a search filter.	The scope of the search might limit the entries returned. For example, LDAP_SCOPE_ONELEVEL (1) limits the search scope to one level from the starting point. At the same time, LDAP_SCOPE_SUBTREE (2) searches the entire subtree. Specify the correct scope depending on what you are looking for.
When a complete DN is specified, you get the error "Given DN cannot reside in location Ö."	The DN's location suffix does not match the value specified in the *BRM_home*/**sys/ldap.idl** file. For example, if **ldap.idl** specified a location **o=example.com** and the complete DN specified **uid=user1**, **o=example**. Verify that the location suffix **o=example** and the value specified for location **o=example.com** match.

8

LDAP Manager Utilities

Learn about the syntax and parameters for the Oracle Communications Billing and Revenue Management (BRM) LDAP Manager utilities.

Topics in this document:

load_channel_config

Use this utility to load channel definitions into the **/channel** object in the Oracle Communications Billing and Revenue Management (BRM) database and to export existing channels to an XML file. Define the channel definitions in the *BRM_home***/sys/data/config/channel_config.xml** file or another file that uses the same format. The format of the XML file is specified in the **channel_config.xsd** schema file in the *BRM_home***/xsd** directory.

> **⚠ Caution**
>
> When you run the **load_channel_config** utility, it overwrites the existing channel definitions in the **/channel** object in the BRM database. If a channel definition exists in the BRM database, but isn't included in the **channel_config.xml** file, the database definition won't be overwritten.

When you run the utility, the **channel_config.xml** and **channel_config.xsd** files must be in the same directory. If you do not run the utility from the directory in which **channel_config.xml** is located, include the complete path to the file. For example:

```
load_channel_config BRM_home/sys/data/config/channel_config.xml
```

For information on the contents and format of the **channel_config.xml** file, see "About Defining Channels".

> **ⓘ Note**
>
> To connect to the BRM database, the **load_channel_config** utility needs a configuration file in the directory from which you run the utility. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.

Location

*BRM_home***/sys/data/config**

Syntax

```
load_channel_config [-d] [-v] [-t] [-r] [-h] channel_config.xml
```

Parameters

-d
Displays detailed information about status and error messages as the utility loads data.

-v
Writes error information for debugging purposes to the utility log file. By default, the file is located in the same directory as the utility and is called **default.pinlog**. You can specify a different name and location in the **Infranet.properties** file.

-t
Checks the validity of the **config_channel.xml** file without creating the **/channel** object.

-r
Retrieves the contents of the **/channel** object and writes it to the XML file specified.

-h
Displays the syntax and parameters for this utility.

***channel_config*.xml**
The XML file containing the channel definitions. A sample file (*BRM_home*/**sys/data/config/channel_config.xml**) is included, which you can edit to meet your specific business needs.
If you copy the **channel_config.xml** file to the same directory from which you run the **load_channel_config** utility, you don't have to specify either the path or the file name.
If you run the command in a different directory from where the utility's **pin.conf** file is located, you must include the entire path for the file.

pin_channel_export

LDAP Manager uses the **pin_channel_export** utility to collect channel events accumulated in the BRM database since the last run of this utility. After collecting all necessary information and translating the BRM object model to the directory server object model, the utility pushes the data to the LDAP directory servers.

> **Note**
>
> - To connect to the BRM database, the **pin_channel_export** utility needs a configuration file in the directory from which you run the utility. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.
> - To specify the time interval after which the utility runs and to specify whether to delete channel events from the Informant database after they are pushed to the LDAP directory server, edit the **pin_channel_export** utility's configuration file. See "Installing LDAP Manager".

For more information, see "About Channels and Data Propagation".

Location

BRM_home/**apps/exportapps**

Syntax

```
pin_channel_export [-f family_ID]
```

Parameters

-f *family_ID*
Publishes channels with the specified family ID to the LDAP directory server specified in the channel definition. There must be one **pin_channel_export** instance running per family ID. For example, the following command publishes channels with the family ID **100**:

```
pin_channel_export -f 100
```