Chapter 7

Maintaining the NDS Database

Overview

This chapter discusses how NDS maintains the integrity of its database. Specifically, it will discuss the following topics:

- Backing up and restoring NDS data
- Repairing time stamps
- Sending and receiving all updates
- Inspecting and analyzing a server's NDS database
- Getting a remote entry ID
- Inspecting an entry
- Reading references

Concepts to Know

To understand this chapter, you should be familiar with the following Novell Directory Services concepts:

- Entry
- Partition
- Replica
- Time stamp
- Directory Information Base (DIB)
- NDS reference

Preserving the Database

Like any other database system, Novell Directory Services provides features to maintain the database in its day-to-day operations.

As has been discussed already, NDS provides fault tolerance through replication and synchronization. This protects data against media or system failures. In addition, several problems could affect the database's integrity. For example, an operator might set a server's clock ahead, causing it to issue invalid time stamps. Values stamped with these values could not be overwritten or deleted. This chapter discusses the ways NDS deal with these and other problems.

Backing Up and Restoring NDS Data

Novell Directory Services is a database that coexists with a file system; however, NDS does not back up its data to the file system. Instead it uses its own backup routines: *Backup Entry*, and *Restore Entry*.

Backing Up an Entry

This operation allows a backup program to operate as a client and save a copy of an entry to be used as a backup. The backup program uses ordinary NDS access to walk the tree and locate the information to be backed up.

For each entry to be backed up, the backup program, acting as a client, sends a *Backup Entry* request to the server holding that entry. The information is returned in one or more backup chunks, which the backup program can then write to an external medium. Each chunk returned contains the server's NDS version to allow future software versions to determine the format used when the data was backup to. Information backed up can be restored using the *Restore Entry* operation described in this chapter. To back up an entry:

Table 7.1
Backing Up an Entry

Module	Action
Requesting module	1. Sends the server a *Backup Entry* request.
Server	2. Checks that the requesting module has managed rights for that entry 3. Returns a *Backup Entry* reply containing a completion code, an iteration handle, and the backup information.

Restoring an Entry

To restore NDS data, the server must use the *Restore Entry* operation to

retrieve the backup chunks from the backup programs. The server places the backed up entries in its database and assigns them new time stamps.

When a restored entry collides with an existing entry, the restored entry overrides any existing information.The server deletes the existing entry's attribute values, applies the restored values, and gives the entry a new creation time. The server then places on the new entry an obituary indicating the creation time of the overridden entry. The obituary allows servers to distinguish between the restored entry and the overridden entry.

The *Restore Entry* operation restores a deleted entry in one of three situations:

- It restores an entry from a backup when the original entry still exists on the server
- It restores an entry from a backup when the original entry does not exist on the server
- When a client is moving an entry, *DSARestoreEntry* adds the entry to its new location

Restoring an Entry from a Backup (Existing Entry)

When NDS restores an entry from backup, the original entry may still exist in the Directory tree. The request indicates that the entry exists in the tree by placing *[DS]* at the end of the version information. In this case, the old entry must be deleted before the backup can be restored.

When an NDS entry is restored, some of its attributes are *filtered*, meaning that they are not restored with the entry. For all object classes, the following attributes are filtered:

- Last Reference Time
- Backlink
- Obituary
- Reference
- Revision

In addition, the following table shows the attributes that are filtered for specific object classes.

Table 7.2
Filtered Attributes (By Object Class)

Object Class	Attributes Filtered
Partition	All attributes
Print Server	Host Device Network Address
Queue	Queue Directory
Server	Public Key Network Address

Table 7.2
Filtered Attributes (By Object Class)

Object Class	Attributes Filtered
User	Login Time Network Address

In addition, the following conditions may apply to restoring the entry:

- If the original entry is present on the server, it does not receive a new time stamp. If it exists but is not present, it receives a new time stamp.
- If the original entry is a Queue that is not present on the server, restoring the entry removes the Queue Directory attribute from the entry's list of attributes.
- If the entry is a Queue whose Host Server attribute is not changing, the operation leaves this attribute in the list. If the Host Server attribute value is changing, the server sends a packet to remove the queue directory.
- If the entry is a Print Server whose Host Device attribute value is not changing, the operation leaves this attribute in the list. If the Host Device is changing, the server sends a packet to remove the original directory.

To restore an entry from a backup when the original entry still exists in the tree:

Table 7.3
Restoring an Entry from a Backup (Existing Entry)

Module	Action
Requesting module	1. Sends a *Restore Entry* request to the server.
Server	2. Checks to see if the entry is being restored from a backup or from a move. 3. Checks that the entry exists on the server. 4. Checks that the client has Remove and Create rights to delete the old entry and add the new entry. 5. Checks to see that the entry is writable. 6. Checks to see if the entry is a partition root entry. 7. Checks to see if the entry is an alias. 8. Checks the entry's present base class. 9. Purges the naming values and any attribute values that are not valid for the new entry's object class. 10. Adds the new entry. 11. Returns a *Restore Entry* reply indicating success or failure.

Restoring an Entry from a Backup (Entry Not Existing)

The *Restore Entry* operation differs when the original entry does not exist in the tree. The requesting module must have Create rights to restore the entry on the server. In addition, if the original entry was a Queue, the

operation deletes the Queue Directory attribute from the entry's list of attributes.

To restore an entry from a backup when the original entry does not exist in the tree:

Table 7.4
Restoring an Entry from a Backup (Entry Not Existing)

Module	Action
Requesting module	1. Sends a *Restore Entry* request to the server.
Server	2. Checks to see if the entry is being restored from a backup or from a move. 3. Checks that the entry exists on the server. 4. Checks that the client has Create rights to add the new entry. 5. Checks to see that the entry is writable. 6. Checks to see if the entry is a partition root entry. 7. Checks to see if the entry is an alias. 8. Checks the entry's present base class. 9. Purges the naming values and any attribute values that are not valid for the new entry's object class. 10. Adds the new entry. 11. Returns a *Restore Entry* reply indicating success or failure.

Restoring an Entry from a Move

The Move operation places the entry in a new partition. Consequently, the new entry does not replace an existing entry. To restore an entry that is moving:

Table 7.5
Restoring a Moved Entry

Module	Action
Requesting module	1. Sends a *Restore Entry* request to the server.
Server	2. Verifies that it holds the master replica of the entry. 3. Checks that the client has Create rights to add the entry. 4. Checks to see if the entry is being restored from a backup or a move. 5. Checks the entry's object class. 6. Checks to see if the entry is an alias and obtains the aliased entry's name. 7. Checks to see if the entry is a print server or print queue. If so, creates a file directory associated with the entry. 8. Creates the entry record. 9. Adds the subordinate entry to its parent. 10. Adds the object class and naming values. 11. Adds attribute values. 12. Returns a *Restore Entry* reply.

The *Restore Entry* request is the same as the request used to restore from a

backup. The *DS_RESTORE_MOVING* request flag is set to 0, indicating that the entry is being restored from a *Move Entry* operation.

Repairing Time Stamps

If an NDS operator set a server's clock ahead of the other servers in the network, time stamps issued by that server would be invalid because they and new time stamps would always be in the future. Thus, in a collision, the future time stamp would always override the valid time stamp. The *Repair Time Stamps* operation, although drastic, remedies such a situation.

To repair time stamps:

Table 7.6
Repairing Time Stamps

Module	Action
Requesting module	1. Sends a *Repair Time Stamps* request to the server holding the master replica of the affected partition.
Server holding master replica	2. Checks all of the partition's time stamps in it local database. 3. Assigns new time stamps to any entries having time stamps that are ahead of the server's current clock time. 4. Declares a new epoch by setting a new Epoch time stamp on the partition root object. 5. Returns a *Repair Time Stamps* reply to the client. 6. Sends a *Start Update Replica* request to other servers holding replicas of the affected partition.
Server holding a nonmaster replica of the partition	7. Checks its Epoch against the Epoch it receives in the *Start Update Replica* request. 8. Returns an *Old Epoch* completion code in the reply.
Server holding master replica	9. Sets the partition's operation in progress to "Repairing Timestamps." This locks the partition until the operation is complete. 10. Sends *Update Replica* requests to all servers in the replica ring. In each request, the server sets the *New Epoch* request flag. It continues sending *Update Replica* requests until it has transferred all the partition's data.
Server holding a nonmaster replica	11. Purges its replica of the partition and replaces it with the data it receives in the *Update Replica* requests.
Server holding the master replica	12. Unlocks the partition.

Sending All Updates

An error might prevent data from being propagated to all replicas of a partition. To send all information to all replicas, a client sends a *Partition Function* request with the function code set to *Send All Updates.* This causes the server to send all the partition data to all replicas.

To send all updates:

Table 7.7
Sending All Updates

Module	Action
Requesting module	1. Sends a *Partition Function* request to the server. In the request, the client sets the function code to *Send All Updates.*
Server holding affected replica	2. Synchronizes with all other replicas, sending all partition information regardless of time stamps.
Server receiving partition information	3. Decides which information to keep, depending on time stamps.

Receiving All Updates

This operation replaces a replica on a specific server if that replica's time stamps are incorrect. For example, an operation to repair the local database may not have finished correctly, leaving the replica's state unknown. To remedy such a problem, a client sends the master replica a *Partition Function* request with the function code set to *Receive All Updates.*

To receive all updates:

Table 7.8
Receiving All Updates

Module	Action
Requesting module	1. Sends a *Partition Function* request to the server. In the request, the client sets the function code to *Receive All Updates.*
Server holding affected replica	2. Tags the replica as *New*. 3. Synchronizes with the specified replica, sending all partition information regardless of time stamps.
Server receiving partition information	4. Purges all replica information and replaces it with the data it receives from the master replica.

Inspecting and Analyzing a Server's NDS Database

NDS provides three operations that allow a client to analyze the database information held on a given server:

- *Inspect Entry.* This operation allows a client to obtain information about an entry and any errors that entry has experienced.
- *Read References.* This operation reports information about references a server holds.

The following sections describe each of these operations.

Inspecting an Entry

This operation is used to verify an entry's existence and list any errors associated with that entry. To inspect an entry:

Table 7.9
Inspecting an Entry

Module	Action
Requesting module	1. Sends a *Inspect Entry* request to the server. This request specifies the entry to be inspected.
Server	2. Checks that the client has *Browse* [Entry] rights to the entry. 3. Verifies the entry's existence and get a list of errors. 4. Returns a *Inspect Entry* reply to the server.

Reading References

A reference is a hidden attribute; clients cannot read a reference using normal *Read* operation. A reference uses the Distinguished Name syntax and is a per-replica attribute stored only on the replica where it originated.

Unlike a back link, which keeps track of relationships between servers, references track relationships between entries on the same server. The *Read References* operation reports these relationships, as well as the relationships between bindery objects in bindery services.

To list information about the references a server holds:

Table 7.10
Reading References

Module	Action
Requesting module	1. Sends a *Read Reference* request to the server.
Server	2. Returns a *Read References* reply to the server.

File Directories Related to NDS Entries

Each NDS entry whose object class is Queue has a Queue Directory attribute value, which identifies a file system directory where the queue's files are stored. The directory and queue to be on different systems.

NDS uses two operations to maintain the correspondence between the Queue Directory attribute value and its file system directory:

- *Create Entry Dir*. The server holding the NDS entry sends this request to the server where the directory is to be created.
- *Remove Entry Dir*. A server changing or deleting the NDS entry sends this request to the server holding the directory, which deletes the directory.

The following sections describe these two operations.

Creating an Entry Directory

Each NDS entry of object class *Queue* has a *Queue Directory* attribute. This attribute identifies a file system directory where the queue's files are stored.

To create a directory associated with a queue object, a server holding a specific entry acts as the requesting module. For example, when a server creates a Queue entry, it requests that another server create a file system directory associated with that entry. The requesting server sends a *Create Entry Dir* request to the server holding the volume that will hold the entry directory. To create an entry directory:

Table 7.11
Creating an Entry Directory

Module	Action
Requesting module	1. Sends a *Create Entry Dir* request to the server holding the volume that will hold the new directory.
Server	2. Creates the directory and generates a random name for the directory. It begins at x10000000 and increments by 1 until it succeeds. It appends this name to the name given in the Directory Name field. 3. Returns a *Create Entry Dir* reply containing the name (file path) of the directory.

Removing an Entry Directory

When a print queue or print server entry is deleted, its associated file system directory must also be deleted. To remove an entry directory:

Table 7.12
Removing an Entry Directory

Module	Action
Requesting module	1. Sends a *Remove Entry Dir* request to the server.
Server	2. Resolves the name of the entry. 3. Reads the root entry of the entry's partition. 4. Read the entry's base class. 5. Stores the entry's base class, partition name, entry name, client name, and directory name. 6. Schedules the background task (checks for *Dir Removal Event*). 7. Returns a *Remove Entry Dir* reply indicating success or failure.

Table 7.12
Removing an Entry Directory

Module	Action
Dir Removal *Event* Process	8. Processes list saved in step 5. 9. Checks that the requesting module is in the partition root's replica ring. 10. If the entry is a print server, removes the directory based on the Entry ID. If the entry is a queue, removes the directory based on the Directory name.

Checking a User's Security Equivalence Vector

Although a User entry may hold a Security Equals attribute, its value may not accurately reflect the entry's security equivalence. *Check SEV* allows a client to check the entry's current security equivalence vector.

To check an entry's security equivalence vector:

Table 7.13
Checking an Entry's Security Equivalence Vector

Module	Action
Requesting module	1. Sends a *Check SEV* request to the server holding the User entry.
Server	2. Resolves the entry's name. 3. Checks the entry against the entry indicated in the "Possible Equivalence" field to determine whether the two entries are actually equivalent. 4. Returns a *Check SEV* reply indicating success or failure.

Checking a Console Operator

A user can have operator privileges for a server. This privilege is reflected in the server entry's and the user's security equivalences. Because checking these vectors can be time consuming, the server stores a list of operators separately. If an entry with operator privileges changes, the server making the change sends a *Check Console Operator* request to the server whose operator list is affected. The request identifies the user, and the receiving server checks its local database to determine if it should add or delete the user from the operator list.

To check a console operator:

Table 7.14
Checking a Console Operator

Module	Action
Requesting module	1. Sends a *Check Console Operator* request to the server holding the affected Operator list.
Server	2. Adds or deletes the entry to or from its Operator list. 3. Returns a *Check Console Operator* reply indicating success or failure.