**Original Draft Published: May 14, 2001**
**Last Revised: April 11, 2008**

# Native LDAP Phase II (and later) libsldap and related Architecture Specification

**This document describes SUNW Project private internal interfaces used by naming service components to provide LDAP Naming service support. These interfaces are prohibited from use outside the naming service project except with explicit PSARC contracts on these interfaces. Any use of these interfaces outside the naming service project or outside contract boundaries is completely unsupported by Sun. These interfaces are very volatile and are subject to change in incompatible ways as frequently as a build by build basis. This document also describes other commited components related to these interfaces, the LDAP tools APIs and the LDAP schema required to support the LDAP Naming service.**

**NOTE: this is an evolving document and represents the library at the time of last revision.**

## 1.0 Native LDAP Phase 2 Introduction

The Lightweight Directory Access Protocol (LDAP) has emerged as the industry standard protocol for accessing directory servers. The dominant protocol independent interfaces to naming services within Solaris are the standard getXbyY APIs. In order for Solaris to successfully be part of LDAP based directory service, an LDAP backend to the Naming Service switch is required.

The purpose of the original Native LDAP phase 1 and phase 2 efforts were to provide LDAP naming service integration into Solaris. This means supporting LDAP service through the Naming Service switch allowing Solaris clients to obtain Naming information from an LDAP-enabled directory. It also included transition procedures, support for password update, and updating Solaris core applications to be LDAP friendly. In other words, it should behave like other Solaris Naming services, NIS and NIS+.

Subsequent enhancements of naming services, including the sparks [http://www.opensolaris.org/os/project/sparks/] and Duckwater [http://www.opensolaris.org/os/project/duckwater/] OpenSolaris projects continue to evolve these interfaces in an ongoing, some times incompatible manner.

## 2.0 Table of Contents

Section 3.0 Overview of the differences between phase 1 and phase 2

Section 4.0 Architecture layers of the Naming Service switch with the LDAP backend.

Section 5.0 Simplified LDAP interfaces

Section 6.0 Format of a local LDAP configuration.

Section 7.0 LDAP cache manager daemon is described.

Section 8.0 Security model required by the Simplified LDAP API.

Section 9.0 Administration commands and user tools.

Section 10.0 Other components making up the project and the LDAP switch backend.

Section 11.0 Clients setup procedure.

Section 12.0 Interoperability issues with servers and clients.

Section 13.0 Schema used in the directory.

## 3.0 Key Phase 2 Enhancements

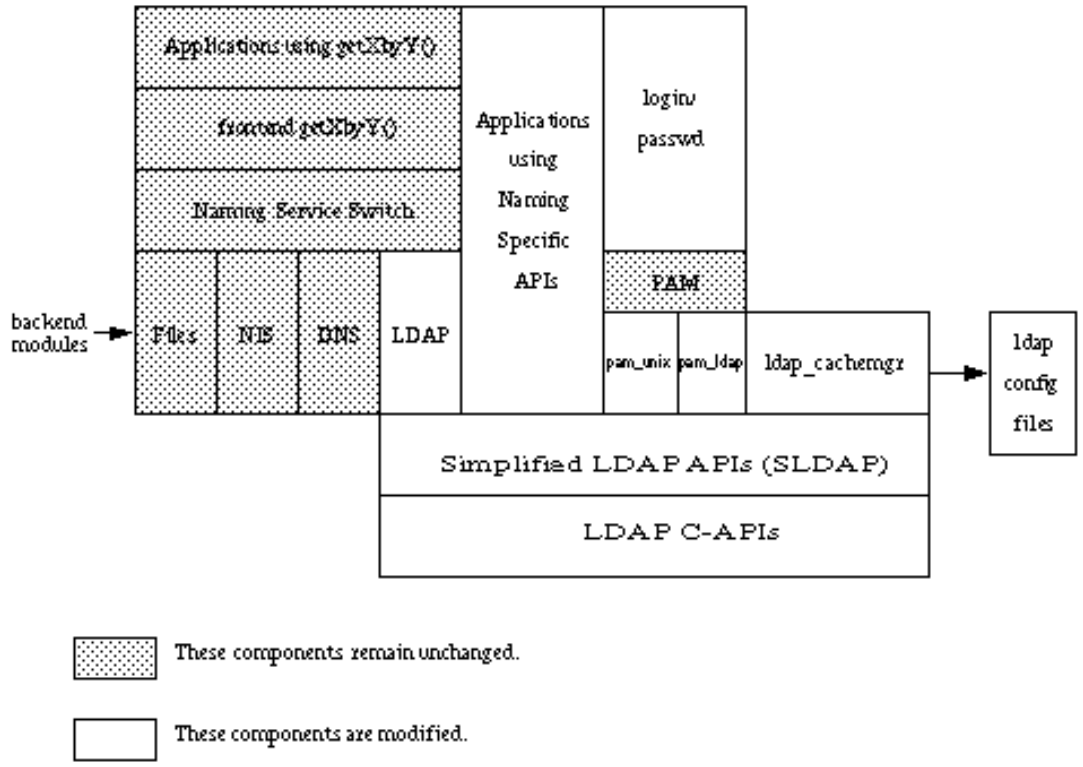
The long term goal is to evolve the Simplified LDAP API (libsldap) into an API that might be exposed outside the consolidation. This API must provide sufficient functionality to meet the needs described in the LDAP phase II and later enhancements, help to create tools that will improve the ease of setup for LDAP naming services, provide more flexible use of the directory through attribute mapping, and provide a stronger security model.

The following is a list of enhancements provided in the library. This is not intended as an all inclusive list.

- The Duckwater project [PSARC/2007/694 and related] are providing standalone and other ongoing enhancements to make usability, installation and management of client configurations more robust.
- Sparks [PSARC/2005/133] provided feature and other ongoing enhancements and interface changes to support per-user authentication and shared connection management.
- Phase 2 provides a flexible interface to the Simplified LDAP library so that the Simplified LDAP API can re-map the rfc2307bis attribute set into an existing directory setup.
- Phase 2 updates the security model to support strong authentication and TLS encrypted sessions, as well as providing the ground work for adding support for password management.
- The phase 1 concept of Domains have been removed to simplify the API. Domains were never supported completely in the original Simplified LDAP API and do not add value with the introduction of the new profile functionality.

It is anticipated that other projects such as Winchester will also evolve libsldap in possible upwards compatible and incompatible manners.

## 4.0 Simplified LDAP Architecture




The figure above illustrates the current Naming Service Switch architecture and how the LDAP switch backends fits into the switch engine. It also illustrates how the LDAP switch backend, some Solaris applications, and login/passwd utilities are layered on top of the Simplified LDAP API. Additionally, the components which are modified as part of the second phase are identified.

### 4.1 Interface Overview

LDAP which stands for Lightweight Directory Access Protocol is difficult to use. It requires many LDAP API calls in order to setup a connection to the server, retrieve the data and to parse the results. In addition, it requires that the caller locate and specify the LDAP server and understand the security mechanism. The simplified LDAP naming interface provides a simpler way of accessing the data where callers need to only make one call to retrieve their naming information from the LDAP server. All the relevant LDAP connection information such as server names may be pre-configured and stored in the active name service back-end configuration (NSBEC). This not only simplifies the LDAP name server switch back-end implementation, but it also simplifies other administration tasks. On the other hand the LDAP connection information may be specified by a client application.

Typically the simplified LDAP interface manages the session connection to LDAP for the application automatically. The simplified LDAP API will use the pre-configured connection and bind information to setup

and tear down LDAP connections as needed. Generally the simplified LDAP interface will setup and tear down LDAP connections on a per search basis. The application can optionally request to have a connection remain persistent and it can tell the library to unbind and recycle the connection when necessary.

All these interfaces are reentrant with the exception of __ns_ldap_list() with the callback (see __ns_ldap_list() section below). They are safe to used by both single-threaded and multi-threaded applications. Results and errors are stored in dynamically allocated structures. It is caller's responsibility to free the result structure by calling __ns_ldap_freeResult(), __ns_ldap_freeEntry(), __ns_ldap_freeError(), __ns_ldap_freeParam(), __ns_ldap_freeAuth(), __ns_ldap_freeSearchDescriptors(), __ns_ldap_freeAttributeMaps(), __ns_ldap_freeObjectclassMaps().

The new standard based profile documents new schema attributes that were non-existent in the Native LDAP phase 1 profile (solarisNamingProfile). These new attributes define a mapping schema that is stored in the LDAP directory. These new schema values give administrators the ability to re-map rfc2307bis schema into other LDAP schema. By supporting this functionality, administrators are able to map existing directory deployments into rfc2307bis compliant schema as long as the data model and syntaxes are similar. With this feature the simplified LDAP interface is capable of processing requests using non-rfc2307bis complaint directory configurations. This function helps enable support for existing third party directory configurations such as Windows 2000 Active Directory (however validating Native LDAP running against an AD server is not within the scope of this project and will be covered through another project).

The simplified LDAP library can use the LDAP cache manager to cache configuration and connection information for the library. This tool reduces load on the network and the LDAP server by reducing the number of repetitious queries needed to configure the simplified LDAP library when a request is performed. The LDAP cache manager works in conjunction with the name service switch cache manager (nscd) significantly reducing the number of LDAP server queries made by a single system.

The simplified LDAP library also provides client application to specify configuration and connection information explicitly (so-called "standalone" schema). In this case the library is able to perform all operations passing over the LDAP cache manager.

The Simplified LDAP interfaces are delivered as Contract Private interfaces.

## 4.2 Flags

The following is a list of flags used by some of the Simplified LDAP API. The value of the flag can be logically ORed together.

| | |
|---|---|
| NS_LDAP_HARD | specifies that the search operation should continue trying to contact a server until a definitive result is returned. The default is to try each server defined in the active LDAP configuration at least once. |
| NS_LDAP_FOLLOWREF | specifies that the LDAP library should follow the referrals returned by the LDAP servers. The default, if not defined in the configuration, is to follow referrals. |

NS_LDAP_NOREF specifies that the LDAP library should not follow the referrals returned by the LDAP servers. The default, if not defined in the configuration, is to follow referrals.

NS_LDAP_ALL_RES specifies that the interface should return all matching entries including the ones from the alternate search baseDN. The default it only returns the matching entries from the first search baseDN that contains matching entries.

NS_LDAP_SCOPE_BASE confines the scope for search operation to the base object only. The default, is to search one level if NS_LDAP_SEARCH_SCOPE is not defined in the configuration.

NS_LDAP_SCOPE_ONELEVEL
specifies the scope for search operation to only include one level. This is the default flag that will be used if NS_LDAP_SEARCH_SCOPE is not defined in the configuration.

NS_LDAP_SCOPE_SUBTREE
specifies that the scope for search operation to include the entire subtree rooted at and including the base object. This flag should be used with caution due to the fact that any searches with this flag turned on, would walk the entire sub-tree under the base object.

NS_LDAP_NEW_CONN specifies that the call will force a new connection to be created. If a connection currently exists, then the existing connection will be closed and re-opened (refreshed). If a connection does not exist one is created.

NS_LDAP_KEEP_CONN specifies that the connection created for the call should be kept open to be used in future calls to simplified LDAP. Using this flag improves performance at the cost of keeping a connection to the server open. By default connections are not kept open between requests to the API. Applications that wish to keep a connection open for an extended period of time need to consider the resource load on the LDAP server in doing this. An application can close a connection at anytime via __ns_ldap_freeAuth.

NS_LDAP_NOMAP disables schema mapping and translation for this call. This includes any attribute and/or objectclass mapping that may occur as a result of the API call.

NS_LDAP_PAGE_CTRL specifies that a paging control should be used to control the flow of the results back. Depending on the support available from the server either simple page control or virtual list view control will be used to return the results in pages. This is the default for __ns_ldap_*entry() API calls.

NS_LDAP_NO_PAGE_CTRL

specifies that no paging control should be used to control the flow of the results back. Neither simple page control nor virtual list view control will be used when this flag is passed. This is the default for all calls other except for the __ns_ldap_*entry() calls.

**NOTE:** Specifying multiple opposing flags (e.g. no referral and follow referral) is not allowed and will result in an NS_LDAP_INVALID_PARAM error.

## 4.3 Defined Service Names

The following table contains a listing of service names as traditionally supported by various naming services in Solaris. The table also provides a mapping between the service name, service type, data structures, default container, and object classes used to define entries in the directory.

Service type and the C data structure are what passed to __ns_ldap_addTypedEntry() (defined in __ns_ldap_addTypedEntry()). Default container and default objectclass are the location and schema used to create entries in the directory server. These defaults can be overridden using service search descriptor and schema mapping features of the configuration profile.

| Service Name | Service type (See Section 5.4.3, p. 27) | Structure Used in Conversion (See Section 5.4.3, p. 27) | Default Container | Default Objectclass |
|---|---|---|---|---|
| “passwd” | “passwd” | struct passwd | ou=people | posixAccount |
| “passwd” | “shadow” | struct spwd | ou=people | shadowAccount |
| “group” | “group” | struct group | ou=group | posixGroup |
| “netgroups” | “netgroups” | struct _ns_netgroups {<br>char *name;<br>char **triplet;<br>char **netgroup;<br>} | ou=netgroup | nisNetgroup |
| “hosts” | “hosts” | struct hostent | ou=hosts | ipHost |
| “networks | “networks | struct netent | ou=networks | ipNetwork |
| “netmasks” | “netmasks” | struct _ns_netmasks {<br>char *netnumber;<br>char *netmask;<br>} | ou=networks | ipNetwork |
| “rpc” | “rpc” | struct rpcent | ou=rpc | oncRpc |
| “protocols” | “protocols” | struct protoent | ou=protocols | ipProtocol |

| | | | | |
|---|---|---|---|---|
| “services” | “services” | struct servent | ou=services | ipService |
| “bootparams” | “bootparams” | struct _ns_bootp {<br>char *name;<br>char **param;<br>} | ou=ethers | bootableDevice |
| “ethers” | “ethers” | struct _ns_ethers {<br>char *name;<br>char *ether;<br>} | ou=ethers | ieee802Device |
| “publickey” | “publickey” | struct _ns_pubkey {<br>char *name;<br>char *pubkey;<br>char *privkey;<br>} | ou=people<br>ou=hosts | nisKeyObject |
| “aliases” | “aliases” | struct _ns_alias {<br>char *alias;<br>char **member;<br>}; | ou=aliases | mailGroup |
| “project” | “project” | struct project | ou=projects | SolarisProject |
| “auth_attr” | “auth_attr” | struct authattr_t | ou=solarisauthattr | SolarisAuthAttr |
| “prof_attr” | “prof_attr” | struct profattr_t | ou=solarisprofattr | SolarisProfAttr |
| “prof_attr” | “exec_attr” | struct execattr_t | ou=solarisprofattr | SolarisExecAttr |
| “passwd” | “user_attr” | struct user_attr_t | ou=people | SolarisUserAttr |
| “passwd” | “audit_user” | struct au_user_str_t | ou=people | SolarisAuditUser |
| “printers” | “printers” | | ou=printers | sunPrinter |
| “pam”[1] | | | | |

### 4.4 Authentication Structure

The authentication to the server is done via passing a ns_cred_t structure to the Simplified LDAP API. This structure contains both the authentication information as well as the credential required for the method chosen. In most cases, passing a NULL is supported and results in using default authentication method with the default

1 “pam” service is only used to set the authentication method for pam modules. This service does not currently store any data and hence has no schema associated with it.

identity. In this release of the product, only anonymous and proxy identities are supported. Self identity will be supported at a later release. For more information about the security model see Section 8.1 on page 51.

The following table defines the data structure and the corresponding types used to populate authentication information in the ns_cred_t structure to be passed to the Simplified LDAP API. Note that even though CertCred_t structure is defined in this document, certificate based authentication of the client to the server is not supported in this phase. Furthermore, GSSAPI and SPNEGO sasl mechanisms are not supported in this release. And finally, SASL options are not supported as the current implementation of our LDAP C-API does not support them.

```
typedef enum CredLevel {
     NS_LDAP_CRED_ANON        = 0,  /* Anonymous identity */
     NS_LDAP_CRED_PROXY       = 1,  /* Proxy identity */
     NS_LDAP_CRED_SELF        = 2   /* Self identity, not supported */
} CredLevel_t;

typedef enum AuthType {
     NS_LDAP_AUTH_NONE        = 0,  /* No Authentication */
     NS_LDAP_AUTH_SIMPLE,     = 1,  /* UserID & password */
     NS_LDAP_AUTH_SASL,       = 2,  /* SASL mechanism dependent */
     NS_LDAP_AUTH_TLS         = 3,  /* TLS protected method */
} AuthType_t;

typedef enum TlsType {
     NS_LDAP_TLS_NONE,        = 0,  /* no authentication */
     NS_LDAP_TLS_SIMPLE,      = 1,  /* SASL integrity */
     NS_LDAP_TLS_SASL,        = 2   /* SASL privacy */
} SecType_t;

typedef enum SaslMech {
     NS_LDAP_SASL_NONE        = 0,  /* No SASL mechanism */
     NS_LDAP_SASL_CRAM_MD5    = 1,  /* SASL CRAM-MD5 mechanism */
     NS_LDAP_SASL_DIGEST_MD5  = 2,  /* SASL DIGEST-MD5 mechanism */
     NS_LDAP_SASL_EXTERNAL    = 3,  /* SASL EXTERNAL mech., not supported */
     NS_LDAP_SASL_GSSAPI      = 4,  /* SASL GSSAPI mech., not supported */
     NS_LDAP_SASL_SPNEGO      = 5   /* SASL SPNEGO mech., not supported */
} SaslMech_t;

typedef enum SaslOpt {
     NS_LDAP_SASLOPT_NONE,    = 0,  /* no security */
     NS_LDAP_SASLOPT_INT      = 1,  /* SASL integrity, not supported */
     NS_LDAP_SASLOPT_PRIV     = 2   /* SASL privacy, not supported */
} SaslOpt_t;
```

```
typedef struct UnixCred {
     char        *userID;          /* Unix Name or Unix Uid or DN */
     char        *passwd;          /* Unix Password */
} UnixCred_t;

typedef struct CertCred {
     char        *certpath;        /* Certificate path */
     char        *keypath;         /* Key database path */
     char        *passwd;          /* Password */
     char        *nickname;        /* nickname */
} CertCred_t;

typedef struct ns_auth {
     AuthType_t        type;       /* authentication method */
     TlsType_t         tlstype;    /* TLS usage information */
     SaslMech_t        saslmech;   /* SASL mechanism to use */
     SaslOpt_t         saslopt;    /* SASL mechanism option */
} ns_auth_t;

typedef struct ns_cred {
     ns_auth_t         auth;       /* authentication information  */
     char              *certpath;  /* certificate path            */
     union {
           UnixCred_t  unix_cred;  /* unix credential             */
           CertCred_t  cert_cred;  /* X509 certificate credential */
     } cred;
} ns_cred_t;
```

### 4.5 Service Search Descriptor

A service search descriptor describes the characteristics of a search in a given configuration for a given service. The service search descriptor structure defines the combination of a basedn, a scope and a filter used by a service to find a specific type of entry or entries. Service search descriptors are defined as part of the Native LDAP configuration. One or more service search descriptors may exist for any given service (database). The following structure defines an instance of a service search descriptor:

```
/* service search descriptor */
typedef struct ns_ldap_search_desc {
     char        *basedn;          /* search base dn */
     ScopeType_t scope;            /* search scope */
     char        *filter;          /* search filter */
} ns_ldap_search_desc_t;
```

### 4.6 Attribute Mapping

An attribute mapping describes the attribute mapping relationship for a given service. This structure defines the combination of original attribute and the mapped attributes. Since it is possible to map one attribute to many, the mappedAttr is a NULL terminated array of mapped attributes. Attribute mappings are defined as part of the Native LDAP configuration. The following structure defines an instance of an attribute mapping:

```
/* attribute mapping */
typedef struct ns_ldap_attribute_map {
    char        *origAttr;        /* original attribute */
    char        **mappedAttr;     /* mapped attribute(s) */
} ns_ldap_attribute_map_t;
```

### 4.7 Objectclass Mapping

An objectclass mapping describes the objectclass mapping relationship for a given service. This structure defines the combination of original objectclass and the mapped objectclass. Objectclass mappings are defined as part of the Native LDAP configuration. The following structure defines an instance of an objectclass mapping:

```
/* objectclass mapping */
typedef struct ns_ldap_objectclass_map {
    char        *origOC;          /* original objectclass */
    char        *mappedOC;        /* mapped objectclass */
} ns_ldap_objectclass_map_t;
```

### 4.8 Return Code

The return code for the Simplified LDAP API are defined in this section. Depending on the return code, extra information may be available through the ns_ldap_error_t structure which is returned in the errorp parameter defined in the Simplified LDAP

API. In the case of a successful __ns_ldap_*() call, the errorp will be set to NULL.

The ns_ldap_error_t structure is defined as the following:

```
/* the ns_ldap_error_t to return detailed error information */
typedef struct ns_ldap_error
    int         status;           /* specific error code */
    char        *message;         /* error messages */
} ns_ldap_error_t;
```

The return codes for the Simplified LDAP APIs are:

| | |
|---|---|
| NS_LDAP_SUCCESS | successful operation. No extra information will be returned in the errorp. |
| NS_LDAP_OP_FAILED | failed operation. No extra information will be returned in the errorp. |
| NS_LDAP_INVALID_PARAM | bad parameter passed to function. No extra information will be returned in the errorp. |
| NS_LDAP_NOTFOUND | entry not found. No extra information will be returned in the errorp. |
| NS_LDAP_MEMORY | internal memory allocation error. No extra information will be returned in the errorp. |
| NS_LDAP_CONFIG | LDAP configuration problem. errorp will contain detailed error information.<br>NS_CONFIG_SYNTAX - syntax error;<br>NS_CONFIG_NODEFAULT - no default value;<br>NS_CONFIG_NOTLOADED - configuration not loaded;<br>NS_CONFIG_NOTALLOW - operation requested not allowed;<br>NS_CONFIG_FILE – active LDAP back-end configuration problem;<br>NS_CONFIG_CACHEMGR - error with ldap_cachemgr door. |
| NS_LDAP_PARTIAL | partial result returned. errorp will contain detailed error information.<br>NS_PARTIAL_TIMEOUT - unable to retrieve all results when NS_LDAP_ALL_RES flag is set;<br>NS_PARTIAL_OTHER - error encountered while retrieving all results. |
| NS_LDAP_INTERNAL | internal LDAP error encountered during the operation. errorp will contain detailed error information. The status field from the ns_ldap_error_t structure will contain the standard LDAP error code as defined in the ldap.h. The message field will contain the error string as returned by the ldap_err2string() for the error code defined in the status field. If exists, the message from the LDAP server as returned by ldap_get_option(LDAP_OPT_ERROR_STRING) will be appended to this message field. |

**NOTE:** The caller is responsible to free the errorp structured returned by the Simplified LDAP API.

### 4.9 Result Structure

The following shows the structure used by various Simplified LDAP APIs to store LDAP entries.

```
typedef struct ns_ldap_attr {
    char        *attrname;        /* attribute name */
    uint_t      value_count;      /* number of values */
    char        **attrvalue;      /* array of attribute values */
} ns_ldap_attr_t;

typedef struct ns_ldap_entry  {
    uint_t            attr_count;       /* number of attributes */
    ns_ldap_attr_t    **attr_pair;      /* array of attribute pairs */
    struct ns_ldap_entry *next;         /* next entry pointer */
} ns_ldap_entry_t;

typedef struct ns_ldap_result  {
    uint_t            entries_count;    /* number of entries */
    ns_ldap_entry_t   *entry;           /* link list of entry data */
} ns_ldap_result_t;
```

### 4.10 “Standalone” configuration data.

The following structures contain configuration and connection information describing a directory server specified by a client application.

```
/*
 * The type of standalone configuration specified be a client application
 * The meaning of the requests is as follows:
 *
 * NS_CACHEMGR:    libsldap will request all the configuration
 *                 via door_call(3C)
 *                 to ldap_cachemgr.
 * NSBEC:          the name of a local back-end configuration is specifie
 * NS_LDAP_SERVER: the consumer application has specified a directory ser
 *                 to communicate to.
 * NS_PREDEFINED:  reserved for internal use
 */
typedef enum {
    NS_CACHEMGR = 0,
    NS_BEC,
    NS_LDAP_SERVER,
    NS_PREDEFINED
} ns_standalone_request_type_t;

/*
 * This structure describes an LDAP server specified by a client applicat
 */
typedef struct ns_dir_server {
    char *server;          /* A directory server's IP */
    uint16_t port;         /* A directory server's port. */
                           /* Default value is 389 */
    char *domainName;      /* A domain name being served */
                           /* by the specified server. */
                           /* Default value is the local */
                           /* domain's name */
    char *profileName;     /* A DUAProfile's name. */
                           /* Default value is 'default' */
    ns_auth_t *auth;       /* Authentication information used */
                           /* during subsequent connections */
    char *cred;            /* A credential level to be used */
                           /* along with the authentication info */
    char *host_cert_path;  /* A path to the certificate database */
                           /* Default is '/vat/ldap' */
    char *bind_dn;         /* A bind DN to be used during */
                           /* subsequent LDAP Bind requests */
    char *bind_passwd;     /* A bind password to be used during */
                           /* subsequent LDAP Bind requests */
} ns_dir_server_t;
```

```
/*
 * This structure contains either a name of the NSBEC or
 * the information describing an LDAP server
 */
typedef struct ns_standalone_conf {
    union {
          ns_dir_server_t server;
          char *nsbec_name; /* NSBEC's name */
          void *predefined_conf; /* Reserved for internal use */
    } ds_profile;     /* A type of the configuration */

#define SA_NSBEC ds_profile.nsbec_name

#define SA_SERVER ds_profile.server.server
#define SA_PORT   ds_profile.server.port
#define SA_DOMAIN ds_profile.server.domainName
#define SA_PROFILE_NAME ds_profile.server.profileName
#define SA_AUTH   ds_profile.server.auth
#define SA_CRED   ds_profile.server.cred
#define SA_CERT_PATH ds_profile.server.host_cert_path
#define SA_BIND_DN ds_profile.server.bind_dn
#define SA_BIND_PWD ds_profile.server.bind_passwd

    ns_standalone_request_type_t type;
} ns_standalone_conf_t;
```

# 5.0 Simplified LDAP API

## 5.1 “Standalone” configuration

### 5.1.1 __ns_ldap_initStandalone()

**Synopsis:**

```
ns_ldap_return_code __ns_ldap_initStandalone(
            const ns_standalone_conf_t conf,
            ns_ldap_error_t **errorp);
```

**Description:** This function is used to initialize the library to perform search/update operations bypassing ldap_cachemgr(1M).

The conf parameter describes a directory server specified by the client process. See Section 4.10 on page 15 for the description of the ns_standalone_conf_t structure fields.

If any error occurs during the library initialization phase, the function returns a pointer to an object describing the error via the errorp parameter.

A client application is supposed to call this function if the LDAP client is not initialized yet or the application is going to connect to an LDAP server different from the one specified in the active NSBEC.

The behavior of the function depends on what type of configuration was provided by the conf parameter. If the user process specifies an NSBEC name (ns_standalone_conf_t::type is set to NS_BEC), the function obtains the NSBEC using libnsconf and treats it as a DUAProfile. Otherwise (ns_standalone_conf_t::type is set to NS_LDAP_SERVER), if the process provides an LDAP server's address, the function establishes a connection to the LDAP server and requests for an appropriate DUAProfile. Then __ns_ldap_initStandalone() connects to all the servers specified in the default and preferred server lists and obtains their Root DSE. All the obtained information is used during all following operations instead of the information received from ldap_cachemgr(1M) via door calls.

The special case when ns_standalone_conf_t::type is set to NS_PREDEFINED is reserved for the internal libsldap's purpose.

In case when the caller sets ns_standalone_conf_t::type to NS_CACHEMGR, libsldap will obtain all the configuration information via door_call(3C) to ldap_cachemgr(1M).

#### 5.1.2 __ns_ldap_getConnectionInfo()

**Synopsis:**

```
ns_ldap_return_code __ns_ldap_getConnectionInfo(
      const ns_dir_server_t *server,
      const ns_cred_t *cred,
      char **config,
      char **baseDN,
      ns_ldap_error_t **error);
```

**Descriptions:** This function obtains a base DN and a DUAProfile from a specified directory server.

The server parameter specifies a selected directory sever.

If the baseDN parameter is not NULL, the function looks for the appropriate base DN on the specified server and returns an address of the string representing the base DN.

If the config parameter is not NULL, the function requests the DUAProfile specified in the server structure and returns the address of a string containing the profile in the ldap_cachemgr(1M) door call format.

The cred structure contains authentication information and credential required to establish a connection.

If an error occurs, the error parameter returns information about the error.

#### 5.1.3 __ns_ldap_getRootDSE()

**Synopsis:**

```
ns_ldap_return_code __ns_ldap_getRootDSE(
      const char *server_addr,
      char **rootDSE,
      ns_ldap_error_t **error,
      int anon_fallback);
```

**Description:** This function obtains the root DSE from a specified LDAP server.

The server parameter specifies an address of a server to be connected to.

The function establishes a connection to the server using the current libsldap configuration and requests the root DSE from the server. The obtained data is returned as a string in the ldap_cachemgr(1M) door call format.

If an error occurs, the error parameter returns an information about the error.

If the anon_fallback parameter is set to 1 and establishing a connection fails, __s_api_getRootDSE() will try once again using anonymous credentials.

#### 5.1.4 __ns_ldap_pingOfflineServers()

**Synopsis:**

```
ns_ldap_return_code __ns_ldap_pingOfflineServers(void);
```

**Description:** This function iterates through the list of the configured LDAP servers and "pings" those which are marked as removed or if any error occurred during the previous receiving of the server's root DSE. If the function is able to reach such a server and get its root DSE, it marks the server as on-line. Otherwise, the server's status is set to "Error".

For each server the function tries to connect to, it fires up a separate thread and then waits until all the treads finish.

The function returns NS_LDAP_INTERNAL if the Standalone mode was not initialized or was canceled prior to an invocation of __ns_ldap_pingOfflineServers().

#### 5.1.5 __ns_ldap_cancelStandalone()

**Synopsis:**

```
void _ns_ldap_initStandalone(void);
```

**Description:** This function cancels the Standalone mode and destroys all the data related to the Standalone mode.

### 5.2 Search APIs

#### 5.2.1 __ns_ldap_list()

**Synopsis:**

```
int __ns_ldap_list(
            const char *service,
            const char *filter,
            int (*init_filter_cb)(const
                    ns_ldap_search_desc_t *desc,
                    char **realfilter,
                    const void *userdata),
            const char * const *attribute,
            const ns_cred_t *cred,
            const int flags,
            ns_ldap_result_t **result,
            ns_ldap_error_t **errorp,
            int (*callback)(const ns_ldap_entry_t
            *entry,ns_ldap_error_t **errorp,
```

```
                const void *userdata),
        const void *userdata);
```

**Description:** This API is used to search for entries in the LDAP name space.

The service parameter specifies the service name from which the data should be retrieved. The common services supported are listed in the nsswitch.conf(4) man page. The service names are listed in Section 4.3 on page 7. This API will use the provided service name to help determine and generate the LDAP baseDN for the search operation. This API may perform multiple LDAP search operations, as determined by the current profile configuration, in order to complete the specified request.

The filter parameter specifies the criteria to use during the search to determine which entries to return. This parameter consists of a boolean combination of attribute-value assertions, where each attribute-value assertion consists of two parts: an attribute and a value. The value field could contain wildcards. For further information about LDAP filters, please refer to RFC2251.

If an init_filter_cb callback is provided by the calling program then this routine will be called by the list routine prior to an LDAP search. The callback will be provided with a service search descriptor and the userdata argument passed into __ns_ldap_list. Using this data the callback can create a search filter to be used by the list function for the provided service search descriptor. The callback should return a LDAP search filter through realfilter argument. If the callback succeeds it should return NS_LDAP_SUCCESS. Any other return code will be considered an error and the realfilter argument will be ignored. The filter created by the filter callback should be created using malloc(3), calloc(3) or realloc(3) library routines. The __ns_ldap_list function will call free(3) on the realfilter data pointer when it has finished using it. If no filter callback is provided or if the filter call back does not return NS_LDAP_SUCCESS then the __ns_ldap_list routine will automatically create a search filter using the available profile data.

The attribute parameter specifies the attributes to return along with each entry matching the search. It is a NULL terminated array of attribute strings specifying the names of the attributes you want to return. Passing NULL attribute argument means that all available attributes should be returned.

The cred parameter specifies the authentication method to be used along with credential information. When passing a NULL cred pointer, the default authentication method defined in the active LDAP back-end configuration will be used.

The flags parameter defines how the function will respond to various conditions. Supported flags are:

NS_LDAP_HARD
NS_LDAP_FOLLOWREF
NS_LDAP_NOREF
NS_LDAP_ALL_RES
NS_LDAP_KEEP_CONN
NS_LDAP_PAGE_CTRL
NS_LDAP_NOMAP

If the list operation is successful, the result will be returned in the result parameter. If the operation is unsuccessful, then the result parameter will be NULL and the error parameter may contain detailed error information (see Return Code).

The callback parameter is an optional parameter. It's a pointer to a function that will process the entry that is returned from the search. If this pointer is NULL, then all the entries that match the search criteria are returned in the ns_ldap_result_t structure, otherwise this function will be called once for each entry returned. When called, this function should return NS_LDAP_CB_NEXT when additional entries are desired and NS_LDAP_CB_DONE when it no longer wishes to see any more entries. The last parameter, userdata, is simply passed to the callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.

**NOTE 1:** The callback routine must not free the entries passed to it. The __ns_ldap_list() internal will free the resources allocated for the entries.

**NOTE 2:** __ns_ldap_list() is not MT-Safe with callbacks. In other words, you cannot call any of the Simplified LDAP routines within the callback routine.

**Returns:** this function returns the error code as specified in Return Code.

#### 5.2.2 __ns_ldap_firstEntry(), __ns_ldap_nextEntry() and __ns_ldap_endEntry()

**Synopsis:**

```
int __ns_ldap_firstEntry(
             const char *service,
             const char *filter,
             int (*init_filter_cb)(const
                      ns_ldap_search_desc_t *desc,
                      char **realfilter,
                      const void *userdata),
             const char * const *attribute,
             const ns_cred_t cred,
             const int flags,
             void **cookie,
             ns_ldap_result_t **result,
             ns_ldap_error_t **errorp,
             const void *userdata);

int __ns_ldap_nextEntry(
             void         *cookie,
             ns_ldap_result_t **result,
             ns_ldap_error_t **errorp);

int __ns_ldap_endEntry(
             void         **cookie,
             ns_ldap_error_t **errorp);
```

**Description:** The __ns_ldap_firstEntry() and __ns_ldap_nextEntry() are used to enumerate one entry at a time. The __ns_ldap_endEntry() is used to free all the resources allocated on the client side for the enumeration.

__ns_ldap_firstEntry() takes a few extra parameters: service, filter, attribute, cred and flags.

The service parameter specifies the service name from which the data should be retrieved. The common services supported are listed in the nsswitch.conf(4) man page. The service names are listed in Section 4.3 on page 7. This API will use the provided service name to help determine and generate the LDAP baseDN for the search operation. This API may perform multiple LDAP search operations, as determined by the current profile configuration, in order to complete the specified request.

The filter parameter specifies the criteria to use during the search to determine which entries to return. This parameter consists of a boolean combination of attribute-value assertions, where each attribute-value assertion consists of two parts: an attribute and a value. The value field could contain wildcards. For further information about LDAP filters, please refer to RFC2251.

If an init_filter_cb callback is provided by the calling program then this routine will be called by the list routine prior to an LDAP search. The callback will be provided with a service search descriptor and the userdata argument passed into __ns_ldap_firstEntry. Using this data the callback can create a search filter to be used by the search function for the provided service search descriptor. The callback should return a LDAP search filter through realfilter argument. If the callback succeeds it should return NS_LDAP_SUCCESS. Any other return code will be considered an error and the realfilter argument will be ignored. The filter created by the filter callback should be created using malloc(3), calloc(3) or realloc(3) library routines. The __ns_ldap_firstEntry function will call free(3) on the realfilter data pointer when it has finished using it. If no filter callback is provided or if the filter call back does not return NS_LDAP_SUCCESS then the __ns_ldap_firstEntry routine will automatically create a search filter using the available profile data.

The attribute parameter specifies the attributes to return along with each entry matching the search. It is a NULL terminated array of attribute strings specifying the names of the attributes you want to return. Passing NULL attribute argument means that all available attributes should be returned.

The cred parameter specifies the authentication method to be used along with credential information. When passing a NULL cred pointer, the default authentication method defined in the active LDAP back-end configuration will be used.

The flags parameter defines how the function will respond to various conditions. Supported flags are:

NS_LDAP_HARD
NS_LDAP_FOLLOWREF
NS_LDAP_NOREF
NS_LDAP_KEEP_CONN
NS_LDAP_NO_PAGE_CTRL
NS_LDAP_NOMAP

By default, __ns_ldap_firstEntry() and __ns_ldap_nextEntry() use a page control to control the flow of

information from the server. Using NS_LDAP_NO_PAGE_CTRL applications can eliminate use of the paging controls. However caution must be taken as if the number of entries matching the search criteria exceeds the server imposed limit, only a subset of results might be returned.

For the __ns_ldap_firstEntry() and __ns_ldap_nextEntry(), if the operation is successful, the result will be returned in the result parameter. If the operation is unsuccessful, then the result parameter will be NULL and the error parameter may contain detailed error information (see Return Code). __ns_ldap_firstEntry() also returns a cookie in the cookie parameter to the caller to be used for subsequent __ns_ldap_nextEntry() calls. This cookie contains internal information and thus it should treated as an opaque data. It should only be freed by calling the __ns_ldap_endEntry().

__ns_ldap_nextEntry() takes the cookie parameter returned by __ns_ldap_firstEntry(). The order in which the entries are returned is not guaranteed. Further, should an update occur on the server between client calls, there is no guarantee that the entry added or modified will be seen by the client.

__ns_ldap_endEntry() is used free all internal resources allocated for the enumeration including the cookie. It also frees the cookie resource and resets it to NULL.

**NOTE:** __ns_ldap_endEntry() takes the address of the pointer for the cookie.

**Returns:** these functions return the error code as specified in Return Code.

#### 5.2.3 __ns_ldap_uid2dn() and __ns_ldap_host2dn()

**Synopsis:**

```
int __ns_ldap_uid2dn(
            const char *uid,
            char **userDN,
            const ns_cred_t *cred,
            ns_ldap_error_t **errorp);

int __ns_ldap_host2dn(
            const char *host,
            const char *domain,
            char **hostDN,
            const ns_cred_t *cred,
            ns_ldap_error_t **errorp);
```

**Description:** These functions map an UNIX user identity and a host to an LDAP distinguished name in the client's domain, if domain is set to NULL. Otherwise, it uses the supplied domain. The uid parameter can either be an UNIX user name or user ID number in string format. For example, "user1" or "100". The host parameter is a hostname. For mapping of other UNIX identities to LDAP distinguished names, __ns_ldap_list() function should be used.

The derived user and host DN will be returned in the userDN and hostDN parameter respectively. If the uid or

the host specified map into more than one LDAP DN, then the function will fail and the NS_LDAP_OP_FAILED error code will be returned.

**NOTE:** It's the caller's responsibility to free the returned memory in userDN or hostDN.

The cred parameter contains the authentication information for the LDAP connection. If this parameter is NULL, then the information defined in the active LDAP back-end configuration will be used.

The errorp parameter may contain detailed error information in case the operation fails.

**Returns:** This function returns the error code as specified in Return Code.

#### 5.2.4 __ns_ldap_dn2domain()

**Synopsis:**

```
int __ns_ldap_dn2domain(
            const char *dn,
            char **domain,
            const ns_cred_t *cred,
            ns_ldap_error_t **errorp);
```

**Description:** This function maps a DN into a domain name based on the value of the nisDomain attribute in the DIT.

If the dn mapping into a domain is not successful, then the function will fail and the NS_LDAP_OP_FAILED error code will be returned.

This function will attempt to map any DN to a domain name. However since this information is cached in ldap_cachemgr (1M), only RDNs should be used as input to this function. This ensures a smaller cache, and hence better performance.

**NOTE:** It's the caller's responsibility to free the returned memory in domain.

The cred parameter contains the authentication information for the LDAP connection. If this parameter is NULL, then the information defined in the active LDAP back-end configuration will be used.

The errorp parameter may contain detailed error information in case the operation fails.

**Returns:** This function returns the error code as specified in Return Code.

## 5.3 Authentication API

### 5.3.1 __ns_ldap_auth()

**Synopsis:**

```
int __ns_ldap_auth(
              const ns_cred_t *cred,
              const int flag,
              ns_ldap_error_t **errorp,
              LDAPControl **serverctrls,
              LDAPControl **clientctrls);
```

**Description:** This function is used to authenticate users. The cred parameter contains the authentication information for the LDAP connection.

The flags parameter defines how the function will respond to various conditions. The only supported flag is:

NS_LDAP_KEEP_CONN

The errorp parameter may contain detailed error information in case the operation fails.

The `serverctrls` and `clientctrls` parameters are optional and should be set to NULL by default. If the application decides to use an LDAP control it is the responsibility of the application to create and manage the controls within the application framework.

**Returns:** This function returns NS_LDAP_SUCCES if the authentication is successful. It returns NS_LDAP_OP_FAILED if authentication fails. Any other failure, it returns the error code as specified in Return Code.

## 5.4 Add/Modify/Delete APIs

### 5.4.1 __ns_ldap_addEntry() and __ns_ldap_delEntry()

**Synopsis:**

```
int __ns_ldap_addEntry(
              const char *service,
              const char *dn,
              const ns_ldap_entry_t *entry,
              const ns_cred_t *cred,
              const int flags,
              ns_ldap_error_t **errorp);

int __ns_ldap_delEntry(
              const char *service,
              const char *dn,
              const ns_cred_t *cred,
```

```
                const int flags,
                ns_ldap_error_t **errorp);
```

**Description:** These APIs are used to add and delete an entry to and from the LDAP namespace.

The service parameter specifies the service name associated with this data. The common services supported are listed in the nsswitch.conf(4) man page. The service names are listed in Section 4.3 on page 7. This API will use the provided service name to help perform any necessary attribute mapping changes that may be required.

The dn parameter defines the distinguish name of the entry to be created or deleted.

The cred parameter specifies the authentication method to be used along with credential information. You must specify this parameter in order to make any change to the LDAP directory service.

**NOTE:** Before deleting any entry, you must find the "dn" for the entry using __ns_ldap_list(), __ns_ldap_first(), or __ns_ldap_next(). The "dn" value will be returned as one of the attribute in the result structure from these list APIs. It can then be used to specify the "dn" parameter for the __ns_ldap_*Entry() routines**.**

__ns_ldap_addEntry() also takes the entry parameter which specifies the attributes of the new entry named by the dn parameter.

The flags parameter defines how the function will respond to various conditions. Supported flags are:

NS_LDAP_FOLLOWREF
NS_LDAP_NOREF
NS_LDAP_KEEP_CON
NS_LDAP_NOMAP

**Returns:** these functions return the error code as specified in Return Code. In case of unsuccessful operation, the error parameter may contain detailed error information (see Return Code).

#### 5.4.2 __ns_ldap_addAttr(), __ns_ldap_delAttr() and __ns_ldap_repAttr()

**Synopsis:**

```
int __ns_ldap_addAttr(
                const char *service,
                const char *dn,
                const ns_ldap_attr_t * const *attr,
                const ns_cred_t *cred,
                const int flags,
                ns_ldap_error_t **errorp);

int __ns_ldap_delAttr(
                const char *service,
```

```
                const char *dn,
                const ns_ldap_attr_t * const *attr,
                const ns_cred_t *cred,
                const int flags,
                ns_ldap_error_t **errorp);

int __ns_ldap_repAttr(
                const char *service,
                const char *dn,
                const ns_ldap_attr_t * const *attr,
                const ns_cred_t *cred,
                const int flags,
                ns_ldap_error_t **errorp);
```

**Description:** These APIs are used to add, delete and replace attributes for an entry from the LDAP namespace.

The service parameter specifies the service name associated with this data. The common services supported are listed in the nsswitch.conf(4) man page. The service names are listed in Section 4.3 on page 7. This API will use the provided service name to help perform any necessary attribute mapping changes that may be required.

The dn parameter specifies the distinguished name for the entry whose content are to be modified.

**NOTE:** Before modifying any attribute, you must find the “dn” for the entry using __ns_ldap_list(), __ns_ldap_first(), or __ns_ldap_next(). The “dn” value will be returned as one of the attribute in the result structure from these list APIs. It can then be used to specify the “dn” parameter for the __ns_ldap_*Attr() routines**.**

The attr parameter is an NULL terminated array of ns_ldap_attr_t elements (see the Result Structure section on page 12 for more detail on the ns_ldap_attr_t). The cred parameter specifies the authentication method to be used along with credential information. You must specify this parameter in order to make any change to the LDAP directory service.

The flags parameter defines how the function will respond to various conditions. The supported flags are:

NS_LDAP_FOLLOWREF
NS_LDAP_NOREF
NS_LDAP_KEEP_CONN
NS_LDAP_NOMAP

In case of unsuccessful operation, the error parameter may contain detailed error information (see Return Code).

__ns_ldap_addAttr() adds the values contained in the attr_value to the attribute given in the attr_name. If the attribute does not already exist within the entry, it will be created. If any of the values supplied already exist in the entry, an error will be returned to the caller and none of the modifications will have been performed.

__ns_ldap_delAttr() deletes the values contained in the attr_value from the attribute given in the attr_name. If all attribute's values are removed in this way, the attribute itself will be deleted. If one or more of the attribute values specified does not exist in the attribute, an error will be returned to the caller and none of the modification will have been performed. If the attr_value parameter is NULL, the entire attribute will be deleted from the entry.

__ns_ldap_repAttr() removes all existing values of the attribute given in the attr_name parameter and replaces them by the values contained in the attr_value. If the given attribute does not already exist in the entry, it is created. If the attr_value is NULL, this function acts like a delete operation, and the entire attribute is deleted from the entry. A replace with NULL values will always succeed, while a delete may fail if the attribute does not exist.

**Returns:** these functions return the error code as specified in Return Code.

#### 5.4.3 __ns_ldap_addTypedEntry()

**Synopsis:**

```
int __ns_ldap_addTypedEntry(
            const char *servicetype,
            const char *basedn,
            const void *data,
            const int create,
            const ns_cred_t *cred,
            const int flags,
            ns_ldap_error_t **errorp);
```

**Description:** This routine provides a simplified add function for many of the well known services used by this API. This function first creates an simplified LDAP entry structure given a service type and a pointer to the specified data structure. This routine translates UNIX data structures into simplified LDAP attribute/value records based on the current attribute mapping definitions defined by the client's LDAP naming profile. Next this function generates a full distinguished name (DN) using the given arguments. This routine constructs the DN using components from the data structure and type passed down and the basedn (if provided) and data from the LDAP naming profile. Finally this operation performs an __ns_ldap_addEntry() operation and adds the created record into the directory.

basedn - The value used here (if provided) helps to create the DN for the object.

If this value is NULL the routine uses the default base dn to construct an object DN or use the first base dn in the service search descriptor defined in the profile for the given service.

If a base DN value is provided, it is used in conjunction with the appropriate ID from the data structure pointed to create a dn for the entry.

servicetype - The servicetype parameter specifies the service type associated with this data. The common server types are listed in Section 4.3 on page 7. This API will use the provided service types to translate from data to

LDAP schema and perform schema mapping if required.

data - void * pointer to the data structure that contains the type specific data structure. The common structures are listed in Section 4.3 on page 7.

create - if this argument is set to any positive value, the routine will attempt to create an entry, otherwise it will attempt to modify an already created entry in the directory.

The remaining arguments are defined and passed through to __ns_ldap_addEntry(). Their definitions can be found elsewhere in this document.

The flags parameter defines how the function will respond to various conditions. The supported flags are:

NS_LDAP_FOLLOWREF
NS_LDAP_NOREF
NS_LDAP_KEEP_CONN
NS_LDAP_NOMAP

**Returns:** This function returns the error code as specified in Return Code.

## 5.5 LDAP Configuration APIs

### 5.5.1 __ns_ldap_setParam() and __ns_ldap_getParam()

**Synopsis:**

```
int __ns_ldap_setParam(
             const ParamIndexType type,
             const void *data,
             ns_ldap_error_t **errorp);

int __ns_ldap_getParam(
             const ParamIndexType type,
             void ***data,
             ns_ldap_error_t **errorp);
```

**Description:** These functions are used to set and retrieve internal LDAP configuration parameters. For more information on the definition of type and data see Section 6.2 on page 38.

Parameters set by the __ns_ldap_setParam() function will apply to all subsequent Simplified LDAP API calls made from the same thread that changed the information. If this function is called from the main thread, all threads created by this main thread after calling __ns_ldap_setParam() will inherit these information.

__ns_ldap_setParam() can also be used to reset the value to the default value by passing a NULL as the data parameter. This will reset the value back to what was originally defined in the cache file if defined, or to the

default value as defined by each option defined below.

type - The ParamIndexType as defined in Configuration Index Type. The data types are also described in detail in the same section.

**Returns:** these functions return the error code as specified in Return Code.

## 5.6 Attribute/Objectclass mapping APIs

### 5.6.1 __ns_ldap_getSearchDescriptors()

**Synopsis:**

```
int __ns_ldap_getSearchDescriptors(
            const char *service,
            ns_ldap_search_desc_t ***desc,
            ns_ldap_error_t **errorp);
```

**Description:** This function returns a NULL terminated array of service search descriptors as defined by the service name in the client's LDAP naming profile. If multiple service search descriptors are found then they will be returned in the order that they are found.

service - The service parameter specifies the service name associated with this descriptor. The common services supported are listed in the nsswitch.conf(4) man page. The service names are listed in Section 4.3 on page 7.

desc - NULL terminated array of service search descriptors as defined by the current client profile.

**Returns:** This function returns the error code as specified in Return Code.

### 5.6.2 __ns_ldap_getAttributeMaps()

**Synopsis:**

```
int __ns_ldap_getAttributeMaps(
            const char *service,
            ns_ldap_attribute_map_t ***maps,
            ns_ldap_error_t **errorp);
```

**Description:** This function returns a NULL terminated array of attribute maps as defined in the Directory User Agent (DUA) profile. All attribute mappings for service will be returned in the array.

service - The service parameter specifies the service name associated with this mapping. The common services supported are listed in the nsswitch.conf(4) man page. The service names are listed in Section 4.3 on page 7.

**Returns:** This function returns the error code as specified in Return Code.

#### 5.6.3 __ns_ldap_getObjectClassMaps()

**Synopsis:**

```
int __ns_ldap_getObjectClassMaps(
            const char *service,
            ns_ldap_objectclass_map_t ***maps,
            ns_ldap_error_t **errorp);
```

**Description:** This function returns a NULL terminated array of object class maps as defined in the DUA profile. All object class mappings for service will be returned in the array.

service - The service parameter specifies the service name associated with this mapping. The common services supported are listed in the nsswitch.conf(4) man page. The service names are listed in Section 4.3 on page 7.

**Returns:** This function returns the error code as specified in Return Code.

#### 5.6.4 __ns_ldap_getMappedAttributes()

**Synopsis:**

```
char **__ns_ldap_getMappedAttributes(
            const char *service,
            char *origAttribute);
```

**Description:** This function returns a NULL terminated array of one or more attributes that is presently mapped to the specified attribute. This routine performs the attribute mapping conversion and returns the actual attribute name that is currently used to map the specified attribute into the LDAP server. If the specified attribute maps to more than 1 attribute, as may be the case with gecos, then the array will contain an ordered list of attributes used in the mapping. In the 1 to many mapping scenario, attribute values are assumed to be concatenated in order to create the one attribute. If these attributes are identical the original attribute is returned. If the value of the origAttribute is a string OID then this routine returns the mapped attribute or the attribute name for the OID as is appropriate.

**Returns:** This function returns the name of a mapped attribute or itself.

#### 5.6.5 __ns_ldap_getOrigAttribute()

**Synopsis:**

```
char **__ns_ldap_getOrigAttribute(
            const char *service,
            const char *mappedAttribute);
```

**Description:** This function returns a NULL terminated array of one or more attributes that the specified attribute presently maps to. This routine performs the reverse attribute mapping conversion. If the specified attribute maps to more than 1 attribute, as may be the case with gecos, then the array will contain an ordered list of attributes used in the mapping. In the 1 to many mapping scenario, attribute values are assumed to be

concatenated in order to create the one attribute. If these attributes are identical the original attribute is returned. If the value of the mappedAttribute is a string OID then this routine returns the mapped attribute or the attribute name only if a mapping for the OID exists.

**Returns:** this function returns the name of a mapped attribute or itself. This routine will return NULL if no reverse mapping can be found.

#### 5.6.6 __ns_ldap_getMappedObjectClass()

**Synopsis:**

```
char **__ns_ldap_getMappedObjectClass(
            const char *service,
            char *origObjectClass);
```

**Description:** This function returns a NULL terminated array of one or more object classes that is presently mapped to the specified objectclass. This routine performs the attribute mapping conversion and returns the actual objectclass names that are currently used to map the specified objectclass into the LDAP server. If the mapped objectclass is identical the original objectclass is returned. If the value of the origObjectClass is a string OID then this routine returns the mapped object classes for the OID as is appropriate.

**Returns:** this function returns the list of a mapped object classes or itself.

#### 5.6.7 __ns_ldap_getOrigObjectClass()

**Synopsis:**

```
char **__ns_ldap_getOrigObjectClass(
            const char *service,
            const char *mappedObjectClass);
```

**Description:** This function performs the reverse objectclass mapping and returns a NULL terminated array of one or more object classes that the specified objectclass is mapped to. This routine performs the attribute mapping conversion and returns the original objectclass names that were used to map the specified objectclass into the LDAP server. If the mapped objectclass is identical the original objectclass is returned. If the value of the mappedObjectClass is a string OID then this routine returns the mapped object classes for the OID if an objectclass mapping exists.

**Returns:** this function returns the name of a mapped attribute or itself. This routine returns NULL if no reverse mapping can be found.

### 5.7 Utility APIs

#### 5.7.1 __ns_ldap_getServiceAuthMethods()

**Synopsis:**

```
int __ns_ldap_getServiceAuthMethods(
```

```
                const char *service,
                ns_auth_t ***auth,
                ns_ldap_error_t **errorp);
```

**Description:** This function returns a NULL terminated array of service authentication methods as defined by the service name in the client's LDAP naming profile. If multiple service authentication methods are found then they will be returned in the order that they are found.

service - The service parameter specifies the service name associated with the authentication methods. The common services supported are listed in the nsswitch.conf(4) man page. The service names are listed in in Section 4.3 on page 7.

desc - NULL terminated array of service authentication methods as defined by the current client profile.

**Returns:** This function returns the error code as specified in Return Code.

#### 5.7.2 __ns_ldap_getAttr()

**Synopsis:**

```
char **__ns_ldap_getAttr(
            const ns_ldap_entry_t *entry,
            const char *attrname);
```

**Description:** This function is used to retrieve the attribute values for the attribute type specified in the attrname parameter from the entry specified by the entry parameter. The attribute type defined is case insensitive.

**Returns:** it returns a pointer to a static array of attribute value strings. It returns NULL if none exists. Caller must not free this pointer.

#### 5.7.3 __ns_ldap_err2str()

**Synopsis:**

```
char *__ns_ldap_err2str(
            int err);
```

**Description:** This function retrieves the error message string as defined in Return Code that corresponds to the error code returned.

**Returns:** it returns a pointer to a static string. Caller must not free this string pointer.

#### 5.7.4 __ns_ldap_initAuth()

**Synopsis:**

```
__ns_ldap_initAuth(const char auth_mech,
      ns_auth_t *auth,
      ns_ldap_error_t **errorp);
```

**Description:** This function initializes an ns_auth_t structure provided by a caller according to a specified authentication mechanism.

#### 5.7.5 __ns_ldap_freeResult()

**Synopsis:**

```
     int __ns_ldap_freeResult(
                  ns_ldap_result_t **result);
```

**Description:** This function frees memory resources.

**NOTE**: This routine takes the address of the pointer for the result structure.

__ns_ldap_freeResult() frees the ns_ldap_result_t structure allocated by the following functions:

- __ns_ldap_list()
- __ns_ldap_firstEntry()
- __ns_ldap_nextEntry()

**Returns:** This function returns NS_LDAP_SUCCESS if successful. NS_LDAP_OP_FAILED otherwise.

#### 5.7.6 __ns_ldap_freeError()

**Synopsis:**

```
     int __ns_ldap_freeError(
                  ns_ldap_error_t **errorp);
```

**Description:** This function frees the errorp structure and sets the errorp pointers to NULL.

**NOTE:** This routine takes the address of the pointer for the error structure.

__ns_ldap_freeError() frees the ns_ldap_error_t structure allocated by any Simplified LDAP API.

**Returns:** this function returns NS_LDAP_SUCCESS if successful. NS_LDAP_OP_FAILED otherwise.

#### 5.7.7 __ns_ldap_freeSearchDescriptors()

**Synopsis:**

```
int __ns_ldap_freeSearchDescriptors(
              ns_ldap_search_desc_t ***desc);
```

**Description:** This function is used to free memory resources associated with a search descriptor.

**NOTE:** This routine takes the address of the pointer to an array of service search descriptors.

__ns_ldap_freeSearchDescriptors() frees the ns_ldap_search_desc_t structure arrays allocated by the following function:

- __ns_ldap_getSearchDescriptors()

**Returns:** these function return NS_LDAP_SUCCESS if successful. NS_LDAP_OP_FAILED otherwise.

#### 5.7.8 __ns_ldap_freeCred()

**Synopsis:**

```
int __ns_ldap_freeCred(
              const ns_cred_t **credp);
```

**Description:** This function frees memory resources associated with a credential structure.

**Returns:** these function return NS_LDAP_SUCCESS if successful. NS_LDAP_OP_FAILED otherwise.

#### 5.7.9 __ns_ldap_freeAttributeMaps()

**Synopsis:**

```
int __ns_ldap_freeAttributeMaps(
              ns_ldap_attribute_map_t ***maps);
```

**Description:** This function is used to free memory resources associated with attribute maps.

**NOTE:** This routine takes the address of the pointer to an array of attribute maps.

__ns_ldap_freeAttributeMaps() frees the ns_ldap_attribute_map_t structure arrays allocated by the following function:

- __ns_ldap_getAttributeMaps()

**Returns:** these function return NS_LDAP_SUCCESS if successful. NS_LDAP_OP_FAILED otherwise.

#### 5.7.10 __ns_ldap_freeObjectclassMaps()

**Synopsis:**

```
int __ns_ldap_freeObjectclassMaps(
              ns_ldap_objectclass_map_t ***maps);
```

**Description:** This function is used to free memory resources associated with objectclass maps.

**NOTE**: This routine takes the address of the pointer to an array of object class maps.

__ns_ldap_freeObjectclassMaps() frees the ns_ldap_objectclass_map_t structure arrays allocated by the following function:

- __ns_ldap_getObjectclassMaps()

**Returns:** these function return NS_LDAP_SUCCESS if successful. NS_LDAP_OP_FAILED otherwise.

## 5.8 Simplified Library Wrapper Functions [currently these interfaces are non-functional, and unsupported]

The following LDAP library wrappers are provided in the simplified interface for advanced users. The routines map to the standard LDAP library functions with the following exceptions. The LDAP server connection is maintained by the simplified library based on the necessary profiles and configuration setups defined in previous sections. The LDAP *ld argument used in the standard library functions is replaced by the flags variable. The flags variable may contain any of the flags defined in Flags. Currently, the only supported flags are NS_LDAP_KEEP_CONN and NS_LDAP_NOMAP (both of which are the default behavior). In case of modify routines, application must call __ns_ldap_auth() with a NS_LDAP_KEEP_CONN prior to calling the wrapper routines.

In this release these routines do not perform attribute and objectclass mapping on the attributes passed through the functions. Search bases. This mapping is the responsibility of the caller.

```
int __ns_ldap_search(int flags, char *base, int scope, char
*filter, char *attrs[], int attrsonly);

int __ns_ldap_search_s(int flags, char *base, int scope, char
*filter, char *attrs[], int attrsonly, LDAPMessage **res);

int __ns_ldap_search_st(int flags, char *base, int scope, char
*filter,   char *attrs[], int attrsonly, struct timeval
*timeout, LDAPMessage **res);

int __ns_ldap_search_ext(int flags, char *base, int scope, char
*filter, char **attrs, int attrsonly, LDAPControl **serverctrls,
LDAPControl   **clientctrls,   struct    timeval
*timeoutp, int sizelimit, int *msgidp);

int __ns_ldap_search_ext_s(int flags, char *base, int scope, char
*filter, char **attrs, int attrsonly, LDAPControl **serverctrls,
LDAPControl   **clientctrls,   struct    timeval
```

```
*timeoutp, int sizelimit);

int __ns_ldap_compare(int flags, char *dn, char *attr, char
*value);

int __ns_ldap_compare_s(int flags, char *dn, char *attr, char
*value);

int __ns_ldap_compare_ext(int flags, char *dn, char *attr, struct
berval   *bvalue,   LDAPControl **serverctrls, LDAPControl
**clientctrls, int *msgidp);

int __ns_ldap_compare_ext_s(int flags, char *dn, char *attr,
struct berval *bvalue, LDAPControl **serverctrls,
LDAPControl **clientctrls);

int __ns_ldap_result(int flags, int msgid, int all, struct timeval
*timeout, LDAPMessage **result);

int __ns_ldap_add(int flags, char *dn, LDAPMod *attrs[]);

int __ns_ldap_add_s(int flags, char *dn, LDAPMod *attrs[]);

int __ns_ldap_add_ext(int flags, char *dn, LDAPMod **attrs,
LDAPControl **serverctrls, int * msgidp);

int __ns_ldap_add_ext_s(int flags, char *dn, LDAPMod **attrs,
LDAPControl **serverctrls, LDAPControl **clientctrls);

int __ns_ldap_modify(int flags, char *dn, LDAPMod *mods[]);

int __ns_ldap_modify_s(int flags, char *dn, LDAPMod *mods[]);

int __ns_ldap_modify_ext(int flags, char *dn, LDAPMod **mods,
LDAPControl **serverctrls, LDAPControl **clientctrls, int
*msgidp);

int __ns_ldap_modify_ext_s(int flags, char *dn, LDAPMod **mods,
LDAPControl **serverctrls, LDAPControl **clientctrls);

int __ns_ldap_delete(int flags, char *dn);

int __ns_ldap_delete_s(int flags, char *dn);

int __ns_ldap_delete_ext(int flags, char *dn, LDAPControl
**serverctrls, LDAPControl **clientctrls, int *msgidp);

int __ns_ldap_delete_ext_s(int flags, char *dn, LDAPControl
**serverctrls, LDAPControl **clientctrls);

int __ns_ldap_modrdn(int flags, char **dn, char **newrdn);

int __ns_ldap_modrdn_s(int flags, char **dn, char **newrdn, int
deleteoldrdn);

int __ns_ldap_modrdn2(int flags, char **dn, char **newrdn, int
deleteoldrdn);

int __ns_ldap_modrdn2_s(int flags, char **dn, char **newrdn, int
deleteoldrdn);
```

```
int __ns_ldap_rename(int flags, char *dn, char *newrdn, char
*newparent, int deleteoldrdn, LDAPControl **serverctrls,
LDAPControl **clientctrls, int *msgidp);

int __ns_ldap_rename_s(int flags, char *dn, char *newrdn, char
*newparent, int deleteoldrdn, LDAPControl **serverctrls,
LDAPControl **clientctrls);
```

# 6.0 LDAP Back-End Configuration Format and Parameter Formats

## 6.1 LDAP Client Configuration

The Simplified LDAP interface gets its LDAP configuration information from the ldap_cachemgr(1M) with an exception to applications utilizing the Standalone mode (see Section 5.1, page 16). ldap_cachemgr(1M), as well as “standalone” applications, get the configuration from a combination of loading a profile from the directory server and using the local active LDAP configuration.

Most of this information is pre-configured by administrators and stored in the profile on the LDAP server. During the client initialization stage, this information is downloaded to the client where a local copy, along with the credential information, is stored in a LDAP back-end configuration. The information will be shared by all processes on the LDAP client machine through the ldap_cachemgr(1M) and the Simplified LDAP interface with an exception to the “standalone” applications accessing the configuration directly. A significant change from the original profile is that as of phase II, the credential information is no longer downloaded from the directory and must be configured locally using the naming configuration utilities (nscfg(1M) and nsadm(1M)) or the ldapclient(1M) command. This is to improve the security of the proxy credentials and ultimately the security of the data stored in the server.

ldap_cachemgr(1M) ensures the information in the local LDAP configuration is valid by downloading the latest profile and storing it in the local storage at regular intervals. The expiration period (“Time-To-Live” value) for the profile itself is stored in the profile as well.

Some configuration information is required while other information optional. This is explained in the following sections. You must use nscfg(1M) or ldapclient(1M) to edit any parameter in a local LDAP configuration. No manual modification is supported. Mis-configuring the active LDAP back-end configuration may cause failure in naming lookups from your system, and possibility of a hung client.

## 6.2 LDAP Back-end Configuration Index Type

The following index types are used to specify which configuration parameter is in the data field. ParamIndexType is an enum type which is used to identify various configuration parameters to __ns_getParam() and __ns_setParam().

For the __ns_ldap_setParam() function, the data parameter is a pointer to the new data value to be set. For the __ns_ldap_getParam() function, the data parameter is a pointer to a NULL terminated array of pointers that point to parameter values returned from the function. The value type is defined in the following sections.

For example, in the case for NS_LDAP_SERVERS_P, the value type should be as follows:

- for __ns_ldap_setParam(): (void *data) should be cast to (char *data);
- for __ns_ldap_getParam(): (void ***data) should be cast to (char ***data).

In the case NS_LDAP_TRANSPORT_SEC_P, the value type should be as follows:

- for __ns_ldap_setParam(): (void *data) should be cast to (int *data);
- for __ns_ldap_getParam(): (void ***data) should be cast to (int ***data).

The following is a list of supported parameter indexes:

```
typedef enum {
    NS_LDAP_FILE_VERSION_P,       /* client information version number */
    NS_LDAP_BINDDN_P,             /* proxy bind DN */
    NS_LDAP_BINDPASSWD_P,         /* password used for proxy  DN */
    NS_LDAP_SERVERS_P,            /* LDAP server list */
    NS_LDAP_SEARCH_BASEDN_P,      /* base DN for LDAP operation */
    NS_LDAP_AUTH_P,               /* LDAP Authentication mechanism * /
    NS_LDAP_SEARCH_REF_P,         /* search referral * /
    NS_LDAP_EXP_P,                /* expiration time for the cached info */
    NS_LDAP_CERTDB_PATH_P,        /* proxy certificate path */
    NS_LDAP_SEARCH_DN_P ,         /* alternate baseDN for search  */
    NS_LDAP_SEARCH_SCOPE_P ,      /* searching scope * /
    NS_LDAP_BIND_TIME_P,          /* bind time limit */
    NS_LDAP_SEARCH_TIME_P,        /* searching time limit */
    NS_LDAP_SERVER_PREF_P ,       /* server preference  filter */
    NS_LDAP_PREF_ONLY_P ,         /* only contact preferred servers * /
    NS_LDAP_CACHETTL_P ,          /* cache time to live */
    NS_LDAP_PROFILE_P ,           /* the profile name */
    NS_LDAP_CREDENTIAL_LEVEL_P,   /* credential level */
    NS_LDAP_SERVICE_SEARCH_DESC_P, /* service search descriptors */
    NS_LDAP_ATTRIBUTEMAP_P,       /* attribute maps */
    NS_LDAP_OBJECTCLASSMAP_P,     /* objectclass maps */
    NS_LDAP_SERVER_LIST_REFRESH_P, /* ldap_cachemgr server list TTL */
    NS_LDAP_SERVICE_AUTH_METHOD_P, /* service authentication method */
    NS_LDAP_SERVICE_CRED_LEVEL_P  /* service credential method */
} ParamIndexType ;
```

**NOTE:** The parameter name specified in an LDAP back-end configuration is the same as the parameter type defined in the ParamIndexType but without the suffix “_P”.

### 6.3 Required Configuration Parameters

The following parameters are required by all clients in order to locate the servers for the desired LDAP session. These parameters are configured on the client during the initialization stage and refreshed periodically when the expiration time expires if a profile was used to configure the client.

Unless specified, all parameters listed below are single-value parameters. Multi-value parameters in this context means multiple results may be returned through __ns_ldap_getParam() call. Depending on the parameter, this might mean that multiple values exist in the same entry using a special separator or in multiple lines.

**NS_LDAP_SERVERS_P**

List of the default servers for the DUA. The data parameter is of type:

set - (char*) pointer to the server addresses with the optional colon separated port number;

get - (char ***) pointer to a NULL terminated array of pointers to server addresses.

**NOTE:** This is a multivalued parameter.
In a local LDAP back-end configuration, the servers are defined by a comma "," separated list.

**NOTE:** This parameter must be defined in the client cache file.

An LDAP configuration example:
NS_LDAP_SERVERS= 100.100.100.1,100.200.200.1:2222

**NS_LDAP_SEARCH_BASEDN_P**

Default base DN used for LDAP operation. The data parameter is of type:

set - (char *) pointer to the search baseDN;

get - (char ***) pointer to a NULL terminated array of pointers to the baseDN.

**NOTE:** This parameter must be defined in the client cache file.

An LDAP configuration example:
NS_LDAP_SEARCH_BASEDN= dc=eng,dc=sun,dc=com

**NS_LDAP_EXP_P**

The expiration time for the configuration information in the client active LDAP configuration (see Section 6.1, page 38). This time is derived based on the time that a local configuration was last refreshed plus the TTL value for the configuration information as defined during the setup time through the profile. Once this time has expired, the configuration is refreshed through ldap_cachemgr. The data parameter is of type:

set - (char *) pointer to the expiration time as defined in time(2).

get - (int ***) pointer to a NULL terminated array of pointers to the expiration time.

The default value is 12 hours from the last refresh. A value of 0 (zero)

means that the cached information will never expire.

An LDAP configuration example:
This parameter is calculated by ldap_cachemgr and is not stored in the file.

## 6.4 Optional Configuration Parameters

The optional client information are client specific parameters which may be used by a client program. For example, the client's binding DN and password, security method and alternate searchDN. These optional parameters are defined during the initialization either manually through the ldapclient command or based on the DUA profile stored on the LDAP server. When defined through the ldapclient command, these parameters will never change unless the client re-initializes the machine or override them internally with the __ns_ldap_setParam() interface. When defined based on the client profile, ldap_cachemgr updates these parameters once the expiration time expires.

Unless specified, all parameters listed below are single-value parameters. Multi-value parameters in this context means multiple results may be returned through __ns_ldap_getParam() call. Depending on the parameter, this might mean that multiple values exist in the same entry using a special separator or in multiple lines.

**NS_LDAP_BINDDN_P**

The directory DN for the proxy entity, the LDAP identity used during the authentication process. This parameter is only used when the credential level is set to proxy and at least one of the authentication methods require a bind DN. If the value of the parameter is set to "NS_HOST_PROXY", the host name's directory entry will be used to bind to the server as the proxy entity (Section 8.1 on page 51). The data parameter is of type:

set - (char *)

pointer the bind DN name;

get -

(char ***) pointer to a NULL terminated array of pointers to the bind DN.

There is no default value for this parameter.

An LDAP configuration example:
NS_LDAP_BINDDN= cn=proxyagent,ou=profile,dc=eng,dc=sun,dc=com

**NS_LDAP_BINDPASSWD_P**

Password used for authenticating proxy user (as defined in (NS_LDAP_BINDDN) to the directory. This is only used when the client is configured with a credential level of proxy and at least one of the authentication methods require a password credential. The data parameter is of type:

set - (char *)

pointer to the password;

get -

(char ***) pointer to a NULL terminated array of pointers to the password.

There is no default value for this parameter.

An LDAP configuration example:
NS_LDAP_BINDPASSWD= {NS1}4a3788e8c053424f

**NS_LDAP_AUTH_P**

The Authentication method. This defines the authentication method to be used when binding to the directory server. For more information about ns_auth_t structure see Section 4.4 on page 8.

The data parameter is of type:

set - (char *) pointer to the authentication method;

get - (ns_auth_t ***) pointer a NULL terminated array of pointers to the ns_auth_t structure.

The default value is NS_LDAP_AUTH_NONE in case of version 1, and “none” in case of version 2 profile (see section 13.0, p. 61 for the detail on the LDAP profile versions).

**NOTE:** This is a multivalued parameter.
In a local LDAP back-end configuration, the authentication methods are defined by a comma “,” separated list for version 1, and “;” in case of a version 2 profile (see section 13.0, p. 61 for the detail on the LDAP profile versions).

An LDAP configuration example:
NS_LDAP_AUTH= sasl/DIGEST-MD5;simple;none

**NS_LDAP_CREDENTIAL_LEVEL_P**

Define the credential level to be used for authentication. This parameter is only supported in version 2 profiles (see section 13.0, p. 61 for the detail on the LDAP profile versions). The return type is an enum as defined on page 8. The data parameter is of type:

set - (char *) pointer to the credential level as defined below;

get - (CredLevel_t ***) pointer to a NULL terminated array of pointers to the credential level.

The default is proxy for version 1, and anonymous for version 2 (see section 13.0, p. 61 for the detail on the LDAP profile versions).

**NOTE:** This is a multivalued parameter.
In a local LDAP back-end configuration, the credential levels are defined by a space “ ” separated list.

An LDAP configuration example:
NS_LDAP_CREDENTIAL_LEVEL= proxy anonymous

**NS_LDAP_CERT_PATH_P**

The certificate path for the servers certificate or the trust chain. The parameter can either specify the database file, or the path to the default file 'cert7.db'. This is for support of TLS. The data parameter is of type:

set - (char *)

pointer to the path where the certificate is stored.

get - (char ***) pointer to a NULL terminated array of pointers to certificate path.

The default value is "/var/ldap".

An LDAP configuration example:
NS_LDAP_CERT_PATH= /var/ldap/cert7.db

**NS_LDAP_SEARCH_DN_P**

The alternate baseDN for LDAP search operation. The data parameter is of type:

set - (char *)

pointer to the alternate search baseDN;

get - (char ***) pointer to a NULL terminated array of pointers to the alternate baseDNs.

The alternate baseDN consists of the following format:

<database>:<alt-basedDN-list>
The <database> is the database name defined in the nsswitch.conf(4). The <alt-basedDN-list> is a list of alternate baseDN enclosed with parenthesis and separated by a comma. The lookups to a specific database will be done in the order as it's specified in this parameter.

**NOTE:** This is a multivalued parameter.
In an LDAP back-end configuration, alternate baseDNs are defined in a separate line for each database.

An LDAP configuration example:
NS_LDAP_SEARCH_DN= passwd:(ou=peope,dc=eng,dc=sun,dc=com),\
(ou=people,dc=corp,dc=eng,dc=sun,dc=com)

**NS_LDAP_SERVICE_SEARCH_DESC_P**

The service search descriptors defined in the profile. The data parameter is of type:

set - (char *)

pointer to the service search descriptor;

get - (char ***) pointer to a NULL terminated array of pointers to the service search descriptors.

The service search descriptor format is:

service:base[?[scope][?[filter]]];base[?[scope][?[filter]]];...
The <service> is the service name as defined in Section 4.3, page 7. For more information, refer to Version 2 profile.

**NOTE:** This is a multivalued parameter.
In an LDAP back-end configuration, a separate entry is used for each value.

An LDAP configuration example:
NS_LDAP_SERVICE_SEARCH_DESC= passwd:\
ou=people,dc=eng,dc=sun,dc=com?one?(objectclass=posixAccount);\
ou=people,dc=corp,dc=sun,dc=com?sub

**NS_LDAP_SEARCH_SCOPE_P**

LDAP search scope used when searching for the LDAP entry. The data parameter is of type:

set - (char *) pointer to the search scope type as defined in below;

get - (int ***) pointer to a NULL terminated array of pointers to the search scope type.

Supported search scope types are:

NS_LDAP_SCOPE_BASE,
NS_LDAP_SCOPE_ONELEVEL,
NS_LDAP_SCOPE_SUBTREE.

The default is NS_LDAP_SCOPE_ONELEVEL.

An LDAP configuration example:
NS_LDAP_SEARCH_SCOPE= NS_LDAP_SCOPE_ONELEVEL

**NS_LDAP_SEARCH_TIME_P**

LDAP search time limit when searching for an LDAP entry. The data parameter is of type:

set - (char *) pointer to a string that represents the time limit in seconds;

get - (int ***) pointer to a NULL terminated array of pointers to the time limit.

There is no default for this parameter.

An LDAP configuration example:
NS_LDAP_SEARCH_TIME= 30

**NS_LDAP_BIND_TIME_P**

LDAP bind time limit when binding to an LDAP server (TCP connect time). A value of 0 means the time out is set to default TCP time out. The data parameter is of type:

set - (char *)

pointer to a string that represents the time limit in seconds;

get - (int ***) pointer to a NULL terminated array of pointers to the time limit.

The default is 10 seconds.

An LDAP configuration example:
NS_LDAP_BIND_TIME= 2

**NS_LDAP_SERVER_LIST_REFRESH_P**

An integer number of seconds used to specify the time between server list refreshes in the ldap cache manager. A value of 0 indicates no refresh should be made. The data parameter is of type:

set - (char *)

pointer to the character string;

get - (int ***) pointer to a NULL terminated array of pointers to the character string.

The default value is 600 seconds (10 minutes in seconds).

An LDAP configuration example:
NS_LDAP_SERVER_LIST_REFRESH= 5000

**NS_LDAP_SERVER_PREF_P**

Server preference for the client. The server preference is specified by the server address(es). Clients will try to use the servers specified before trying other servers. The data parameter is of type:

set - (char *)

pointer to a preferred server;

get - (char ***) pointer to a NULL terminated array of pointers to the preferred servers.

The default is to use servers on the local subnet first.

**NOTE:** This is a multivalued parameter.
In an LDAP back-end configuration, the preferred servers are defined either by a comma “,” separated list of servers.

An LDAP configuration example:
NS_LDAP_SERVER_PREF= 100.100.100.1, 100.100.100.2

**NS_LDAP_PREF_ONLY_P**

A boolean flag indicating that only the servers defined in the NS_LDAP_SERVER_PREF_P list are to be contacted. The data parameter is of type:

set - (char *) pointer to the character string;

get - (char ***) pointer to a NULL terminated array of pointers to the character string.

The string value can be either NS_LDAP_TRUE or NS_LDAP_FALSE.

The default value is NS_LDAP_FALSE.

An LDAP configuration example:
NS_LDAP_PREF_ONLY= NS_LDAP_FALSE

**NS_LDAP_CACHETTL_P**

An integer number of seconds used to specify the time between local configuration refreshes in the ldap cache manager (i.e. config data fetched from the server). A value of zero implies that the local configuration is never refreshed. The data parameter is of type:

set - (char *) pointer to the character string;

get - (int ***) pointer to a NULL terminated array of pointers to the character string.

The default value is 43200 (12 hours in seconds).

An LDAP configuration example:
NS_LDAP_CACHETTL= 5000

**NS_LDAP_PROFILE_P** (READ ONLY)

The profile name used to setup this client cache file. This name is also used when refreshing the client side information. The profile name could be a fully qualified distinguished name, or just the name of the profile. The data parameter is of type:

set - set operation is not supported;
get - (char ***) pointer to a NULL terminated array of pointers to the profile name.

There is no default for this parameter.

An LDAP configuration example:
NS_LDAP_PROFILE= engineering

**NS_LDAP_SEARCH_REF_P**

Search referral option. The data parameter is of type:

set - (char *)

pointer to the referral option;

get - (int ***) pointer to a NULL terminated array of pointers to the referral option.

Supported referral types are:

NS_LDAP_FOLLOWREF,
NS_LDAP_NOREF

**NOTE**: These types are internally defined integer constants.

The default is NS_LDAP_FOLLOWREF.

An LDAP configuration example:
NS_LDAP_SEARCH_REF= NS_LDAP_FOLLOWREF

**NS_LDAP_FILE_VERSION_P**

LDAP configuration version number. The data parameter is of type:

set - (char *)

pointer to the version number string;

get - (char ***) pointer to a NULL terminated array of pointers to the character string.

The default for version “1.0” is a NULL string.

An LDAP configuration example:
NS_LDAP_FILE_VERSION= 2.0

**NS_LDAP_ATTRIBUTEMAP_P**

Attribute maps configured in the profile:

set - (char *)

pointer to the attribute map string;

get - (char ***) pointer to a NULL terminated array of pointers to the character string.

The default for version “1.0” is a NULL string.

**NOTE:** This is a multivalued parameter.
In an LDAP back-end configuration, a separate entry is used for each value.

An LDAP configuration example:
NS_LDAP_ATTRIBUTEMAP= passwd:uid=unixUid

**NS_LDAP_OBJECTCLASSMAP_P**

Objectclass maps configured in the profile:

set - (char *)

pointer to the objectclass map string;

get -

(char ***) pointer to a NULL terminated array of pointers to the character string.

The default for version “1.0” is a NULL string.

**NOTE:** This is a multivalued parameter.
In an LDAP back-end configuration, a separate entry is used for each value.

An LDAP configuration example:
NS_LDAP_OBJECTCLASSMAP= passwd:posixAccount=internalAccount

**NS_LDAP_SERVICE_AUTH_METHOD_P**

The service specific authentication method. This defines the authentication method to be used when binding to the directory server for a specific service. The data parameter is of type:

set - (char *)

pointer to the authentication method;

get -

(char ***) pointer a NULL terminated array of pointers to the service authentication method.

get returns the string representation of the authentication method along with the service name (separate by a ‘:’). Parsing and translating this to ns_auth_t structure is the responsibility of the application.

There is no default value.

**NOTE:** This is a multivalued parameter.
In a local LDAP configuration, the authentication methods are defined by a semicolon “;” separated list. In addition each service’s authentication method is specified in one line in the configuration.

An LDAP configuration example:
NS_LDAP_SERVICE_AUTH_METHOD= passwd-cmd:sasl/DIGEST-MD5;simple

**NS_LDAP_SERVICE_CRED_LEVEL_P**

Define the service specific credential level to be used for authentication. This parameter is only supported in version 2 profile (see section 13.0, p. 61 for the detail on the LDAP profile versions). The return type is an enum as

defined in Section 4.4, page 8. The data parameter is of type:

set - (char *) pointer to the credential level as defined below;

get - (char ***) pointer to a NULL terminated array of pointers to the service credential level.

get returns the string representation of the credential level along with the service name (separated by a ':'). Parsing and translating this string is the responsibility of the application.

There is no default value.

**NOTE:** This is a multivalued parameter.
In a local LDAP configuration, the credential levels are defined by a space " " separated list. In addition each service's authentication method is specified in one line in the configuration.

An LDAP configuration example:
NS_LDAP_SERVICE_CRED_LEVEL= passwd:self proxy

### 6.5 Sample LDAP configuration

To access the information stored in the directory, clients must know some basic configuration information. A local copy of the configuration as well as the boot-strapping info is stored in an LDAP back-end configuration containing all configuration information including the proxy credentials required to authenticate to the directory if any are configured.

### 6.6 Sample v1 profile configuration [version 1 profiles are obsolete – use is not recommended]

The following is a sample local LDAP configuration for a version 1 profile (see section 13.0, p. 61 for the detail on the LDAP profile versions):

```
ldap ldapclient
          NS_LDAP_FILE_VERSION= 1.0
          NS_LDAP_SERVERS= 129.1.1.1:234, 129.2.2.2, 129, 3.3.3:123
          NS_LDAP_SEARCH_BASEDN= dc=eng,dc=sun,dc=com
          NS_LDAP_TRANSPORT_SEC= NS_LDAP_SEC_NONE
          NS_LDAP_SEARCH_REF= NS_LDAP_NOREF
          NS_LDAP_SEARCH_DN= hosts:(ou=hosts,dc=sun,dc=com)
          NS_LDAP_SEARCH_SCOPE= NS_LDAP_SCOPE_ONELEVEL
          NS_LDAP_SEARCH_TIME= 20
          NS_LDAP_SERVER_PREF= 129.4.4.4:123, 129.5.5.5
          NS_LDAP_PREF_ONLY= NS_LDAP_TRUE
          NS_LDAP_CACHETTL= 1000
          NS_LDAP_PROFILE= version1
          NS_LDAP_BINDDN= cn=proxyagent,ou=profile,dc=eng,dc=sun,dc=com
          NS_LDAP_BINDPASSWD= {NS1}4a3788e8c05342
```

### 6.7 Sample v2 profile configuration

The following is a sample local LDAP configuration for a version 2 profile (see section 13.0, p. 61 for the detail on the LDAP profile versions):

```
ldap ldapclient
     NS_LDAP_FILE_VERSION= 2.0
     NS_LDAP_SERVERS= 129.1.1.1:234 129.2.2.2
     NS_LDAP_SEARCH_BASEDN= dc=eng,dc=sun,dc=com
     NS_LDAP_AUTH= sasl/DIGEST-MD5;simple;none
     NS_LDAP_SEARCH_REF= TRUE
     NS_LDAP_SEARCH_SCOPE= one
     NS_LDAP_SEARCH_TIME= 20
     NS_LDAP_SERVER_PREF= 129.4.4.4:123 129.5.5.5
     NS_LDAP_CACHETTL= 1000
     NS_LDAP_PROFILE= version2
     NS_LDAP_CREDENTIAL_LEVEL= proxy anonymous
     NS_LDAP_SERVICE_SEARCH_DESC=passwd:ou=employees,?one;ou=contractors,
     NS_LDAP_SERVICE_SEARCH_DESC= hosts:ou=hosts,dc=sun,dc=com?sub
     NS_LDAP_BIND_TIME= 5
     NS_LDAP_ATTRIBUTEMAP= passwd:gecos=sn telephoneNumber officeLocation
     NS_LDAP_ATTRIBUTEMAP= passwd:uid=csuid
     NS_LDAP_OBJECTCLASSMAP= passwd:posixAccount=myAccount
     NS_LDAP_SERVER_LIST_REFRESH= 1000
     NS_LDAP_SERVICE_AUTH_METHOD= passwd-cmd:tls:simple
     NS_LDAP_SERVICE_AUTH_METHOD= pam_ldap:tls:sasl/DIGEST-MD5
     NS_LDAP_SERVICE_AUTH_METHOD= keyserv:tls:simple
     NS_LDAP_BINDDN= cn=proxyagent,ou=profile,dc=eng,dc=sun,dc=com
     NS_LDAP_BINDPASSWD= {NS1}4a3788e8c05342
```

## 7.0 ldap_cachemgr daemon

The ldap_cachemgr daemon is the process that serves the following purposes:

- refreshing the information in the active LDAP back-end configuration from an appropriate LDAP server;
- accessing the configuration information stored in the active LDAP back-end configuration;
- maintaining a sorted and up to date list of servers to be contacted through the Simplified LDAP API as defined by the profile;
- caching common lookups done by the Simplified LDAP API to improve the performance and to reduce the number of unnecessary repetitive lookups performed by individual clients. Note that ldap_cachemgr does not cache search results, but rather the results of rootDSE lookups (e.g. supported sasl mechanisms, controls, etc.);
- building and maintaining a cache of directory RDN to DNS domain name mapping so host entries can be qualified.

The most visible difference between phase I and phase 2, is that ldap_cachemgr process is required to be running at all times after the LDAP client has been initialized. In phase 1 if the cache manager was not running then the refresh was done per process and the active LDAP configuration was not updated. Also, only anonymous connections could be made to the directory server (unless access permission of the active LDAP configuration was changed to allow read access to everyone).

Besides providing the update capability, the ldap_cachemgr also reduces the LDAP network traffic generated during the refresh and provides a robust parsing mechanism which can flag invalid values in the active LDAP configuration. Client processes can only contact the ldap_cachemgr using door(2) system calls through the Simplified LDAP API. There is no exposed programmatic interface to the ldap_cachemgr. See ldap_cachemgr(1M) man page for more info.

In the phase1 of this project, the ldap_cachemgr's primary focus was the control and the management of the actife LDAP configuration. It did not attempt to cache information for usual operation which are almost always required by applications (e.g. rootDSE lookup).

In the phase 2, the LDAP cache manager enhances the DOORs interface between the client library (libsldap) and the cache manager so that the communication between the two components will be more formalized and extensible.

ldap_cachemgr manages the currently configured server list for the DUA and caches the appropriate RootDSE entries for those LDAP servers. This enhancement alone, reduces the number of lookups for common operations performed by NSS LDAP back-end by approximately 50%.

Additionally, ldap_cachemgr maintains a cache of LDAP RDNs and their corresponding DNS domain names. This is used for host lookups and improves the performance of qualifying host names.

## 8.0 Security

The security requirement for the Simplified LDAP API can be divided into two main categories: directory queries (lookup operations), and directory updates (update operations). Depending on the class of operation different levels of security might be required. Additionally the security model could play a very important role in providing a complete common authentication solution with other operating systems and therefore the security architecture should be very flexible to allow support of additional security mechanisms as they become available in future.

Phase one of this project provided support for only the simplest of the security model. The only supported authentication methods were none, simple, and sasl/CRAM-MD5. Furthermore no transport layer security was supported (unless IPSec was used to secure the communication between the client and server).

In this phase of the project, a more complete set of options are supported which should provide customers with a range of options in setting up their environment. The options range from anonymous (no security, very easy to manage, very fast), to TLS protected simple authentication (very secure, difficult to manage, very slow), and some options in between.

Since the current plan is to integrate the iPlanet Directory Server (iDS) into Solaris and use it as a core part of the operating environment, this project is mainly geared toward using all security options that are offered through iDS.

The latest version of iPlanet Directory server, 5.0, supports the following authentication methods:

- none (anonymous)
- simple
- sasl DIGEST-MD5 (no privacy or integrity options)
- X509 certificates

## 8.1 Security Model

The security model in this project is broken up to three sections.

### 8.1.1 Credential Level

To access the information stored in the directory clients can either authenticate to the directory, or use an un-authenticated connection. This notion is defined as the credential level in the profile. There are three different levels possible to setup the client and server for this:

1) Clients do not authenticate to the directory and all the information is available to anyone that can make a connection to the directory. This is called ‘anonymous’ credential level.
2) Clients authenticate to the directory using a proxy identity. This means that this entity is shared by a number of clients to access the directory content. The directory can then authorize the proxy entry to access certain entries in the directory using its ACLs. This is called ‘proxy’ credential level. Using ‘proxy’ does not mean that there will have to be one single entry for all clients to authenticate to the directory however. There can be two extreme in using proxy identity: one proxy entry for a network of clients, and one proxy entry per client. Depending on the authorization requirements of the site, appropriate proxy identity(ies) can be chosen. Note that if the one proxy per client is desirable, the host entry for the client can be used as the proxy entry as it supports the ‘userPassword’ attribute.
3) Every user on each client authenticates to the directory as themselves. Using this method would allow the finest level of granularity in providing access control to the information in the directory. Each user can be individually granted or denied access to entries or even attributes of entries. This is called the ‘self’ credential level.

Currently only the first two options are supported since option #3 requires major efforts in caching authentication information in a secure local storage mechanism. Option #3 is a much more viable solution once both solaris and the directory server support more advanced security mechanisms such as smart cards and

certificate base authentication, or Kerberos. The design of the security layer is done in a way that adding support for option #3 would be achieved with minimal additional work to simplified layer. Support for this feature is planned once the directory server support Kerberos, but is outside the scope of this project.

The credential can be a multi-valued entry meaning that multiple credentials can be allowed. For example, it can be defined to be 'proxy anonymous', which means the client will try the proxy credential level first, if it is not successful, then it will try anonymous.

#### 8.1.2 Authentication Method

Once a identity is established, then an authentication method must be picked to authenticate to the directory with. The authentication method in the profile may also have a transport security option associated with it. For example a simple authentication method can be used over a TLS secured layer for protecting the identities password.

Similar to credential level, the authentication method can be defined as a multi-valued entry. For example, it can be defined to be 'tls:simple; sasl/DIGEST-MD5'. In this example, the client will try to establish a TLS secured line and use a simple authentication bind operation, if that is not successful, then it will try using a DIGEST-MD5 SASL bind operation.

#### 8.1.3 PAM Authentication

PAM framework is used for user authentication to Solaris. Depending on the pam module used in the stack, the security model for LDAP clients change. Clients can be configured to use either pam_unix module, or use the pam_ldap module.

pam_unix module, expects to be able to have read access to user's crypted password before it can authenticate a user. The user is prompted for his user name and password and pam_unix then calls getspnam() to get a copy of the users entry including his crypted password. pam_unix then encrypts the users password and compares it to the users retrieved entry, if they match, the authentication is successful. This works fine, but has a few potential problems associated with it. First the password must be stored using crypt format. Then it requires read access to the encrypted password by either everyone or the proxy identity (depending on the credential level used).

This model does not work with all generic directories or ldap deployments since not all deployments store passwords using crypt format. Additionally, as directories support more advanced security mechanisms, using a crypted password becomes the weak link in network security. To overcome these problems as well as providing support for stronger authentication methods, pam_ldap was introduced. pam_ldap authenticates the users directly to the directory server using the configured authentication method. Note that pam_ldap by definition ignores the credential level setting and always use 'self' to authenticate users.

### 8.2 Client Configuration

If the client is configured to use the proxy credential level, it also needs to store a distinguished name and a password (or other types of credential depending on the authentication method used). This information is stored in a local LDAP back-end configuration. This configuration is only readable by root. All credential information stored is encrypted using a simple two way encryption mechanism. The security of the credential is not the encryption, but rather the access permission. It is encrypted as an additional measure. One major difference between the version 1 profile vs. version 2 profile (see section 13.0, p. 61 for the detail on the LDAP profile

versions) is that the proxy credentials are not stored in the profile anymore and must be manually entered during each client initialization. This was done to improve the security of the system.

As mentioned before, the authentication method might also include information about the transport security. This is new for phase II of the project. In phase I, an additional configuration option "NS_LDAP_TRANSPORT_SEC" was supported which defined what type of transport security must be used. The new encoding format for the authentication method (defined in the DUAconfigProfile document, Version 2 profile) combines this information with the authentication method and stores it in authenticationMethod attribute.

In addition to the above two attributes, another attribute in the configuration profile exist which can be used to improve the security of the system. The service specific authentication method can be defined for a given service to increase the flexibility of the a given profile. Using these attributes each service can have its own level of security associated with it.

In this release the only services which support service authentication methods are 'pam_ldap', 'passwd-cmd', and 'keryserv'. Use of this attribute allows pam_ldap module to override the default authentication method for authenticating users. Similarly, if a service authentication method is defined for passwd-cmd service, the passwd command will use the configured method(s) to authenticate users to the directory for changing their passwords. And finally, defining a serviceAuthenticationMethod attribute for keyserv service, will result in newkey(1M) and chkey(1) to use the defined authentication method when communicating with the directory server. This can greatly improve the options available to the system administrator for choosing the appropriate model for their environment.

### 8.3 System Behavior

Since it is possible that a client is configured with multiple hosts, multiple credential levels, and multiple authentication methods, defining the exact behavior of it is very important. The simplest way to do this is to provide the pseudo code for the connection algorithm. This algorithm is defined in section 5.2 of the profile draft (Version 2 profile).

### 8.4 Write Operations

Write operations will ONLY be done with the 'self' credential level. The user must provide credentials appropriate for the authentication method chosen to the Simplified LDAP API calls. For example in case of sasl/DIGEST-MD5, the user name and a password must be passed to the call through use of ns_cred_t structure. Note that by definition if 'simple' is used as authentication method the password will be sent to the server clear text.

### 8.5 Access Controls

All read/write operations are subject to the ACL[1] restrictions that the LDAP Server might place on the attribute values. This is for most cases desirable since it provides a very powerful method to control access to various entries and their attribute values, but might cause some administration and debugging difficulties as well.

1 Access Control List, also known as Access Control Information (ACI)

One very important note that is in the documentation set is that the administrator MUST disable self modify access control to most entry attributes. If this is not done, then users can modify their own uidNumber, effectively becoming super user. `idsconfig(1M)` tool, sets this ACL for the iPlanet Directory Server.

If an anonymous credential level is used in conjunction with pam_unix, then the directory must be configured to allow read access to the userpassword attribute by anonymous users (all unauthenticated connections). For the obvious reason this is not a recommended configuration.

### 8.6 Password Management in pam_ldap

As mentioned before, pam_ldap was introduced to overcome some of the problems with the default pam_unix module. In addition to the direct authentication to the directory server, pam_ldap can also take advantage of the password management features available through the directory server. Unfortunately due to resource limitation this is outside the scope of this project, but the Simplified LDAP API has been modified to provide support for the required control mechanisms. Once this feature is added, pam_ldap can be used to provide account lock out, password restriction, password history, and password expiry features without major changes to the source base.

# 9.0 Tools and commands – These sections are overview only. See other documentation for full details.

## 9.1 idsconfig(1M)

`idsconfig(1M)` is a shell script which aids in configuring an already installed iPlanet/JES 5.x/6.x directory server. The script prompts for various information it needs, then it configures the directory server and populates the required structure to enable it use by clients, and finally it creates a profile and stores it in the directory. After the command has finished, the directory is ready to be populated by data and serve clients.

## 9.2 ldapclient(1M)

`ldapclient(1M)` is the tool that initializes clients. The tool has three modes of operation:

- profile based initialization
- manual initialization
- manual modification
- generate profiles in an LDIF format

In the profile download mode, the command requires some basic bootstrapping information to find a profile and download it. In this mode, as minimum, the IP address of the server that contains the profile is required. Additionally, a profile name can also be specified. This is the most useful mechanism for configuring clients. Not only the clients can easily get all their configuration parameter with minimal human intervention, the configuration is automatically updated at profileTTL intervals. This means the administrators can change the profile in one place and all clients are automatically are updated within the time limits defined in the profile itself.

The manual initialization is the less desired configuration option as each client must be manually configured. In this mode, various configuration options can be passed on to the ldapclient tool which creates a local LDAP back-end configuration and sets up the client to use LDAP for naming services.

The modification mode is used to update various configuration parameters of a client which was configured using manual initialization method.

And finally, the genprofile mode is used to create additional profiles to be used for client configuration. Using various command line options, this command generates an ldif output which can then be used to add the profile to the directory. The command will check the options for consistency and syntactical error before generating the ldif output.

**NOTE:** Since the name services configuration utilities nscfg(1M) and nsadm(1M) are introduced, ldapclient(1M) is considered obsolete and its use is discouraged.

## 9.3 ldapaddent(1M)

`ldapaddent(1M)` utility is used to migrate from /etc files to LDAP with relative ease. The command takes various text based files and stores them into the directory either using all the configuration parameters that the client is configured with or using an LDAP configuration provided by the user (so called “Standalone” schema, see Section 5.1, page 16). Since the command uses the Simplified LDAP API, it automatically stores all the information in the appropriate containers, and more importantly it uses the specified schema (if default schema is overridden using schema mapping feature).

This utility also has the ability to dump the information in the directory into the text based /etc file format.

## 9.4 ldaplist(1) command

`ldaplist(1)` command provides an easy way of looking up various entries in the directory. It can dump tables, containers, and look up entries for all configured services. Since it goes through the Simplified LDAP API, all configured parameters, automatically get used to access the correct information.

## 9.5 Generic LDAP tools

These tools were originally developed by University of Michigan and more or less are defacto standard tools for low level directory access. These tools are not delivered as part of this project, but since their usage can supplement use of the directory they are mentioned here. None of these commands use the configuration that is used by the Simplified LDAP API, and hence using them requires knowledge of the directory server setup.

### 9.5.1 Ldapsearch(1)

`ldapsearch(1)` command performs a search operation and returns one or more entries that match the specified filter. If a list of attributes is specified, then only the values for those attributes are displayed.

### 9.5.2 ldapmodify(1)/ldapadd(1)

`ldapadd(1)` is a hard link to the `ldapmodify(1)` command. These command perform a create or a modify operation depending on the mode they are in. Both commands take their input using LDIF format (RFC2849). Note that our implementation is not 100% compliant with the LDIF format specified in RFC2849.

### 9.5.3 ldapmodrdn(1)

`ldapmodrdn(1)` modifies an entries Relative Distinguished Name (RDN) using a modifyrdn operation.

## 9.6 ldapdelete(1)

`ldapdelete(1)` deletes one or more entries from the directory. It takes its input in the form of a DN.

# 10.0 Other Components

## 10.1 nss_ldap.so.1

The LDAP Switch backend module is a shared library which translates the frontend getXbyY() calls into LDAP calls and converts the LDAP results into appropriate date structure for the frontend. Similar to other Naming Switch backends, the LDAP backend `nss_ldap.so.1` shared library will provide the exact same set of interfaces used by the Naming Switch. For more detail on these interfaces, please see the Naming Switch design specification.

This project modified the NSS LDAP backend through the phase one of this project to use the new API and take advantage of the new standard based profile. The functionality is the same, the only difference is taking advantage of the new features.

## 10.2 Solaris Applications and Utilities

In addition to the discussed tools/utilities/libraries, there are a few applications/utilities in the OS/Net Consolidation of Solaris that use Naming service specific API instead of going through the Naming switch to access the standard naming information. For example, automounter daemon calls the NIS+ API nis_list() directly to retrieve auto_home information. Thus in order to fully integrate the LDAP Naming service, these applications/utilities had to be modified to use the Simplified LDAP API. In other words, making these applications/utilities LDAP friendly.

In this phase of the project, these applications had to be modified to use the new API and hence the new features.

NOTE: Updating applications/utilities which use private naming information (i.e. non-standard tables) will not be covered in this project.

The applications/utilities that are currently using the Naming specific APIs are:

- automounter daemon - to access the automounter information.
- sendmail daemon - to access the mail aliases and sendmail_var information.
- keyserv utilities - to access public key information.
- passwd utility - to update the user's password entry.
- pam_unix library - to update the user's password entry.

# 11.0 Client Setup Procedure

## 11.1 Setup Utility - ldapclient

A setup utility (`ldapclient`) is provided to set up the Solaris client machines to use LDAP Naming service. This does not include the server side setup. `ldapclient` does however assume that the server has been pre-configured and that the appropriate entries are created on the LDAP servers. This utility can also be used to update the information in the local active LDAP back-end configuration. See the `ldapclient(1M)` for more detail on the ldapclient utility.

There are two ways to initialize a client machine:

- Static - in this case the system administrator will have to define all the required parameters when initializing the client machine. This means the server address(es), the defaultSearchBase, the authentication method(s), etc. must be defined for the ldapclient utility. With this approach, the information stored in a local LDAP configuration is static and will never be refreshed.
- Profile - in this case the system administrator will only have to define the server address and the profile name. The ldapclient utility will automatically download all the proper information the client machine. The only exception to this is the proxy information. If the profile defines a credential level of “proxy”, then the proxy bind DN and password must be entered for each client.

`ldapclient` utility can also be used to modify the parameters in the active LDAP configuration. Unless the expiration time is set to 0 (zero) meaning the information never expires, the information updated by `ldapclient` can be overwritten after a refresh.

**NOTE:** Since the name services configuration utilities nscfg(1M) and nsadm(1M) are introduced, ldapclient(1M) is considered obsolete and its use is discouraged.

## 11.2 Suninstall

The install and upgrade process does include LDAP as one of the Naming service option. This work was done by the Install group. The only potential modification required to the current process is the requirements to enter proxy information if the profile is configured to use it. This modification to the install process will be done by the install group.

# 12.0 Interoperability

## 12.1 Directory Server Requirements

In order to support Solaris Naming clients, LDAP servers such as SunDS need to support the followings:

- LDAP v3 compliant
- Support at least of one of these two controls:
  Virtual List View (draft-ietf-ldapext-ldapv3-vlv-04.txt)

Simple Paged Result (RFC2696)

- Support for compound naming
- Support for auxiliary objectclasses

Almost all of these features are supported by most directory servers in the market. The main exception is Microsoft's Active Directory. MAD does not support compound naming of entries or auxiliary objectclasses. The requirement for auxiliary objectclass can be bypassed through some creative configuration, but the compound naming requirement is an open issue. Microsoft has claimed that the next version of their directory server will add proper support for both these features.

### 12.2 Interoperability with other clients

Currently, there are three implementations of the client side in the industry. A public domain version by Padl Software available for most unix flavors, one by HP for HP-UX, and the one for Solaris. In theory all these should interoperate, but in practice it has not been tested. Our goal is to complete the revision to RFC2307, and then set up a test event with HP and hopefully with Padl Software to test interoperability issues with the other implementations. A possible forum for this might be DirConnect (sponsored by OpenGroup).

### 12.3 Interoperability with Native LDAP Phase I

This project is 100% compatible with Native LDAP phase I clients. However, the new features can only be used with phase II clients only. This raises the question of what if the new features are required by a customer. Do they have to upgrade all their clients to Solaris 9, or are we going to provide the new features on Solaris 8 as well. However, this is beyond the scope of this project and needs to be answered by our marketing.

# 13.0 Appendix - Schema

## 13.1 Version 1 profile

This schema is what is used to define a version 1 in Native LDAP phase 1profile. Version 1 profiles are used for backward compatibility, but new deployments should use the new profile as it offers many new functions and is based on an IETF document (currently in draft stage). This schema defines 15 new attributes and one new objectclass. This schema is considered obsolete.

### 13.1.1 Attribute definition

```
( 1.3.6.1.4.1.42.2.27.5.1.15
  NAME 'SolarisLDAPServers'
  DESC 'LDAP Server address eg. 76.234.3.1:389'
  EQUALITY caseIgnoreIA5Match
  SYNTAX SolarisLDAPServerSyntax )

( 1.3.6.1.4.1.42.2.27.5.1.16
  NAME 'SolarisSearchBaseDN'
  DESC 'Search Base Distinguished Name'
  EQUALITY distinguishedNameMatch
  SYNTAX DN SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.17
  NAME 'SolarisCacheTTL'
  DESC 'TTL value for the profile information eg. 1w, 2d, 3h, 10m, or 5s'
  EQUALITY caseIgnoreMatch
  SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.18
  NAME 'SolarisBindDN'
  DESC 'DN to be used to bind to the directory as proxy'
  EQUALITY distinguishedNameMatch
  SYNTAX DN SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.19
  NAME 'SolarisBindPassword'
  DESC 'Password for bindDN to authenticate to the directory'
  EQUALITY caseExactIA5Match
  SYNTAX OctetString SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.20
  NAME 'SolarisAuthMethod'
  DESC 'Authentication method to be used eg. "NS_LDAP_AUTH_NONE",
       "NS_LDAP_AUTH_SIMPLE" or "NS_LDAP_AUTH_SASL_CRAM_MD5"'
```

```
  EQUALITY caseIgnoreIA5Match
  SYNTAX IA5String )

( 1.3.6.1.4.1.42.2.27.5.1.21
  NAME 'SolarisTransportSecurity'
  DESC 'Transport Level Security method to be used eg. "NS_LDAP_SEC_NONE"
       or "NS_LDAP_SEC_SASL_TLS"'
  EQUALITY caseIgnoreIA5Match
  SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.22
  NAME 'SolarisCertificatePath'
  DESC 'Path to certificate file/device'
  EQUALITY caseExactIA5Match
  SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.23
  NAME 'SolarisCertificatePassword'
  DESC 'Password or PIN that grants access to certificate.'
  EQUALITY caseExactIA5Match
  SYNTAX OctetString SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.24
  NAME 'SolarisDataSearchDN'
  DESC 'Search DN for data lookup in "<database>:(DN0),(DN1),..." format' EQUALITY
  caseIgnoreIA5Match
  SYNTAX IA5String)

( 1.3.6.1.4.1.42.2.27.5.1.25
  NAME 'SolarisSearchScope'
  DESC 'Scope to be used for search operations eg. "NS_LDAP_SCOPE_BASE",
       "NS_LDAP_SCOPE_ONELEVEL" or "NS_LDAP_SCOPE_SUBTREE"'
  EQUALITY caseIgnoreIA5Match
  SYNTAX IA5String SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.26
  NAME 'SolarisSearchTimeLimit'
  DESC 'Time Limit in seconds for search operations'
  EQUALITY integerMatch
  SYNTAX INTEGER SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.27
  NAME 'SolarisPreferredServer'
  DESC 'Preferred LDAP Server address or network number'
  EQUALITY caseIgnoreIA5Match
  SYNTAX IAString )
```

```
( 1.3.6.1.4.1.42.2.27.5.1.28
  NAME 'SolarisPreferredServerOnly'
  DESC 'Boolean flag for use of preferredServer or not'
  EQUALITY booleanMatch
  SYNTAX BOOLEAN SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.1.29
  NAME 'SolarisSearchReferral'
  DESC 'referral chasing option eg. "NS_LDAP_NOREF" or
       "NS_LDAP_FOLLOWREF"'
  EQUALITY caseIgnoreIA5Match
  SYNTAX IA5String SINGLE-VALUE )
```

#### 13.1.2 Objectclass clientProfile

```
( 1.3.6.1.4.1.42.2.27.5.2.7
  NAME 'SolarisNamingProfile'
  SUP top STRUCTURAL
  DESC 'Solaris LDAP Naming client profile objectClass'
  MUST ( cn $ SolarisLDAPServers $ SolarisSearchBaseDN )
  MAY ( SolarisBindDN $ SolarisBindPassword $ SolarisAuthMethod $
                                        SolarisTransportSecurity $ SolarisCertificatePath $
                                        SolarisCertificatePassword $ SolarisDataSearchDN $
                                        SolarisSearchScope $ SolarisSearchTimeLimit $
                                        SolarisPreferredServer $ SolarisPreferredServerOnly $
                                        SolarisCacheTTL $ SolarisSearchReferral ))
```

### 13.2 Version 2 profile

The second phase of the project adds support for a standards based profile schema which should provide interoperability between our implementation of NSS LDAP backend with HP-UX and Linux. The new profile is much more extensible and provides three key improvements.

- service search descriptor - This attribute provides the same functionality as SolarisDataSearchDN, but in addition it allows associating a search scope and a filter set with each RDN to be searched. This makes the DUA much more flexible and hence improves chances of achieving interoperability with other directory servers and/or deployments.
- schema mapping - The new profile provides a mechanism for configuring each service to allow mapping of its attributes and objectclasses to new ones. This feature allows deployments to reduce data redundancy and improves interoperability.
- security options - The profile now has a simpler to understand, but much more powerful security model. Additionally, it allows each service to override the default security configuration and hence providing a new level of configurability for various services.
- serviceCredentialLevel - The new profile enables services to be configured with different credential levels. Even though the Simplified LDAP API support this attribute, none of the backends currently support this feature.
- serviceAuthenticationMethod - The new profile enables services to be configured with different

authentication methods. The only services that currently supports this, are pam_ldap, passwd-cmd, and keyserv.

The new profile is available from IETF through:
http://tools.ietf.org/html/rfc4876

In addition to the above mentioned document, our implementation has an auxiliary objectclass with one additional configuration parameter which is specific to the operation of ldap_cachemgr. This additional parameter can be specified in the profile using this objectclass:

```
(  1.3.6.1.4.1.42.2.27.5.1.55
   NAME 'SolarisCachemgrServerListTTL'
   DESC 'Solaris ldap_cachemgr server list refresh time out value in seconds'
   EQUALITY integerMatch
   SYNTAX INTEGER SINGLE-VALUE )

( 1.3.6.1.4.1.42.2.27.5.2.10
  NAME 'SolarisDUAConfigProfileExtension'
  SUP top AUXILIARY
  DESC 'Solaris extension to DUAConfigProfile'
  MAY (  SolarisCachemgrServerListTTL ))
```

### 13.3 NIS Schema

The LDAP switch module was implemented according to RFC2307. However RFC2307 has a few problems and is missing a few definitions. These problems will be corrected in the revision of the RFC2307, currently in draft form.

The following is a partial list of the maps that are not currently defined in the RFC:

auto_* - does not handle case sensitivity.

hosts - does not handle multihome hosts and entries for IPv6 hosts.

mail_aliases - missing

publickey - missing

netid - missing

2307bis will handle these major issues as well as many other bug fixes. Additionally, the new document will specify detail implementation notes for implementing the NIS backends using the DUA profile. It is expected that the first draft of this document is submitted to IETF by June 2001.

LDAP servers should support this NIS schema, but since the schema used for naming lookup can be overridden using the schema mapping feature of the profile and the Simplified LDAP APIs, they can use other schema as long as all attributes required are accounted for and the data syntaxes are similar.