UNDERSTANDING THE NIS TO LDAP SERVICE (N2L) ARCHITECTURE

Michael Haines, Sun Microsystems, Inc.
Baban Kenkre, Sun Microsystems, Inc.

Sun BluePrints™ OnLine — March 2006

Part No 819-4326-10
Revision 1.0, 3/13/06
Edition: March 2006

Please
Recycle

Table of Contents

Understanding the NIS to LDAP Service (N2L) Architecture

Today, organizations are looking for ways to improve the user experience. Their long term strategy is to provide a single sign-on solution, giving users the freedom to move from one corporate application to another within a single session—without being prompted for a userid and password, or a digital certificates. At the same time, organizations want to limit the number of repository access protocols in use. As a result, many organizations are likely to implement solutions based on the Lightweight Directory Access Protocol (LDAP), a simplified version of the DAP protocol used for accessing a directory service. Many organizations deploy the Lightweight Directory Access Protocol (LDAPv3), enabling them to benefit from well-defined mechanisms that extend the core protocol to provide data in a standardized way. The Solaris™ Operating System (Solaris OS) works with standard and custom schemas to utilize LDAP as a direct naming service repository, and aids early adoption through features such as NIS to LDAP (N2L) functionality.

This Sun BluePrints™ article discusses Network Information Service (NIS) to LDAP transition service (N2L service) support for NIS clients based on naming information stored in the Sun Java™ System Directory Server 5.2 software. The approach taken by the N2L service enables a complete transition from the NIS naming service to the LDAP naming service. It includes detailed installation, configuration, and operational information needed to create a supportable instance of the NIS/LDAP Transition Gateway product offering. While the NIS to LDAP transition product is designed to work with any RFC2307bis-compliant directory (LDAP) server, Sun only supports the N2L Service in conjunction with the Sun Java System Directory (LDAP) Server 5.1 and 5.2 software. It is important to note this article only covers the Sun Java System Directory (LDAP) Server 5.2 software.

Note – Once the transition from NIS to LDAP is complete, LDAP should be considered the master repository.

Background

As networks have grown over the years, so has the complexity and amount of information required to run them. Host names, network system addresses, automounter maps that orchestrate dynamic resource sharing, and the configuration data that boot servers need to boot clients, are all critical pieces of information that keep the network running. Efficiently managing this required information is an important component in supporting a productive network.

With an increase in network communication comes a corresponding increase in the amount of information required by each machine on a network, including:

- All Internet networks require each host to know the address and hostname of other machines on the network.
- In order to maintain a common view of home directories, the automounter requires consistent `auto_master` and `auto_home` maps and common password information.
- Boot servers must have `bootparams` information available to identify their clients.

Even on a small network of two or three machines, maintaining all this information in data files stored on each machine is cumbersome. It is unmanageable on larger networks. To help this effort, the Solaris OS provides the LDAP, NIS, and NIS+ naming services. These naming services make information stored on designated servers available to a set of machines on the network.

What is a naming service? Conceptually, a naming service centralizes the shared information in a network. One or more name servers manage information previously maintained on each individual host, such as hostnames and IP addresses, user names, passwords, and automount maps, and more. In its simplest form, a name service translates names to numbers. This client-server software translates requests for named items, like hostnames, into their associated network numeric equivalent.

Since its inception, the Solaris OS has provided a core platform for naming services. Indeed, Sun's NIS (formerly known as Yellow Pages) was first available in 1985. Today, naming services continue to be an important component of the Solaris OS, and the following four name service solutions address specific needs or architectures:

- *Domain Name Service (DNS)*, providing the translation of hostnames to their associated IP addresses within a TCP/IP network. In addition, DNS also specifies the domain to which each participating machine belongs.
- *Network Information Service (NIS)*, providing a centralized lookup for resources, such as user accounts, host names and addresses, services, automount maps, and other key files that are normally needed on each host. NIS is easy to set up and maintain, and is supported by many vendors.
- *Network Information Service Plus (NIS+)*, providing a central lookup location for resources. NIS+ provides several signficant enhancements to NIS, including support for features of a hierarchical naming structure, distributed administration, built-in security authentication, and cross-domain lookups.
- *LDAP Directory Service*, extending the naming services with a directory service. While a naming service allows an object to be looked up by name, a directory service allows these objects to have attributes. As a result, a directory service enables objects to be looked up by name and their attributes obtained, or searched for by attributes.

Common to All Naming Services

An Introduction to the Name Service Switch

The Solaris OS can use a variety of information sources to obtain network information, a process known as the *Name Service Switch*. Ultimately, the order in which the the name Service Switch accesses these different sources of information is configured in the */etc/nsswitch.conf* file. Organized as a database list, the *nsswitch.conf* file contains a number of entries, each followed by an ordered list of name services from which relevant data can be retrieved.

Each major name service provided in the Solaris OS includes a default */etc/nsswitch* file which is copied to *nsswitch.conf* during configuration (Table 1). For example, selecting NIS as the name service during client installation causes the */etc/nsswitch.nis* configuration file to be copied to */etc/nsswitch.conf*. The default */etc/nsswitch.nis* file in the Solaris 10 OS contains the following database list.

```
$ grep '^[a-z]*:' /etc/nsswitch.nis
passwd: þ þ files nis
group:þ þ þ files nis
hosts:þ þ þ nis [NOTFOUND=return] files
ipnodes:þ þ nis [NOTFOUND=return] files
networks: þ nis [NOTFOUND=return] files
protocols:þ nis [NOTFOUND=return] files
rpc:þ þ þ þ nis [NOTFOUND=return] files
ethers: þ þ nis [NOTFOUND=return] files
netmasks: þ nis [NOTFOUND=return] files
bootparams: nis [NOTFOUND=return] files
publickey:þ nis [NOTFOUND=return] files
netgroup: þ nis
automount:þ files nis
aliases:þ þ files nis
services: þ files nis
printers: þ user files nis
project:þ þ files nis
```

Consider the `passwd` database. The `passwd` entry states that user account information traditionally stored in */etc/passwd*, `passwd(4)`, should be looked for first in local files. If user information does not exist in the local files, a password request is sent to the NIS server. Note that `nis` follows `files`þwithout alternative criteria.

The default criteria can be changed by specifying alternative actions in response to the service status. For example, consider the `hosts` entry above.þ The `hosts` information is first looked up on the NIS server. If the relevant information is not found on the NIS server no further sources are queried, as specified by the criteria `[NOTFOUND=return]`. However, if the NIS service is unavailable the local /etc/inet/hosts file is used, as the default criteria for `UNAVAIL` is `continue`.þ A complete list of status codes can be found in Table 2, and possible actions in Table 3.

Note – As of the Solaris 8 OS, searching for `host` information causes the `ipnodes` database to be searched first.

Note – It is important to understand that major name services cannot be configured on client systems by simply copying over the associated *nsswitch.conf* file. Each name service must be configured using the specific configuration command. Refer to `ypinit(1M)`, `nisclient(1M)`, and `ldapclient(1M)` for more information.

Table 1. Name service template files

Name Service	Name Service Template
Local files	*/etc/nsswitch.files*
DNS	*/etc/nsswitch.dns*
NIS	*/etc/nsswitch.nis*
NIS+	*/etc/nsswitch.nisplus*
LDAP	*/etc/nsswitch.ldap*

Note – Name service template files are simply that—templates—and may need modification based on local site policies before being deployed as the *nsswitch.conf* file. For example, it is common to modify the *msswitch.ldap* file to include DNS on the `hosts` and `ipnodes` lines in the file.

Name Service Switch Status and ActionValues

Using the default */etc/nsswitch.nis* file as an example, the name service switch now presents some action values for several entries. The naming service searches for resolution from the specified source, and returns a status code to the user requesting NIS information. Table 2 identifies the two action codes possible for each status code.

Table 2. Status code action

Action	Description
continue	Try the next source
return	Stop looking for the entry

Two actions are possible for each status code: continue and return. When the action is not explicitly specified, the default action is to continue the search using the next specified information source, as specified in Table 3.

Table 3. Status default action

Status	Description
`SUCCESS`	Return. The requested entry was found in the specified source.
`UNAVAIL`	Continue. The source is not configured on the system and cannot be used. The NIS, NIS+, or LDAP processes could not be found or contacted.
`NOTFOUND`	Continue. The source responded with no such entry. The table, map, or file was accessed but did not contain the needed information.
`TRYAGAIN`	Continue. The source is busy, and might respond if tried again. The name service is running and was contacted, but could not service the request.
`FOREVER`	Loop forever (when used with TRYAGAIN)
`n`	Retry the current source n times, where 0 < *n* < MAX_INT. Once *n* actions are exhausted, action continues to the next source

Consider the following examples:

- `ipnodes`

  In this example, the */etc/inet/ipnodes* file is searched forþthe first entry that matches the requested host name. If noþmatches are found, an appropriate error is returned and other information sources are not searched.

```
ipnodes: files
```

- `passwd`

  In this example, the appropriate files in the */etc* directoryþare searched for the corresponding password entry. If theþentry is not found, the NIS maps are searched for the entry.þIf no entry is found in the NIS maps, an appropriate errorþis returned and other information sources are not searched.

```
passwd: files nis
```

- `hosts`

  In this example, the NIS maps are searched for the entry. If the source (NIS) is not running, the system returns a status of `UNAVAIL` and continues to search the */etc/inet/hosts* file. If the entry returns the status `NOTFOUND`, an appropriate error is returned and the search is terminated without searching the */etc/inet/hosts* file.

```
hosts: nis [NOTFOUND=return] files
```

Traditional NIS

NIS is a distributed name service, a mechanism for identifying and locating network objects and resources. By providing centralized control over network information, the NIS naming service eases network administration. NIS stores information about workstation names and addresses, users, the network, and network services. This collection of network information is referred to as the *NIS namespace.*

NIS uses domains to arrange the machines, users, and networks in its namespace, as well as define the users that can access host names, user information, and other administrative data in its namespace. However, it does not use a domain hierarchy—the NIS namespace is flat. By running the NIS service, organizations can distribute administrative database maps among a variety of servers (master and slaves). These databases can also be updated automatically and reliably from a centralized location to ensure all clients share the same name service information in a consistent manner throughout the network.

The NIS namespace information is stored in NIS maps. Designed to replace UNIX */etc* files, as well as other configuration files, NIS maps store more than names and addresses. As a result, the NIS namespace has a large set of maps. NIS domain maps typically include the following files.

- *auto_home*, contains the automounter configuration file
- *auto_master*, specifies all the automounter maps in a domain
- *bootparams*, defines the boot parameters for systems booting over the network
- *ethers*, specifies Ethernet address mapping operations
- *group*, defines group permissions
- *hosts*, contains information for all known hosts on the network
- *netgroup*, defines network groups
- *netmasks*, contains the network masks used to implement IP subnetting
- *networks*, provides information on known networks
- *protocols*, provides information on protocols used within the network
- *passwd*, contains password information for users
- *rpc*, provides library routines for remote procedure calls
- *services*, provides information for services available in the network, including the name, port number, protocol used, and more
- *aliases*, specifies mechanisms for redirecting user mail
- *timezone*, specifies the timezone for a system
- *ipnodes*, maps hosts to their IP addresses

Note – NIS administrators often create custom maps for their specific network environment needs. This is also an important part of migrating from the NIS to LDAP service.

Figure 1 depicts a traditional NIS architecture.

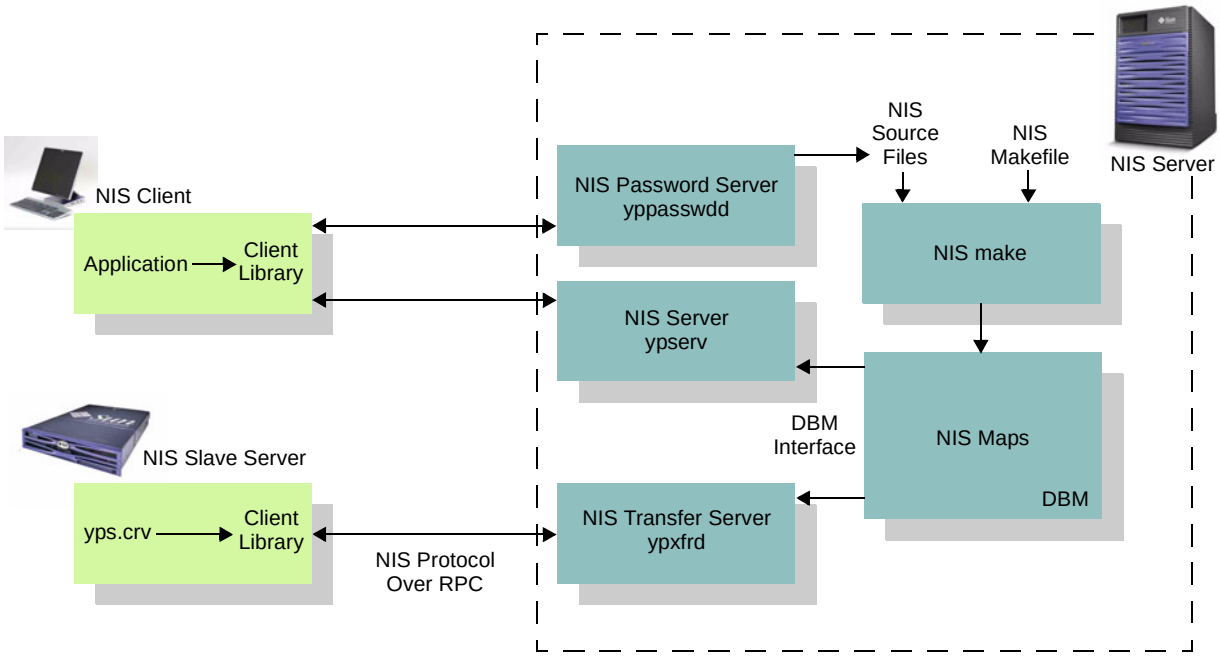

Figure 1. A traditional NIS architecture

The NIS Processes

Various processes are used in the running of a NIS domain. The main processes include:

- `ypserv(1M)`. Running on both master and slave servers, the `ypserv` process answers `ypbind` requests from clients, and responds to client information requests.
- `ypbind`. Running on master and slave servers and client systems, the `ypbind` process makes the initial client-to-server binding request, which it then stores in the */var/yp/binding/domainname* file. The `ypbind` process is capable of binding to another server if the connection is lost with the initial server.

Other processes used in the NIS enviroment include:

- `rpc.yppasswdd`. Running on a master server, the `rpc.yppasswdd` process updates the */etc/passwd*, */etc/shadow*, and *passwd.adjunct* files on the master server, updates the NIS password map, and pushes the NIS password map to all slave servers. This enables users to change passwords using the `yppasswd(1)` or `passwd(1)` commands.
- `ypxfrd`. Running only on a master server, the `ypxfrd` process transfers NIS maps to NIS slave servers at high speed.

Note – The use of `yppasswd(1)` is discouraged, as it is now only a wrapper for the `passwd(1)` command, which should be used instead.

The NIS Maps

The NIS maps are located in the */var/yp/domainname* directory, where *domainname* is the name of the NIS domain. The NIS maps consist of two files (*.pag* and *.dir*) for each map in the directory. The syntax for NIS map names is *map.key.pag* or *map.key.dir*, where:

- *map* is the base name of the map (`hosts`, `passwd`, and so on)
- *key* is the map's sort key (`byname`, `byaddr`, and so on)
- *pag* is the map's data
- *dir* is an index to the *.pag* file if it is large

NIS maps are based on the `ndbm(3c)` database. As a result, the following restrictions in the NDBM database apply to NIS environments:

- The length of a single entry in a `ndbm(3c)` database is limited to 1,024 bytes.
- The `ndbm(3c)` databases only support one key per map that can be used for search operations. Since several naming services databases must be able to be searched with multiple different keys, `ndbm` (and therefore NIS) requires multiple copies of some maps to be stored, including some that contain the same data.

NIS to LDAP Introduction

As of the Solaris 9 OS (Update 5),a NIS/LDAP migration tool is integrated into the Solaris OS. It provides a migration path from an existing NIS name service to an LDAP name service by enabling the `ypserv(1M)` daemon to use a directory (LDAP) server as a repository for NIS data. Toward this end, the NIS database routines, located in the `libnisdb` library, have been modified to store and retrieve NIS object data to and from LDAP. In order to alleviate the overhead of LDAP data retrieval and storage, the `ypserv` daemon is multi-threaded. The `ypserv(1M)` daemon uses the RPC auto-MT mode.

The default NIS/LDAP mappings use the RFC2307bis schemas and are compatible with the Solaris Secured LDAP Client, an LDAP client that provides a full name service on the Solaris OS that is similar to NIS(YP), NIS+, and DNS. The Solaris Secured LDAP Client includes a name service switch module, PAM updates and modules, command modifications for those that use name services directly, such as sendmail and automount, and the creation of a simplified private name service API for accessing LDAP. In addition, the Solaris Secured LDAP Client supports TLSv1/SSL to enable the establishment of a secure transport between clients and servers.

Note – RFC2307bis describes an approach for storing NIS data in an LDAP server. As of this writing, its IETF status is *Experimental*.

Table 4 lists the man pages related to NIS to LDAP migration. In addition, the NIS to LDAP Migration Service is described in the *System Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP)*, in the *Transitioning from NIS to LDAP* chapter.

Table 4. Man pages related to NIS to LDAP migration

New Man Pages	Modified Man Pages
`inityp2l(1M)`	`rpc.ypupdated(1M)`
`NISLDAPmapping(4)`	`ypfiles(4)`
`ypmap2src(1M)`	`ypmake(1M)`
`ypserv(4)`	`rpc.yppasswdd(1M)`
	`ypserv(1M)`

NIS to LDAP Service Components

The NIS to LDAP service is comprised of several components (Figure 2).

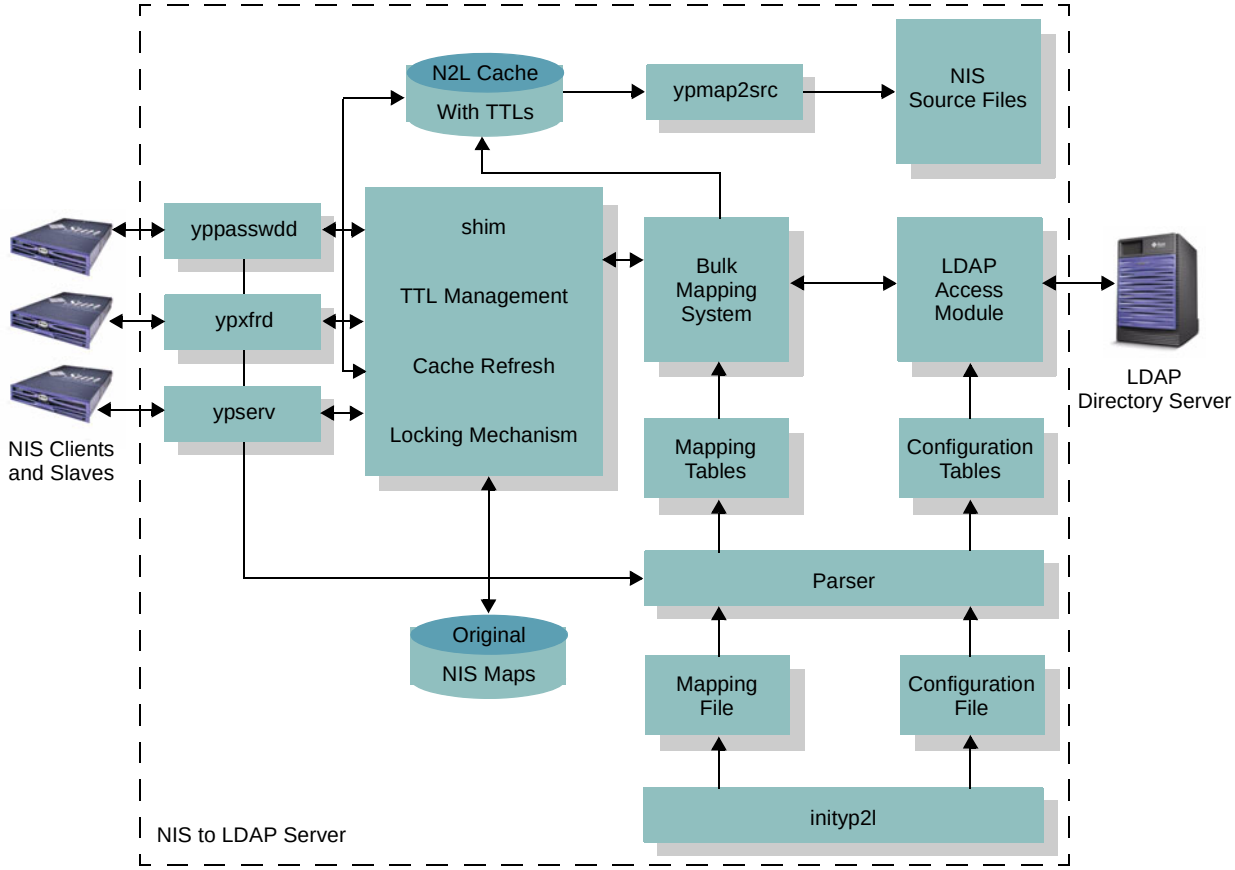

Figure 2. The NIS to LDAP service architecture

N2L Configuration Files

The NIS to LDAP service uses two configuration files:

- `NISLDAPmapping(4)`

  Also referred to as the mapping file, the `NISLDAPmapping(4)` is the primary configuration file. It specifies mappings between NIS map entries and equivalent directory information tree (DIT) entries. The presence of this file on a NIS master server transforms the server into a NIS to LDAP gateway. Similarly, renaming or deleting the file reverts the server to traditional NIS mode.

Note – Sites with custom NIS maps or those requiring different mappings between NIS and LDAP entries must customize the `NISLDAPmapping(4)` file before use.

- ypserv(4)

  The `ypserv(4)` file specifies configuration information, such as the LDAP server list, timeouts, limits, bind DN, authentication method, error actions, and more. The configuration information can either be derived from the LDAP server or can be specified in this file.

How and where are these files created? The Solaris OS includes the `inityp2l(1M)` script. This script assists users with the creation of the initial configuration files through a dialogue. Once created, users can customize these files based on specific needs. The NIS server-side daemons must be restarted to detect any changes to these files.

Note – The `ypserv(4)` file contains sensitive information, such as the LDAP BINDDN password, in clear text. As a result, users should ensure this file has strict file access permissions.

N2L Parser

The parser is implemented in the `libnisdb.so` library. The NIS server-side daemons invoke the parser if they detect the mapping file on startup. The parser parses the configuration files, initializes data structures, and creates in-memory configuration objects and a hash table of mapping objects which are later used by the mapping and LDAP access modules (Figure 3).

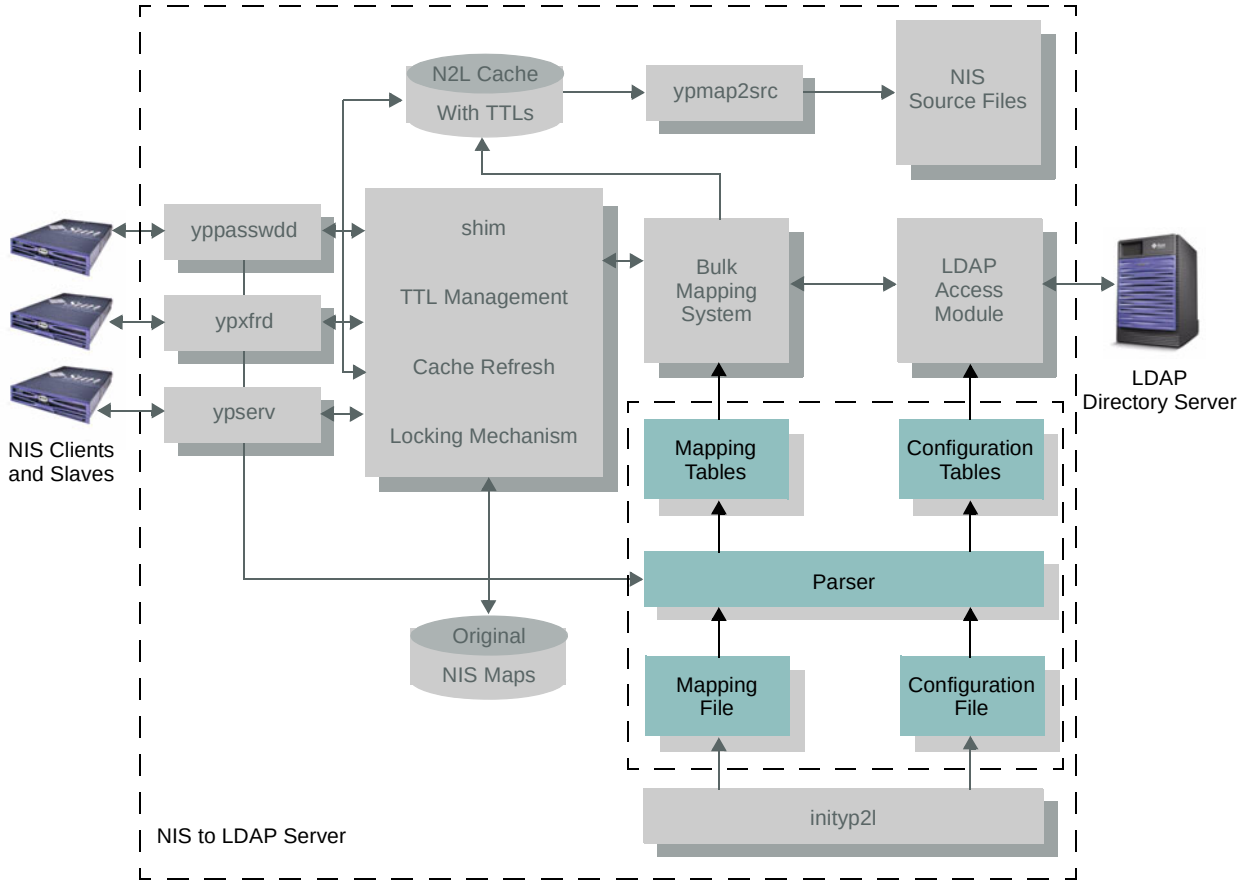

Figure 3. The parser scans configuration files, and creates and initializes in-memory data structures

Each mapping object corresponds to a NIS map per domain, as specified in the mapping file. The parser assigns sequence numbers to these mapping objects based on the order of the corresponding `nisLDAPObjectDN` attributes in the `NISLDAPmapping(4)` file. This ensures the mappings that load primary information in LDAP entries, such as `STRUCTURAL` object classes, are uploaded before those that add auxiliary information.

N2L CACHE

Once the initial upload of data from the original NIS maps to the directory information tree (DIT) is complete, the LDAP server is the authoritative source for data. From this point on, the original NIS maps are no longer used and are left untouched by the N2L gateway. However, the N2L gateway creates and maintains a local cache of the LDAP data in the form of `ndbm(3C)` maps (Figure 4). These N2L maps are named with an `LDAP_` prefix and are created in the */var/yp/domainname/* directory.

Associated with each N2L map is a Time-to-Live (TTL) map, also in `ndbm(3C)` format. Each TTL map contains two types of TTLs: per-entry TTL, and per-map TTL. See the `nisLDAPentryTtl` section of the `NISLDAPMapping(4)` man page for more information. The N2L and TTL maps can be accessed using the `ndbm(3C)` library calls and administration commands, such as `makedbm(1M)`. Note that a N2L map or cache entry is refreshed only if a request is received by the N2L gateway to access the corresponding map or entry and the associated TTL has expired.

```
$ cd /var/yp/domainname/
$ ls LDAP_passwd.byname.???
LDAP_passwd.byname.dir LDAP_passwd.byname.pag

$ ls LDAP_passwd.byname_TTL.???
LDAP_passwd.byname_TTL.dir LDAP_passwd.byname_TTL.pag
```

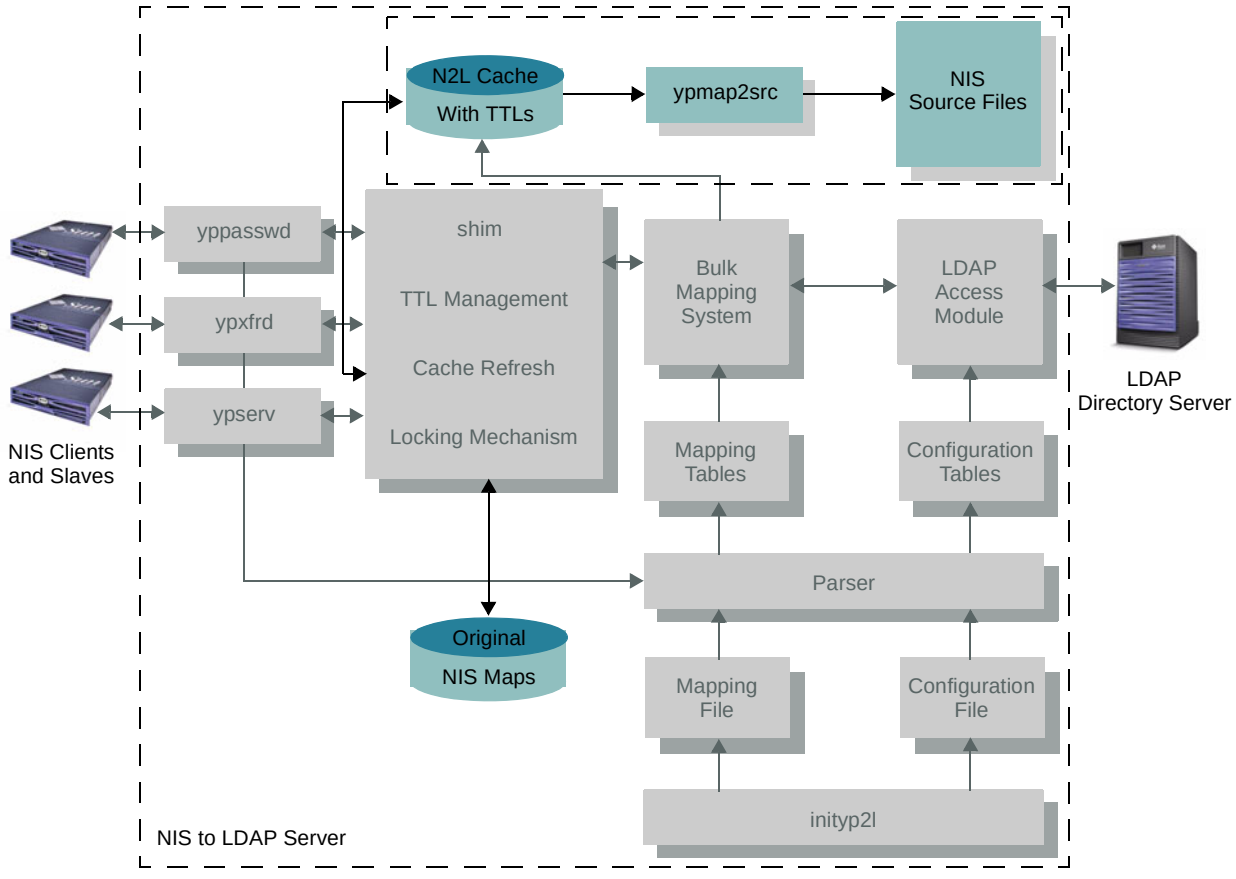

Figure 4. The cache stores N2L map data

N2L Front-End Mechanism

The front-end of the N2L service consists of modified NIS server-side daemons, including the `ypserv(1M)`, `ypxfrd`, and `yppasswdd` daemons. On startup, these server-side daemons check for the existence of the mapping file. If the mapping file is present, the daemons start in N2L mode. Otherwise, they start in traditional mode. The RPC front end, which communicates with NIS clients and slave servers, remains unchanged.

In N2L mode, `ypserv(1M)` supports additional command line options. When invoked with the `-i` option, `ypserv(1M)` uploads data from the original NIS maps to the DIT. The `-I` option is identical to `the -i` option, with the exception that missing domain and container objects are created in the DIT. Note that if the DIT is already populated with data through the use of `ldapaddent`, `ldapmodify`, or something similar, use of the `-i` or `-I` option overwrites that data. The `-r` option instructs `ypserv(1M)` to create and initialize the N2L cache with the data from the DIT. Note that `ypserv(1M)` does not run as a daemon when invoked with the `-i`, `-I` or `-r` options. Furthermore, the NIS server remains unavailable to client requests when administered in this manner.

Note that `rpc.ypupdated(1M)` is not supported in N2L mode. If the mapping file is present, `ypupdated` generates an informational message and exits.

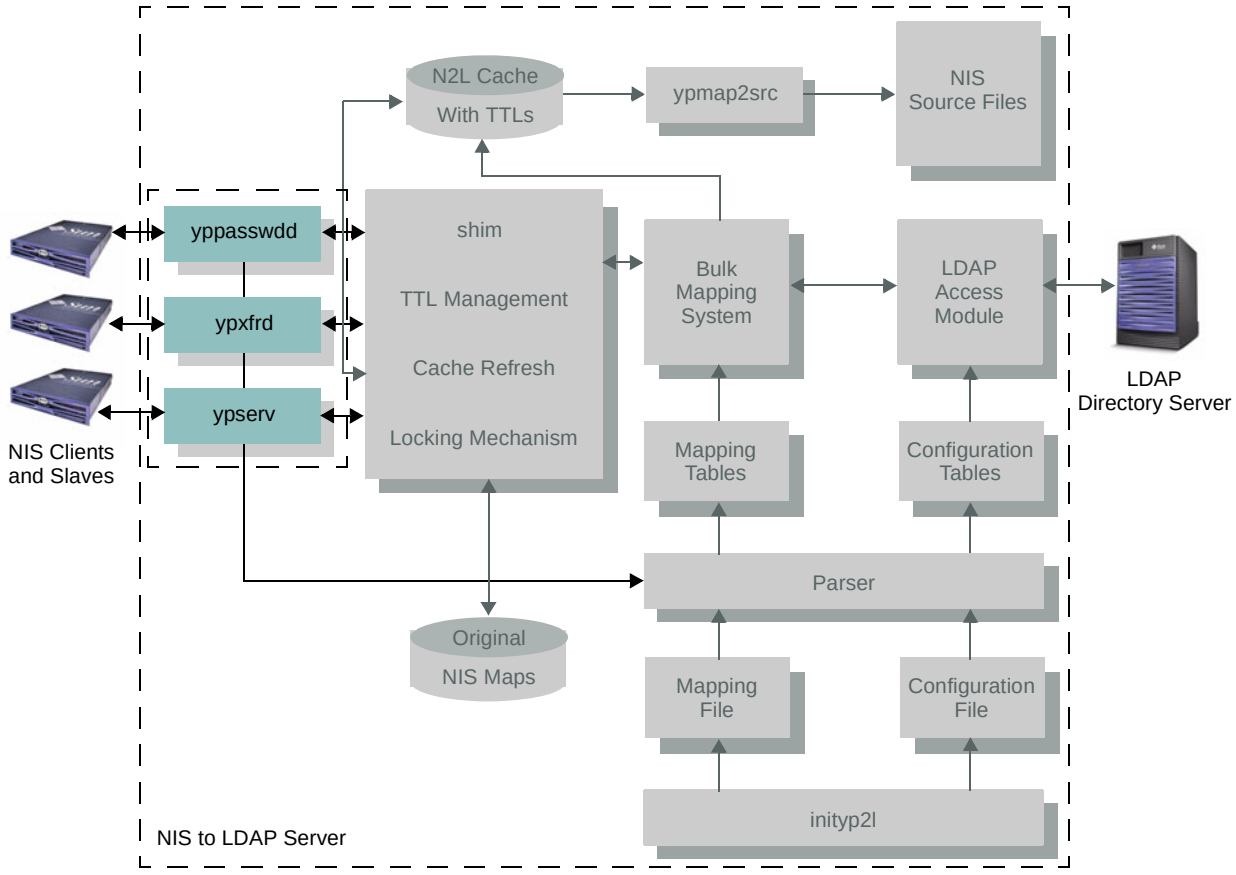

Figure 5. The front-end mechanism checks for the mapping file and starts daemons

N2L shim Mechanism

A traditional NIS server accesses its maps through an interface defined in `dbm(3UCB)`. In an N2L architecture, these calls are intercepted by routines collectively called the *shim* (Figure 6). This thin layer of code intercepts all `ndbm(3C)` library function calls, handles cache updates, and passes the calls to DBM. If a mapping file is not found, the shim passes all calls directly to the standard DBM interface. If a mapping file is found, it implements N2L cache functionality based on calls to `libldap` and the standard DBM interface.

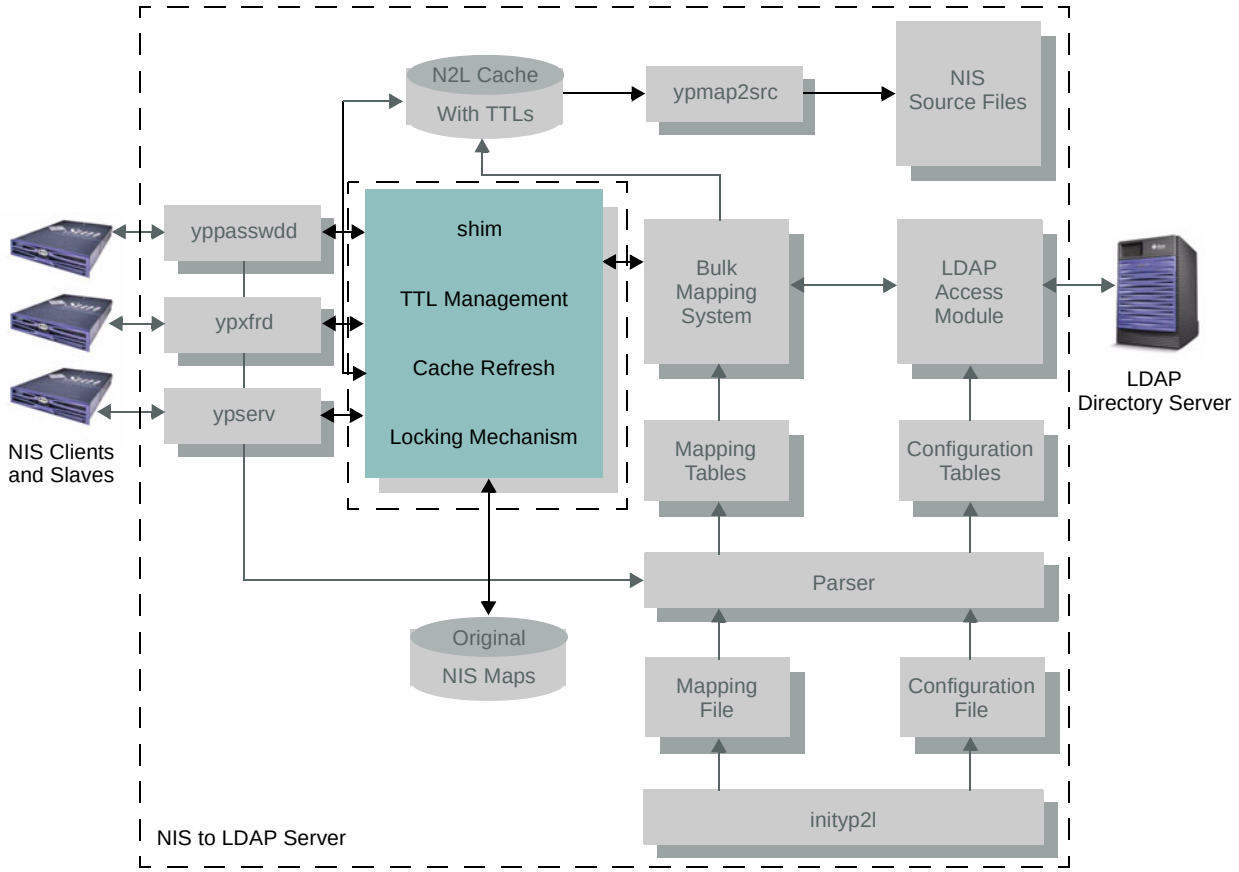

Figure 6. The shim intercepts calls to the NIS maps and translates them to `ndbm` library calls

N2L Mapping System

Implemented in the `libnisdb.so` library, the mapping system provides functionality to convert between NIS and LDAP entries using information from the mapping objects created by the parser (Figure 7). During the conversion process, the mapping system creates intermediate objects known as *rule-values*. Each rule-value object is a collection of `name=value` pairs.

The mapping module can either be invoked by the shim to refresh the cache, or directly by the front-end to upload or download data during initialization (`ypserv -I`, `ypserv -i`, or `ypserv -r`), or for password updates to the DIT (`yppasswdd`). It also creates any missing DIT containers if invoked by using the `ypserv -I` command.

The mapping unit converts IP addresses to preferred format, as defined in RFC2373, before writing to the DIT to ensure compatibility with the rfc2307bis standard. More than one mapping can be associated with a given map and domain. The mapping unit tries all mappings whose indexlist satisfy the NIS or LDAP entry in question. For more information see the `nisLDAPdatabaseIdMapping` section of the `NISLDAPmapping(4)` man page.

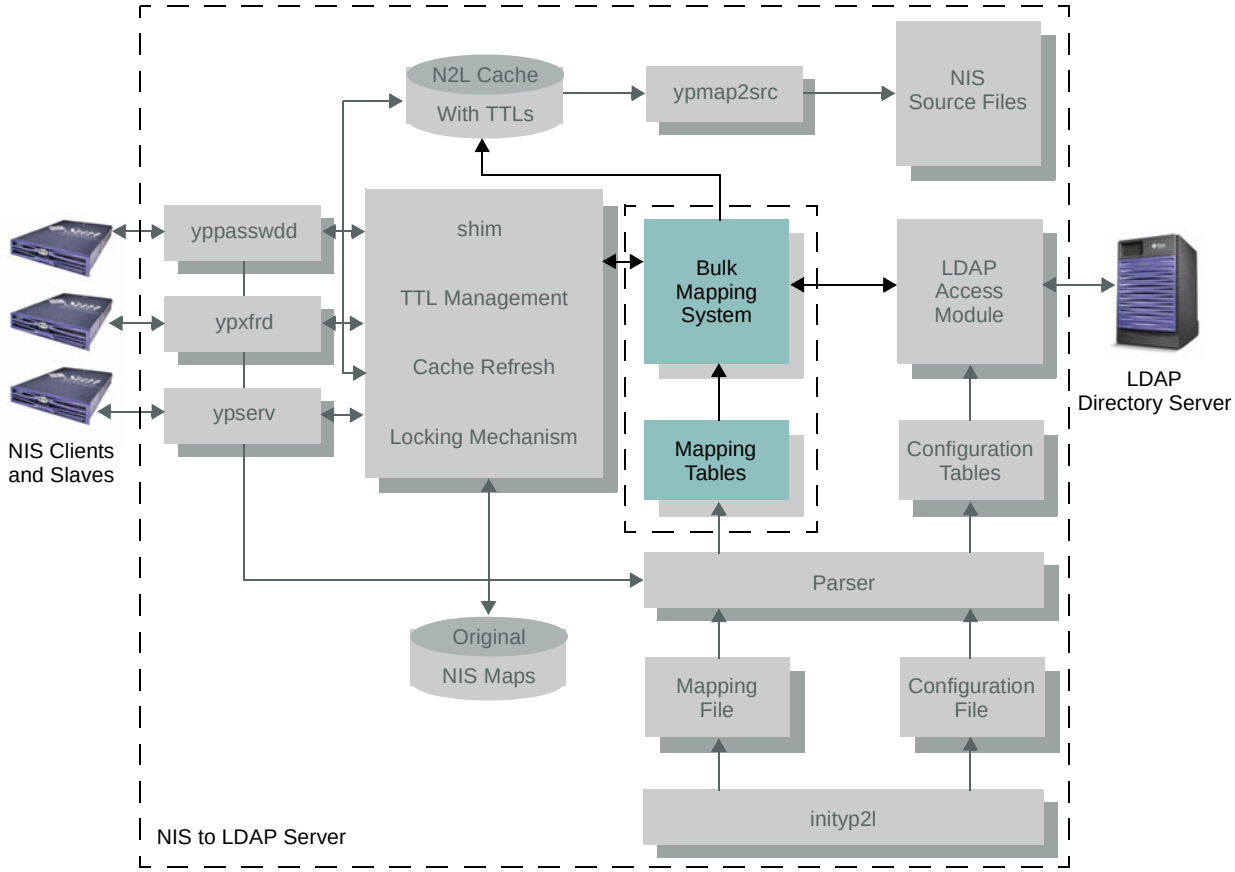

Figure 7. The mapping system converts between NIS and LDAP entries

N2L LDAP Access Module

The LDAP access module is a set of routines, implemented in libnisdb.so, that are called by the mapping system to interact with the LDAP server. To do so, this module uses the libldap library API, including such functions as `ldapbind(3LDAP)`, `ldapsearch(3LDAP)`, `ldapmodify(3LDAP)`, and more.

The module sets up and maintains a list of connections to each server in the preferred server list. After establishing a connection with a LDAP server, the access module determines if the LDAP server supports Simple Page (SP) and Virtual View List (VLV) controls. Although there is no requirement for the LDAP server to support the above controls, N2L prefers VLV or SP controls. More information can be found in the `ypserv(4)` man page.

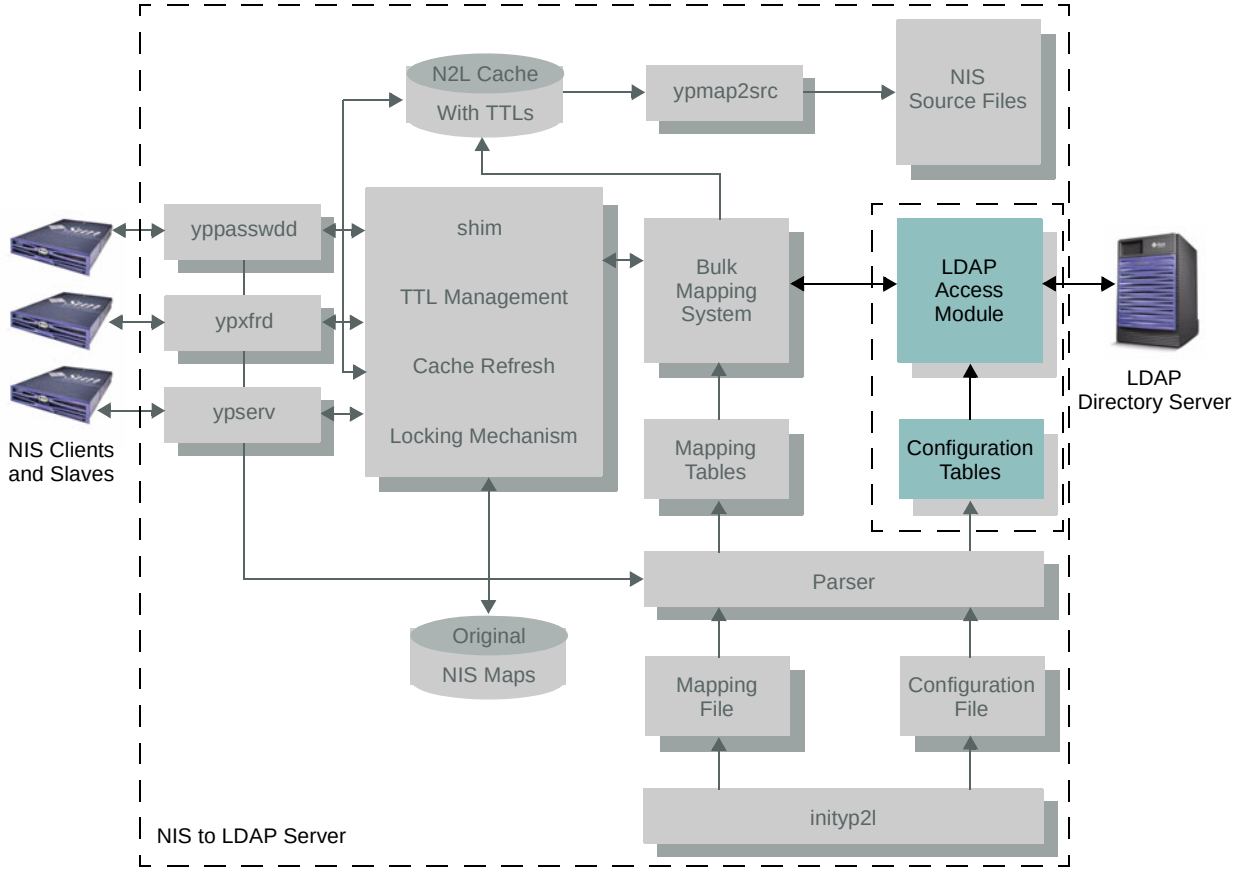

Figure 8. The LDAP access module interacts with LDAP servers

How the N2L Service Works

The N2L service can be understood by reviewing how it handles various types of client requests.

- *Client requests for an entire map*

  When a client requests an entire map, the NIS to LDAP server checks to see if the TTL for that map is valid. If the map TTL is valid data is immediately returned. However, if the map TTL has expired, and a cache refresh is not in progress, the NIS to LDAP server initiates a cache refresh thread in the background.

  The refresh thread collects the new data in a temporary location. Once the download completes, data is moved to the cache. The refresh time depends on many factors, including network traffic, the load on the LDAP and N2L servers, indexing of LDAP data, and the number of entries to be transferred. Rather than waiting for the cache refresh thread to complete, the N2L server returns data from the expired map to prevent blocking of the client request. All client requests for the map that coincide with refresh activity receive this data. All client requests made after the refresh completes receive refreshed data.

- *Client requests for a specific entry*

  When a client requests a specific entry, the NIS to LDAP server checks to see if the entry is in the cache. If the entry is in the cache, and the cache has not expired, the entry is returned to the client. If the entry is in an expired cache, and a cache refresh is not in process, the entry is refreshed from the LDAP directory and returned to the client. If, however, the entry is not found in the LDAP directory, it is deleted from the cache and the client is informed the entry does not exist. If the requested entry is not found in the cache, and the cache is expired, and cache refresh is not in progress, the NIS to LDAP server initiates an update thread in the background to refresh the entire map. The client is informed the entry did not exist. Subsequent calls return the refreshed data.

  During the refresh if multiple LDAP entries matches the given NIS key, only the first match is used. It is important to note that unlike a map refresh, single entry refreshes take place in the foreground. As a result, the client request either receives refreshed data or no data. In the event of a refresh error, the NIS to LDAP server initiates retry attempts. The number of retry attempts, as well as instructions for returning stale cached data or the `YPERR_YPERR` error message to the client, are defined in the `ypserv(4)` configuration file.

- *Client requests for changes to passwords*

  When the NIS to LDAP server receives a client request to change a password, the server obtains a domain list from the mapping file. If no domains are specified, it uses the domain set by the `domainname(1M)` command. For each domain, the user is authenticated using the existing password, and LDAP database and N2L cache are updated with the new password. All changes are pushed to the NIS slave servers. The default mapping file contains commented entries for C2 security. If C2 security is required, the mapping file must be modified to uncomment those entries. For more information see `nisLDAPyppasswddDomains` in `NISLDAPmapping(4)`.

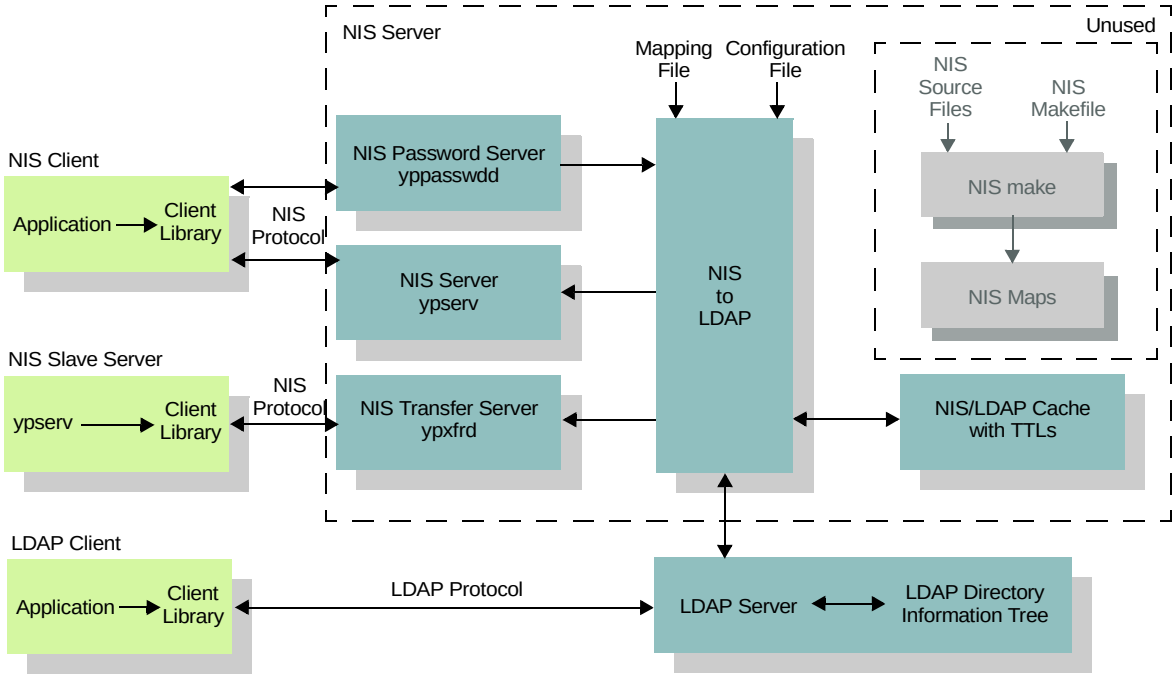

Figure 9. How the NIS to LDAP service works

N2L Service Error Reporting

Like most system daemons, the NIS to LDAP software uses `syslog(3C)`, the standard logging facility in the Solaris OS. This facility works by having programs transmit syslog messages to the syslog daemon, `syslogd(1M)`. A syslog message consists of a *header* and a *body*. The header consists of a tag string (usually the process name), facility code, severity code, timestamp, and process ID. The message body contains the information to be logged.

The `syslogd(1M)` daemon processes messages based on facility and severity codes. The facility code identifies the portion of the system that created the message, such the kernel (`kern`), mail system (`mail`), or authorization system (`auth`). The severity code lists the priority of the message: emergency (`emer`), error (`err`), or informational (`INFO`). This code pair is written as *facility.severity,* such as `kern.debug` or `mail.err`. The configuration information contained in `syslog.conf(4)` dictates how the daemon processes messages, including whether they are written to the console or a file, or transmitted over the network to a central logging host.

If the `ypserv(1M)` daemon is executed with the `-l`, `-i`, and `-r` options, a generic error message is displayed when problems are encountered. This error message states: `Aborting after NIS to LDAP mapping error`. If determining the exact problem is important, `syslog(3c)` must be configured. Most parsing errors display a line number—but may not represent a parsing syntax error. A semantic issue, or an actual error during upload (`-I`) or download (`-r`) may have occured. In addition to configuring `syslog(3C)`, run the `ldapsearch` command tool located in */usr/bin/ldapsearch* using the `bindDN`, `bindPW`, `auth method`, and `LDAP_server_IP:port` as specified in the */etc/default/ypserv* file. This enables typographic errors in the */etc/default/ypserv* configuration file to be found.

Enabling Warning and Informational Messages

The default *syslog.conf* file may not enable warnings and informational messages. Since N2L uses these severity levels to report additional information, it is important to enable them using the following procedure.

1. Become the superuser.
2. Edit the */etc/syslog.conf(4)* file and append `*.warning; *.info` as in the following entry. Note that a tab must separate the two fields.

```
.err;kern.debug;daemon.notice;mail.crit;*.info;*.warning     /var/adm/messages
```

3. Restart the `syslog(3C)` process using the following command sequence.

```
For systems running the Solaris 10 OS:
# svcadm restart svc:/system/system-log:default

For systems running older versions of the Solaris OS:
# /etc/init.d/syslog restart
```

4. View system diagnostic messages by running the `dmesg` command, or the `tail` command on the */var/adm/messages* file.

More information can be found in the `syslog(3C)`, `syslogd(1M)`, and `syslog.conf(4)` man pages.

Setting Up N2L

Before reviewing the steps for setting up N2L certain terminology must be defined.

- Information is stored in the LDAP server in the form of a tree called the Directory Information Tree (DIT). Every node contains data, and any node can be a container. As a result, an LDAP entry may have child nodes.
- The root of the DIT is referred to as the naming context or root suffix, and is created when setting up the LDAP server. Note that if the naming contexts are disjoint, several DITs may exist. However, if everything is contained under a single DN, and the suffixes are sub-suffixes of this DN, the directory server can be considered to have a single DIT.
- Some, but not all, LDAP containers are equivalent to NIS dbm maps. For example, the `ou=hosts` container contains host data, while the `ou=people` container contains POSIX user data, and so on.
- The `baseDN` is the container, or node, in the DIT that holds the naming-specific containers. This node and its underlying nodes are created by running the `idsconfig(1M)` or `ypserv -I` command.
- Either the node representing the root suffix, or a different node under the root suffix, can be used as the `baseDN`.
- The LDAP entry representing the `baseDN` contains the `nisDomain` attribute whose value is the NIS `domainname` being served.

Assumptions

This section makes the following assumptions:

- The NIS `domainname` is `mydomain.mycompany.com`
- The DIT root suffix is `dc=mycompany,dc=com`

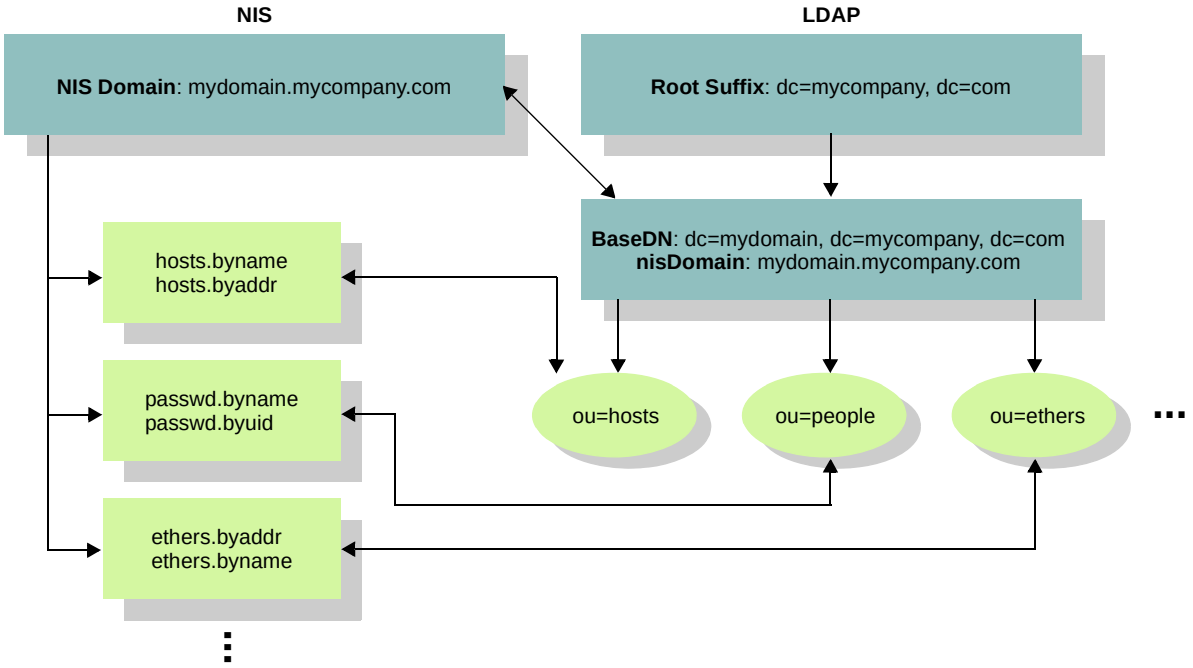

Figure 10. The N2L configuration

N2L Setup Procedure

Perform the following steps on the LDAP server.

1. Backup all configuration data that may change in the following steps. See the *Sun Java™ System Directory Server 5.2 Administration Guide, Chapter 4: Backing Up and Restoring Data* for more information.
2. Install and setup the Directory Server. For systems running the Solaris OS, see the `directoryserver(1M)` manpage or Directory Server documentation for more information.
3. Run the `idsconfig(1M)` command to initialize the Directory Server to store naming information on systems running the Solaris OS. This creates LDAP entries for the `baseDN` and the naming containers. The containers are empty.

Perform the following steps on the NIS master.

1. Stop any NIS services that are currently running using the `ypstop(1M)` command.
2. Configure the NIS master as an N2L server using the `inityp2l(1M)` command. This creates `ypserv(4)` and `NISLDAPmapping(4)` configuration files.
3. Modify the configuration as needed. To add custom mappings to the *NISLDAPmapping* file, see the *Representing Custom Maps in the DIT* section below.
4. If the LDAP server being used is already populated with naming data, proceed to step 6. Otherwise, run the `ypserv -I` command to migrate NIS data to the DIT.
5. Check the output of the `ypserv -I` command and the */var/adm/messages* file for errors.
6. If successful, run the `ypserv -r` command to initialize the N2L cache. Note the two commands can be merged into a single `ypserv -Ir` command.
7. Check the output of the `ypserv -r` command and the */var/adm/messages* file for errors.
8. If successful, start the NIS services using the `ypstart(1M)` command.

Representing Custom Maps in the DIT

1. *Does the information already exist?*

   The first step is to determine whether the information in the custom NIS map must be explicitly stored in the DIT, or if the information can be reconstructed from data in other containers. For example, consider the *netid.byname* NIS map. This map contains information on mappings between netnames and userids or hostnames in the local domain. The information in this map can be reconstructed using the data in the `passwd(4)`, `hosts(4)`, and `groups(4)` maps and the local domain name. Therefore, the default N2L mapping for *netid.byname* does not write information from this map to any container in the DIT. It reconstructs the N2L *netid.byname* cache *(LDAP_netid.byname.**) by pulling information from the containers representing `passwd`, `hosts`, and `groups`. Refer to the netid mappings located in the default `NISLDAPmapping(4)` file created by the `inityp2l(1M)` script.

2. *Can the information be stored in other LDAP entries ?*

   Sometimes a single LDAP entry can hold information belonging to multiple NIS maps. For example, every entry in the LDAP container for `passwd(1)` can contain information belonging to `passwd(1)`, `publickey.byname`, `user_attr(4)`, and `audit_user(4)`. When data is uploaded from N2L to the DIT, mapping writes information for each map above to the LDAP entry. During download or

refresh, it pulls only the information needed to construct the corresponding N2L cache. For more information see the `passwd`, `keys.pass`, `user_attr`, and `audit_user` mappings in the default `NISLDAPmapping(4)` file. Note that LDAP entries support the notion of `STRUCTURAL` and `AUXILIARY` object classes. Mappings that add `STRUCTURAL` object classes should appear before those that add `AUXILIARY` object classes in the mapping file. The `nisLDAPobjectDN` attribute determines the order in which NIS maps are written to the DIT.

3. *Pre-defined schemas or custom schemas ?*

   Refer to the LDAP server documentation to determine if pre-defined schemas can be used for LDAP entries. If an appropriate pre-defined LDAP schema cannot be found, custom schemas can be defined in the LDAP server. See the LDAP server documentation for more information on LDAP schemas. Be sure to add any missing or custom schemas to the LDAP server instance before using them in the mapping file.

4. *nisObject object class*

   The `nisObject` object class was proposed by RFC2307. According to this RFC "*The nisObject object class MAY be used as a generic means of representing NIS entities. Its use is not encouraged; where support for entities not described in this schema is desired, an appropriate schema should be devised. Implementors are strongly advised to support end-user extensible mappings between NIS entities and object classes. (Where the nisObject class is used, the nisMapName attribute may be used as a RDN.)*" See Example 2 below for using this object class.

Steps to Transfer Custom Maps

1. Refer to the list of assumptions and verify system information.
2. Verify the custom LDAP schema(s) exists or has been added to the LDAP server instance.
3. Modify the */var/yp/NISLDAPmapping* file, adding the mapping entries for the custom maps in the appropriate location. See Example 1 of the *Examples Using Custom Maps* section below for more details.

Note – An alternative is to complete the transfer of standard maps first using Steps 4 through 6.

---

4. Run the `ypserv -Ir` command.
5. Verify the LDAP entries using the `ldapsearch(1)` command.

```
$ ldapsearch -h ldap_server -b 'ou=photocopiers,dc=mydomain,dc=mycompany,dc-com" \
         -s sub "(objectclass=*)"
```

6. Determine whether the N2L cache for the custom maps matches the default NIS custom map.

```
$ makedbm -u /var/yp/mydomain.mycompany.com/photocopiers.byname

For N2L cache:

$ makedbm -u /var/yp/mydomain.mycompany.com/LDAP_photocopiers.byname
```

7. Start the NIS services.

```
On Solaris 9 OS and earlier:
$ /usr/lib/netsvc/yp/ypstart

On Solaris 10 OS:
$ svcadm enable svc:/network/nis/client
```

Examples Using Custom Maps

Example 1

Consider a hypothetical NIS map `photocopiers.byname` containing information on photocopiers in an office, including the name, serial number, owner, and a short description. Assume the map is keyed on the name, with the colon (':') serving as a delimiter between fields.

```
$ makedbm -u photocopiers.byname
copier-11 copier-11:1234511:Baban Kenkre:ABC photocopier
copier-12 copier-12:1234512:Michael Haines:XYZ photocopier
```

This example uses the object class named `device`, defined by RFC 2256, to create photocopier entries in the DIT. Using this approach, the `photocopiers.byname` NIS map can be stored in the LDAP DIT as follows:

```
dn: ou=photocopiers,dc=mydomain,dc=mycompany,dc=com
ou: photocopiers
objectClass: organizationalUnit
objectClass: top

dn: cn=copier-11,ou=photocopiers,dc=mydomain,dc=mycompany,dc=com
objectClass: top
objectClass: device
cn: copier-11
serialNumber: 1234511
owner: Baban Kenkre
description: ABC photocopier

dn: cn=copier-12,ou=photocopiers,dc=mydomain,dc=mycompany,dc=com
objectClass: top
objectClass: device
cn: copier-12
serialNumber: 1234512
owner: Michael Haines
description: XYZ photocopier
```

This can be configured by using the following `NISLDAPMapping` entries:

```
nisLDAPdomainContext mydomain.mycompany.com :
dc=mydomain,dc=mycompany,dc=com

nisLDAPentryTtl photocopiers.byname:1800:5400:3600

nisLDAPcommentChar photocopiers.byname: ''

nisLDAPnameFields photocopiers.byname: \
("%s:%s:%s:%s", name, serialnum, own_by, details)

nisLDAPobjectDN photocopiers.byname: \
ou=photocopiers,?one? \
objectClass=device:

nisLDAPattributeFromField photocopiers.byname: \
dn=("cn=%s,", rf_key), \
cn=rf_key, \
serialNumber=serialnum, \
owner=own_by, \
description=details

nisLDAPfieldFromAttribute photocopiers.byname: \
rf_key=cn, \
serialnum=serialNumber, \
own_by=owner, \
details=description
```

Example 2

Continuing with the `photocopiers.byname` nis map example, another approach is to use the `nisObject` object class to store the information in the LDAP DIT as follows:

```
dn: nisMapName=photocopiers,dc=mydomain,dc=mycompany,dc=com
objectClass: top
objectClass: nisMap
nisMapName: photocopiers.byname

dn: cn=copier-11,nisMapName=photocopiers,dc=mydomain,dc=mycompany,dc=com
objectClass: top
objectClass: nisObject
cn: copier-11
nisMapName: photocopiers.byname
nisMapEntry: copier-11:1234511:Baban Kenkre:ABC photocopier

dn: cn=copier-12,nisMapName=photocopiers,dc=mydomain,dc=mycompany,dc=com
objectClass: top
objectClass: nisObject
cn: copier-12
nisMapName: photocopiers.byname
nisMapEntry: copier-12:1234512:Michael Haines:XYZ photocopier
```

This can be configured by using the following `NISLDAPMapping` entries:

```
nisLDAPdomainContext mydomain.mycompany.com :
dc=mydomain,dc=mycompany,dc=com

nisLDAPentryTtl photocopiers.byname:1800:5400:3600

nisLDAPcommentChar photocopiers.byname: ''

nisLDAPnameFields photocopiers.byname: \
("%s", value)

nisLDAPobjectDN photocopiers.byname: \
nisMapName=photocopiers,?one? \
objectClass=nisObject:

nisLDAPattributeFromField photocopiers.byname: \
dn=("cn=%s,", rf_key), \
cn=rf_key, \
nisMapName=("photocopiers.byname"), \
nisMapEntry=value

nisLDAPfieldFromAttribute photocopiers.byname: \
rf_key=cn, \
value=nisMapEntry
```

Example 3

Continuing with the photocopiers example introduced in Example 1, assume there are two NIS dbm maps: `photocopiers.byname` and `photocopiers.bynumber`. These NIS maps contain the same photcopier data but uses a different field as the key. Note that `photocopiers.byname` is identical to Example 1.

```
$ makedbm -u photocopiers.bynumber
1234511 copier-11:1234511:Baban Kenkre:ABC photocopier
1234512 copier-12:1234512:Michael Haines:XYZ photocopier
```

Although NIS requires two dbm maps—each using a different key—to represent the same data, the LDAP DIT does not have this limitation. As a result, both NIS maps can be mapped to the same LDAP container. The same pre-defined `device` object class used in Example 1 is used for the LDAP entries. The information in the LDAP DIT is identical to that in Example 1. This can be configured by using the following `NISLADPmapping` entries:

```
nisLDAPdomainContext mydomain.mycompany.com : dc=mydomain,dc=mycompany,dc=com

# The following allows us to use the identifier "photocopiers" to
# represent information common to photocopiers.byname and photocopiers.bynumber.
nisLDAPdatabaseIdMapping photocopiers: photocopiers.byname photocopiers.bynumber

nisLDAPentryTtl photocopiers:1800:5400:3600

nisLDAPcommentChar photocopiers: ''

nisLDAPnameFields photocopiers: \
("%s:%s:%s:%s", name, serialnum, own_by, details)

nisLDAPobjectDN photocopiers.byname: \
ou=photocopiers,?one? \
objectClass=device:

# Disable writes from N2L to DIT for photocopiers.bynumber
# since the same data will be written by the above
# photocopiers.byname mapping. Note the absence of the
# trailing ":" below to disable writes

nisLDAPobjectDN photocopiers.bynumber: \
ou=photocopiers,?one? \
objectClass=device

nisLDAPattributeFromField photocopiers.byname: \
dn=("cn=%s,", rf_key), \
serialNumber=serialnum, \
cn=rf_key
nisLDAPattributeFromField photocopiers.bynumber: \
dn=("cn=%s,", name), \
serialNumber=rf_key, \
cn=name
nisLDAPattributeFromField photocopiers: \
owner=own_by, \
description=details

nisLDAPfieldFromAttribute photocopiers.byname: \
rf_key=cn
nisLDAPfieldFromAttribute photocopiers.bynumber: \
rf_key=serialNumber
nisLDAPfieldFromAttribute photocopiers: \
name=cn, \
serialnum=serialNumber, \
own_by=owner, \
details=description
```

N2L Toolkit

All of the source code and scripts contained in the N2L Toolkit are presented in the following sections. Users should have the requisite expertise and access to the appropriate developer tools before using the tools described below.

The `TTLDUMP` Utility

The `ttldump` utility prints out entries from a N2L map and displays the remaining TTL for each entry, as well as the entire map. The following output is representative of the information displayed when running the `ttldump` utility against an existing map.

```
$ more ttldump.out
TTL file found keys are:-
Key (len 14)= "YP_EXPIRY_TIME" Value 1084871976s (-2550s remaining)
Key (len 20)= "YP_OLD_MAP_DATE_TIME" Value Not found

Normal keys are:-
Key (len 14)= "submission/udp" Value (len 37)= "submission 587/udp  #    see RFC 2476"
    TTL = 1084872820s (-1706s remaining)
Key (len 13)= "timserver/tcp" Value (len 21)= "time 37/tcp timserver"
    TTL = 1084870669s (-3857s remaining)
Key (len 12)= "csnet-ns/tcp" Value (len 17)= "csnet-ns 105/tcp "
    TTL = 1084872092s (-2434s remaining)
Key (len 9)= "ldaps/tcp" Value (len 55)= "ldaps 636/tcp  # LDAP protocol over TLS/SSL (was sldap)"
    TTL = 1084870724s (-3802s remaining)
```

Several pieces of information are important:

- `YP_EXPIRY_TIME`

  `YP_EXPIRY_TIME` is the per-map TTL. The TTL value is the time (on the UNIX system clock) when it will expire. The value in parentheses indicates the number of seconds remaining before the TTL expires. If the map has already expired, this value is negative. In this example, the entire expired map will be refreshed when a client requests the entire map, or attempts to access an entry that is not found inside the map. When the refresh completes, the map expiration time is set to the current clock plus the running TTL for the map.

- `YP_OLD_MAP_DATE_TIME`

  This value stores the date the old NIS map file was last updated. This is used when checking if the `ypmake` utility was run by mistake. In this example, *Value Not found* indicates there is probably no entry—either an old style map does not exist (new maps were pulled from LDAP), or this map was never initialized due to it never being accessed.

- `Normal keys`

  This is where normal map entries are listed. For each entry, the output consists of the NIS key (key length and string), Value (value length and string), and the entry TTL. A negative entry TTL indicates the entry has expired. An expired entry is refreshed when it is requested by a client.

- First key

  This value pair consists of a length and a string. The `(len = X)` sections indicate the length of the string, followed by the string. The key is `submission/udp` and the value *"submission 587/udp # see RFC 2476"*. Note that non-printable characters in the key or string are printed as numbers.

- TTL of the first key

  The last item in the output is the TTL of the first key. In this example, the TTL has expired. This is not a problem or error. Upon the next YP access to this entry, the entry and TTL are updated, and the new value returned.

The `ttldump` provides information similar to the `ypcat` tool, and more—it displays *special* entries (those starting with `YP_`), as well as the TTLs which can be used when YP is not responding, making it useful for investigating failures.

`TTLDUMP` Utility Source Code

Following is the source code for the `ttldump` utility.

```
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
#pragma ident   "@(#)ttldump.c      1.0    06/02/07 SMI"

#include <stdio.h>
#include <fcntl.h>
#include <strings.h>
#include <stdlib.h>
#include <ctype.h>
#include <ndbm.h>

/*
 * Test program to dump N2L maps and associated TTLs.
 *
 * USAGE : ttldump <N2L Mapname>
 * E.g.  : To dump N2L's netgroup map : ttldump LDAP_netgroup
 *
 * NOTES :
 *
 * Prints out the DBM file and associated TTL values (if any) in human
 * readable form. Non printable characters are displayed numerically.
 *
 * Negative TTLs are expected. These are entries that have timed out
 * but not been accessed recently.
 *
 * Special entries in the DBM file, that start 'YP_', have no TTLs
 *
 * The 'whole map' TTL is the YP_EXPIRY_TIME entry.
 *
 */

#define   TRUE1
#define   FALSE0

void dump_datum(datum *, int);
void usage(char *);
```

```
int
main(int argc, char *argv[]) {
    datum   ttl_key, key, data;
    DBM     *db, *ttl_db;
    char    *ttl_str;

    if (argc != 2)
        usage(argv[0]);

    db = dbm_open(argv[1], O_RDONLY, 0);
    if (db == NULL) {
        fprintf(stderr, "Could not open %s\n", argv[1]);
        exit(1);
    }

    ttl_str = (char *)malloc(strlen(argv[1]) + strlen("_TTL") + 1);
    if (ttl_str == NULL) {
        fprintf(stderr, "Memory allocation failure\n");
        exit(1);
    }

    strcpy(ttl_str, argv[1]);
    strcat(ttl_str, "_TTL");

    ttl_db = dbm_open(ttl_str, O_RDONLY, 0);
    free(ttl_str);
    if (ttl_db == NULL) {
        printf("TTL file not found\n");
    } else {
        /* Get + print special TTL values */
        printf("TTL file found\n");
        key.dptr = "YP_EXPIRY_TIME";
        key.dsize = strlen(key.dptr);
        data = dbm_fetch(ttl_db, key);
        printf("Map TTL = ");
        dump_datum(&data, TRUE);
        printf("\n");

        key.dptr = "YP_OLD_MAP_DATE_TIME";
        key.dsize = strlen(key.dptr);
        data = dbm_fetch(ttl_db, key);
        printf("Old map update time = ");
        dump_datum(&data, TRUE);
        printf("\n");
    }

    printf("\nData :\n");
    for (key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db)) {
        data = dbm_fetch(db, key);
        printf("Key");
        dump_datum(&key, FALSE);
        printf("\tValue");
        dump_datum(&data, FALSE);
```

```
        /* Dump TTL info for entry (if any) */
        if (ttl_db != NULL) {
            /* Special keys don't have TTLs */
            if (0 != strncmp(key.dptr, "YP_", strlen("YP_"))) {
                data = dbm_fetch(ttl_db, key);
                printf("\n\tTTL= ");
                dump_datum(&data, TRUE);
            }
        }
        printf("\n");
    }

    if (ttl_db != NULL)
        dbm_close(ttl_db);
    dbm_close(db);
    return (0);
}

void
usage(char *prog_name) {
    fprintf(stderr, "Sun Microsystems Inc.\n");
    fprintf(stderr, "NIS to LDAP Map ttldump utility.\n");
    fprintf(stderr, "Usage: %s <N2L-mapname>\n", prog_name);
    fprintf(stderr, "\tEx: ttldump LDAP_netgroup\n");
    exit(1);
}
void
dump_datum(datum *data, int ttl) {
    struct timeval  tv;
    struct timeval  now;
    int i;

    if (data->dptr == NULL) {
        printf("Not found");
        return;
    }
    if (ttl == FALSE) {
        printf("(len %d)= \"", data->dsize);
        for (i = 0; i < data->dsize; i ++)
            if (isprint((int)data->dptr[i]))
                printf("%c", data->dptr[i]);
        else
                printf("(%d)", data->dptr[i]);
        printf("\"");
    } else {
        /* Get current time */
        gettimeofday(&now, NULL);

        /* Have to get the alignment right */
        bcopy(data->dptr, &tv, sizeof (struct timeval));
        printf("%ds (%ds remaining)", tv.tv_sec,
                    tv.tv_sec - now.tv_sec);
    }
}
```

`update-slave-servers.sh` Script

Following is the source to the update-slave-servers.sh script.

```
#!/bin/ksh
#
# ident "@(#)update-slave-servers.sh       1.0    06/02/07 SMI"
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

DMN=`/bin/domainname`
if [ -d "/var/yp/$DMN" ]; then
    found="false"
    for n2lmap in /var/yp/$DMN/LDAP_*_TTL.dir
    do
        map=`/usr/bin/basename "$n2lmap" _TTL.dir | cut -c 6-`
        if [ "$map" != "*" ]; then
            echo yppush -v -d $DMN $map
            /usr/lib/netsvc/yp/yppush -d $DMN $map
            echo "yppush = " $?
            found="true"
        fi
    done
    if [ "$found" == "false" ]; then
        echo "/var/yp/$DMN: N2L maps not found"
    fi
else
    echo "/var/yp/$DMN: not found"
fi
```

MakeFile

Following is the source to the makefile for creating the `ttldump` utility.

```
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "@(#)Makefile   1.0     06/02/07 SMI"
#

PROG= $(TTLDUMPPROG) $(UPDATESLAVESERVERSPROG)

ROOTBINDIR= ../bin
ROOTPROG=  $(PROG:%=$(ROOTBINDIR)/%)

TTLDUMPPROG= ttldump
TTLDUMPOBJ= ttldump.o
TTLDUMPSRC= $(TTLDUMPOBJ:.o=.c)

UPDATESLAVESERVERSPROG= update-slave-servers
UPDATESLAVESERVERSSRC= update-slave-servers.sh

OBJS= $(TTLDUMPOBJ)
SRC= $(TTLDUMPSRC) $(UPDATESLAVESERVERSSRC)
```

```
INS= install
RMDIR= /usr/bin/rmdir
CP= /usr/bin/cp -f
CHMOD= /usr/bin/chmod
DIRMODE= 755
EXECMODE= 555
INS.dir= $(INS) -s -d -m $(DIRMODE) $@
INS.executable= $(RM) $@; $(INS) -s -m $(EXECMODE) -f $(@D) $<

all: $(PROG)

$(ROOTBINDIR):
    $(INS.dir)

$(ROOTBINDIR)/%: $(ROOTBINDIR) %
    $(INS.executable)

$(TTLDUMPPROG): $(TTLDUMPOBJ)
    $(LINK.c) -o $@ $(TTLDUMPOBJ)

$(UPDATESLAVESERVERSPROG): $(UPDATESLAVESERVERSSRC)
    $(CP) $(UPDATESLAVESERVERSSRC) $@
    $(CHMOD) $(EXECMODE) $@

clean:
    -$(RM) $(OBJS)

clobber: clean
    -$(RM) $(PROG)

install: all $(ROOTPROG)

uninstall: clobber
    -$(RM) $(ROOTPROG)
    -$(RMDIR) $(ROOTBINDIR)
FRC:
```

Readme File

Following is the source to the N2L Toolkit README file.

```
Directory structure:
--------------------

n2lkit/README:= This file
n2lkit/src:= Source code
n2lkit/bin:= executables


Build Instructions:
-------------------

Build the sources, create "n2lkit/bin" directory and install the
executables in it:
    cd src
    make install
```

```
Uninstall Instructions:
-----------------------

Uninstall the executables and remove "n2lkit/bin":
(Note: This will not remove the sources and the package)
    cd src
    make uninstall
Uninstall the package:
(Note: This will REMOVE the sources and the package)
    cd ~
    pkgrm SUNWn2lkitsrc


Executables:
------------

ttldump:
    dumps the TTL of the N2L cache. For usage utter "ttldump"
    at the prompt.

update-slave-servers:
    pushes all N2L maps to NIS slave servers. Uses domainname(1M)
    to determine the NIS domain.
```

Conclusion

Sun is committed to the LDAP protocol as a naming service. Indeed, the Sun Java System Directory Server offers LDAP functionality for the Solaris 9 OS and Solaris 10 OS, including strong support for industry standards, scalability, and flexibility in deployment. Migrating from the NIS naming service environment to the secured LDAP environment in the Solaris 8, 9, and 10 OS offers many benefits. LDAP directories can:

- Serve as a common repository for user identity and management. A streamlined identity infrastructure can help reduce overall administrative tasks.
- Provide a building block for addressing future requirements, such as service on demand initiatives.
- Deliver the ability to reuse an existing identity repository for new applications, contributing to an increased return on investment and faster time-to-service.

By integrating the Sun Java System Directory Server into the Solaris 9 OS and Solaris 10 OS, Sun delivers a naming and directory infrastructure that provides a centralized repository for user management and system configuration data. This information can be accessed through industry standard protocols, making it available throughout the organization. This results in a competitive total cost of ownership with a total network-wide application quality of service.

About the Authors

Michael Haines is a Senior Engineer at Sun. Since joining Sun in 1989, Michael has worked in various engineering roles, with emphasis on the Solaris Operating System and naming services. Michael is also the co-author of two Sun BluePrints books: *LDAP in the Solaris Operating Environment,* and *Solaris and LDAP Naming Services*.

Baban Kenkre is a Senior Engineer at Sun, and has worked in the Solaris Directory and Security Technology Group for more than five years.

Acknowledgments

The authors thank the following for their advice and contributions to this article:

- Martin Reifenrath, Remote Services Delivery—Solaris Networking
- Bertil Lindblad, Senior Technical Consultant
- Stacey Marshall, Senior Software Engineer
- Serge Dussud, Senior Software Engineer, Operating Platform Group
- Jon Tricker, Solaris Security Technology Engineering

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN **Web** sun.com