Plug-In API Reference

Sun™ ONE Directory Server

Version 5.2

816-6701-10
June 2003

Contents

About This Guide

Sun™ ONE Directory Server 5.2 is a powerful and scalable distributed directory server based on the industry-standard Lightweight Directory Access Protocol (LDAP). Sun ONE Directory Server software is part of the Sun Open Net Environment (Sun ONE), Sun's standards-based software vision, architecture, platform, and expertise for building and deploying Services On Demand.

Sun ONE Directory Server is the cornerstone for building a centralized and distributed data repository that can be used in your intranet, over your extranet with your trading partners, or over the public Internet to reach your customers.

Purpose of This Guide

This reference covers the data types and structures, functions and parameter block data that make up the public Sun ONE Directory Server plug-in API. Refer to it as you develop server plug-ins to extend Sun ONE Directory Server functionality.

Prerequisites

To make the most of this document, you must already be familiar with LDAP and with programming in the C language.

You should also have a clear understanding of the server extensions to develop, preferably including appropriate specifications.

For directory client programming, download the Sun ONE Directory Server Resource Kit. Refer to "Downloading Directory Server Tools," on page 8 for instructions.

Typographical Conventions

This section explains the typographical conventions used in this book.

`Monospaced font` - This typeface is used for literal text, such as the names of attributes and object classes when they appear in text. It is also used for URLs, filenames, and examples.

Italic font - This typeface is used for emphasis, for new terms, and for text that you must substitute for actual values, such as placeholders in path names.

The greater-than symbol (>) is used as a separator when naming an item in a menu or sub-menu. For example, Object > New > User means that you should select the User item in the New sub-menu of the Object menu.

NOTE Notes, Cautions, and Tips highlight important conditions or limitations. Be sure to read this information before continuing.

Default Paths and Filenames

All path and filename examples in the Sun ONE Directory Server product documentation are one of the following two forms:

- *ServerRoot*`/...` - The *ServerRoot* is the location of the Sun ONE Directory Server product. This path contains the shared binary files of Directory Server, Sun ONE Administration Server, and command line tools.

  The actual *ServerRoot* path depends on your platform, your installation, and your configuration. The default path depends on the product platform and packaging as shown in Table 1.

- *ServerRoot*`/slapd-`*serverID*`/...` - The *serverID* is the name of the Directory Server instance that you defined during installation or configuration. This path contains database and configuration files that are specific to the given instance.

NOTE Paths specified in this manual use the forward slash format of UNIX and commands are specified without file extensions. If you are using a Windows version of Sun ONE Directory Server, use the equivalent backslash format. Executable files on Windows systems generally have the same names with the `.exe` or `.bat` extension.

Table 1 Default *ServerRoot* Paths

Product Version	*ServerRoot* Path
Solaris Packages[1]	`/var/mps/serverroot` - After configuration, this directory contains links to the following locations: • `/etc/ds/v5.2` (static configuration files) • `/usr/admserv/mps/admin` (Sun ONE Administration Server binaries) • `/usr/admserv/mps/console` (Server Console binaries) • `/usr/ds/v5.2` (Directory Server binaries)
Compressed Archive Installation on Solaris and Other UNIX Systems	`/var/Sun/mps`
Zip Installation on Windows Systems	`C:\Program Files\Sun\MPS`

1. If you are working on the Solaris Operating Environment and are unsure which version of the Sun ONE Directory Server software is installed, check for the existence a key package such as `SUNWdsvu` using the `pkginfo` command. For example: `pkginfo | grep SUNWdsvu`.

Directory Server instances are located under *ServerRoot*`/slapd-`*serverID*`/`, where *serverID* represents the server identifier given to the instance on creation. For example, if you gave the name `dirserv` to your Directory Server, then the actual path would appear as shown in Table 2. If you have created a Directory Server instance in a different location, adapt the path accordingly.

Table 2 Default Example `dirserv` Instance Locations

Product Version	Instance Location
Solaris Packages	`/var/mps/serverroot/slapd-dirserv`
Compressed Archive Installation on Solaris and Other UNIX Systems	`/usr/Sun/mps/slapd-dirserv`
Zip Installation on Windows Systems	`C:\Program Files\Sun\MPS\slapd-dirserv`

Downloading Directory Server Tools

Some supported platforms provide native tools for accessing Directory Server. More tools for testing and maintaining LDAP directory servers, download the Sun ONE Directory Server Resource Kit (DSRK). This software is available at the following location:

```
http://wwws.sun.com/software/download/
```

Installation instructions and reference documentation for the DSRK tools is available in the *Sun ONE Directory Server Resource Kit Tools Reference*.

For developing directory client applications, you may also download the iPlanet Directory SDK for C and the iPlanet Directory SDK for Java from the same location.

Additionally, Java Naming and Directory Interface (JNDI) technology supports accessing the Directory Server using LDAP and DSML v2 from Java applications. Information about JNDI is available from:

```
http://java.sun.com/products/jndi/
```

The JNDI Tutorial contains detailed descriptions and examples of how to use JNDI. It is available at:

```
http://java.sun.com/products/jndi/tutorial/
```

Suggested Reading

Sun ONE Directory Server product documentation includes the following documents delivered in both HTML and PDF:

- *Sun ONE Directory Server Getting Started Guide* - Provides a quick look at many key features of Directory Server 5.2.
- *Sun ONE Directory Server Deployment Guide* - Explains how to plan directory topology, data structure, security, and monitoring, and discusses example deployments.
- *Sun ONE Directory Server Installation and Tuning Guide* - Covers installation and upgrade procedures, and provides tips for optimizing Directory Server performance.
- *Sun ONE Directory Server Administration Guide* - Gives the procedures for using the console and command-line to manage your directory contents and configure every feature of Directory Server.

- *Sun ONE Directory Server Reference Manual* - Details the Directory Server configuration parameters, commands, files, error messages, and schema.
- *Sun ONE Directory Server Plug-In API Programming Guide* - Demonstrates how to develop Directory Server plug-ins.
- *Sun ONE Directory Server Plug-In API Reference* - Details the data structures and functions of the Directory Server plug-in API.
- *Sun ONE Server Console Server Management Guide* - Discusses how to manage servers using the Sun ONE Administration Server and Java based console.
- *Sun ONE Directory Server Resource Kit Tools Reference* - Covers installation and features of the Sun ONE Directory Server Resource Kit, including many useful tools.

Other useful information can be found on the following Web sites:

- Product documentation online: `http://docs.sun.com/coll/S1_DirectoryServer_52`
- Sun software: `http://wwws.sun.com/software/`
- Sun ONE Services: `http://www.sun.com/service/sunps/sunone/`
- Sun Support Services: `http://www.sun.com/service/support/`
- Sun ONE for Developers: `http://sunonedev.sun.com/`
- Training: `http://suned.sun.com/`

Data Type and Structure Reference

This chapter covers data types and structures for use in plug-in functions. Table 1-1 summarizes the list of available data structures. Table 1-2 summarizes the list of callbacks.

Table 1-1 Quick Reference to Data Structures

Data Structure or Callback	Short Description
`berval`	Binary data encoded using Basic Encoding Rules
`computed_attr_context`	Information for use when handling computed attributes
`LDAPControl`	LDAP v3 control associated with an LDAP operation
`LDAPMod`	Set of modifications to a directory entry attribute
`Slapi_Attr`	Directory entry attribute
`Slapi_Backend`	Server backend
`Slapi_ComponentId`	Server assigned identifier, used for internal operations
`Slapi_CondVar`	Conditional variable for thread synchronization
`Slapi_Connection`	Client connection
`Slapi_DN`	Distinguished Name
`Slapi_Entry`	Directory entry
`Slapi_Filter`	Search filter
`Slapi_MatchingRuleEntry`	Matching rule handled by the plug-in
`Slapi_Mod`	A modification to an individual attribute
`Slapi_Mods`	Set of modifications to a directory entry
`Slapi_Mutex`	Mutex for thread synchronization
`Slapi_Operation`	Pending LDAP operation

Table 1-1 Quick Reference to Data Structures *(Continued)*

Data Structure or Callback	Short Description
`Slapi_PBlock`	Parameter block containing LDAP operation data
`Slapi_PluginDesc`	Plug-in description that you provide
`Slapi_RDN`	Relative Distinguished Name
`Slapi_Value`	Individual attribute value
`Slapi_ValueSet`	Set of values of an attribute
`vattr_type_thang`	Attribute that may be virtual

Table 1-2 Quick Reference to Callbacks

Data Structure or Callback	Short Description
`mrFilterMatchFn`	Handles an extensible match filter
`plugin_referral_entry_callback`	Handles referrals found by internal search
`plugin_result_callback`	Handles results sent after internal search
`plugin_search_entry_callback`	Handles entries found by internal search
`roles_get_scope_fn_type`	Determines scope of a role
`send_ldap_referral_fn_ptr_t`	Modifies referrals before sending them to a client
`send_ldap_result_fn_ptr_t`	Modifies a result before sending it to a client
`send_ldap_search_entry_fn_ptr_t`	Modifies an entry before sending it to a client
`slapi_compute_callback_t`	Handles a computed attribute
`slapi_compute_output_t`	Calculates a computed attribute
`slapi_extension_constructor_fnptr`	Handles object extension creation
`slapi_extension_destructor_fnptr`	Handles object extension destruction
`slapi_plugin_init_fnptr`	Handles recursive plug-in registration

berval

Represents binary data encoded using simplified Basic Encoding Rules (BER).

Use a `berval` structure when working with binary attribute values such as a JPEG or audio file.

Syntax

```
/* Defined in ldap.h, which is included by slapi-plugin.h */
#include "slapi-plugin.h"
struct berval {
    unsigned long   bv_len;
    char          * bv_val;
};
```

Fields

This structure has the following fields.

Table 1-3 `berval` Fields

Field	Description
`bv_len`	Size of the encoded data.
`bv_val`	Encoded data itself.

computed_attr_context

An opaque structure representing information such as attribute type, current BER-encoded request and parameter block context for attributes that are computed.

Syntax

```
#include "slapi-plugin.h"
typedef struct _computed_attr_context computed_attr_context;
```

Description

Before the server sends an entry to a client application, it determines whether any of the attributes are computed. If so, it generates those attributes and includes them in the entry.

As part of this process, the server creates a `computed_attr_context` structure to pass relevant information to functions generating the attribute values.

LDAPControl

Represents a client or server control associated with an LDAP operation as specified by LDAP v3.

Client and server controls may be used to extend the functionality of an LDAP operation.

Syntax

```
/* Defined in ldap.h, which is included by slapi-plugin.h */
#include "slapi-plugin.h"
typedef struct ldapcontrol {
    char          * ldctl_oid;
    struct berval   ldctl_value;
    char            ldctl_iscritical;
} LDAPControl, * PLDAPControl;
```

Fields

This structure has the following fields.

Table 1-4 `LDAPControl` Fields

Field	Description
`ldctl_oid`	Object identifier (OID) for the control.
`ldctl_value`	`berval` structure containing the value used by the control for the operation.
`ldctl_iscritical`	`LDAP_OPT_ON` indicates the control is critical to the operation. If the control is critical to the operation but not supported, the server returns `LDAP_UNAVAILABLE_CRITICAL_EXTENSION`. `LDAP_OPT_OFF` indicates the control is not critical.

LDAPMod

Represents a modification to an attribute in a directory entry.

Syntax

```
/* Defined in ldap.h, which is included by slapi-plugin.h */
#include "slapi-plugin.h"
typedef struct ldapmod {
    int                 mod_op;
    char              * mod_type;
    union {
        char          ** modv_strvals;
        struct berval ** modv_bvals;
    } mod_vals;
} LDAPMod;
#define mod_values  mod_vals.modv_strvals
#define mod_bvalues mod_vals.modv_bvals
```

Fields

This structure has the following fields.

Table 1-5 LDAPMod Fields

Field	Description
mod_op	Operation to perform and data type of attribute values. The mod_op field takes the following values specifying the operation to perform. • LDAP_MOD_ADD to add the attribute values to the entry. • LDAP_MOD_DELETE to remove the attribute values from the entry. • LDAP_MOD_REPLACE to replace existing attribute values. To specify binary values, bitwise OR the macro LDAP_MOD_BVALUES with the operation type identifier. `mod->mod_op = LDAP_MOD_ADD|LDAP_MOD_BVALUES;`
mod_type	Attribute type to modify.
mod_values	Pointer to NULL-terminated array of string values for the attribute.
mod_bvalues	Pointer to NULL-terminated array of berval structures for the attribute.

Example

The following code excerpt sets up an LDAPMod to change an mail address.

Code Example 1-1 Preparing Modifications to an Entry

```
#include "slapi-plugin.h"
/* Declare the appropriate structures. */
LDAPMod          mod_attr;         /* Attribute to modify */
LDAPMod        * mods[2];          /* Array of modifications */
char           * mail_vals[]  =    /* New mail address */
                     {"quentin@example.com", NULL};

/* Set up the LDAPMod structure used to modify the entry. */
mod_attr.mod_type   = "mail";
mod_attr.mod_op     = LDAP_MOD_REPLACE;
mod_attr.mod_values = mail_vals;   /* "quentin@example.com" */
mods[0]             = &mod_attr;
mods[1]             = NULL;
```

Code Example 1-1 Preparing Modifications to an Entry *(Continued)*

```
#include "slapi-plugin.h"

/* Modify the entry using slapi_modify_internal_set_pb()... */
```

mrFilterMatchFn

Specifies a filter matching callback function. The server calls this function when processing a search operation, once for each candidate that matches an extensible match search filter.

Syntax

```
#include "slapi-plugin.h"
typedef int (*mrFilterMatchFn) (void* filter, Slapi_Entry* entry,
    Slapi_Attr* vals);
```

Parameters

The callback takes the following parameters.

Table 1-6 `mrFilterMatchFn` Parameters

Parameter	Description
`filter`	Filter created by your filter factory function.
`entry`	`Slapi_Entry` representing the candidate entry checked by the server.
`vals`	`Slapi_Attr` representing the first attribute in the entry. Call `slapi_entry_next_attr()` to iterate through the rest of the attributes.

Description

The server calls this filter matching function when processing an extensible match filter using a matching rule plug-in. An extensible match filter specifies either the OID of a matching rule, or an attribute type, or both, that indicates how matching entries are found. For example, a sound-alike matching rule could be implemented to find all entries that sound like a given value.

To handle the extensible match filter for a matching rule, implement both this callback and a filter factor that creates the filter structure, `filter`. The callback retrieves information about the filter from this structure, such as the attribute type and value, then compares this information with attribute types and values in the candidate `entry`.

Returns

The callback must return `0` if the filter matches, `-1` if the filter does not match. On error, it may return an LDAP error code as specified in the *Result Message* section of RFC 2251, *Lightweight Directory Access Protocol (v3)*.

plugin_referral_entry_callback

Specifies a referral callback function. The server calls this function when the internal search implemented to trigger this callback finds LDAP v3 referrals.

Syntax

```
#include "slapi-plugin.h"
typedef int (*plugin_referral_entry_callback)(char * referral,
    void *callback_data);
```

Parameters

The callback takes the following parameters.

Table 1-7 `plugin_referral_entry_callback` Parameters

Parameter	Description
`referral`	URL of a reference found by the search.
`callback_data`	Pointer passed to the internal search operation. Use this to pass your own data between the callback function and the body of the plug-in.

Description

Pass this as the `prec` parameter of `slapi_search_internal_callback_pb()`. Each time the internal search finds a referral entry, it calls this function.

Returns

The server does not use the callback return value.

plugin_result_callback

Specifies a search result callback function. The server calls this function when the internal search implemented to trigger this callback returns an LDAP result.

Syntax

```
#include "slapi-plugin.h"
typedef void (*plugin_result_callback)(int rc, void *callback_data);
```

Parameters

The callback takes the following parameters.

Table 1-8 `plugin_result_callback` Parameters

Parameter	Description
`rc`	LDAP result code returned by the search.
`callback_data`	Pointer passed to the internal search operation. Use this to pass your own data between the callback function and the body of the plug-in.

Description

Pass this as the `prc` parameter of `slapi_search_internal_callback_pb()`. The server calls this function once for each search operation, unless the search is abandoned, in which case the function is not called.

plugin_search_entry_callback

Specifies an entry callback function. The server calls this function when the internal search implemented to trigger this callback finds an LDAP entry.

Syntax

```
#include "slapi-plugin.h"
typedef int (*plugin_search_entry_callback)(Slapi_Entry *e,
    void *callback_data);
```

Parameters

The callback takes the following parameters.

Table 1-9 `plugin_search_entry_callback` Parameters

Parameter	Description
`e`	Pointer to the entry found by the search.
`callback_data`	Pointer passed to the internal search operation. Use this to pass your own data between the callback function and the body of the plug-in.

Description

Pass this as the `psec` parameter of `slapi_search_internal_callback_pb()`. Each time the internal search finds a referral entry, it calls this function.

Returns

The callback must return `0` to continue the search, `-1` to interrupt the search.

roles_get_scope_fn_type

Specifies a callback function to determine the role of a scope. The plug-in triggers this callback using `slapi_role_get_scope()`.

Syntax

```
#include "slapi-plugin.h"
typedef int (*roles_get_scope_fn_type)(Slapi_Entry *role_entry,
    Slapi_DN ***scope, int *nb_scope);
```

Parameters

The callback takes the following parameters.

Table 1-10 `roles_get_scope_fn_type` Parameters

Parameter	Description
`role_entry`	Entry defining the role
`scope`	Set this to the Distinguished Name of the entry at the base of the scope
`nb_scope`	Set this to a value such as `LDAP_SCOPE_BASE`, `LDAP_SCOPE_ONELEVEL`, or `LDAP_SCOPE_SUBTREE`

Description

This callback determines the role of a scope identified by `role_entry`. Register the callback with the server using `slapi_register_role_get_scope()`.

Returns

The callback returns `0` if successful, `-1` otherwise.

send_ldap_referral_fn_ptr_t

Specifies a callback triggered before the server sends a result to the client.

Syntax

```
#include "slapi-plugin.h"
typedef int (*send_ldap_referral_fn_ptr_t)( Slapi_PBlock *pb,
    Slapi_Entry *e, struct berval **refs, struct berval ***urls );
```

Parameters

The callback takes the following parameters.

Table 1-11 `send_ldap_referral_fn_ptr_t` Parameters

Parameter	Description
`pb`	Current parameter block for the operation.
`e`	Current entry for the operation.
`refs`	Pointer to the NULL-terminated array of `berval` structures containing the LDAP v3 referrals (search result references) found in the entry.
`urls`	Pointer to the array of `berval` structures used to collect LDAP referrals for LDAP v2 clients.

Description

This callback lets you modify referrals returned to the client. Register the callback with the server using `slapi_search_internal_callback_pb()`.

Returns

The callback should return `0` if successful, `-1` otherwise.

send_ldap_result_fn_ptr_t

Specifies a callback triggered before the server sends a result to the client.

Syntax

```
#include "slapi-plugin.h"
typedef void (*send_ldap_result_fn_ptr_t)( Slapi_PBlock *pb,
    int err, char *matched, char *text, int nentries,
    struct berval **urls );
```

Parameters

The callback takes the following parameters.

Table 1-12 `send_ldap_result_fn_ptr_t` Parameters

Parameter	Description
`pb`	Current parameter block for the operation.
`err`	Result code.
`matched`	When sending back an `LDAP_NO_SUCH_OBJECT` result code, use this argument to specify the portion of the target DN that could be matched. (Pass `NULL` in other situations.)
`text`	Error message that you want sent back to the client. (Pass NULL if you do not want an error message sent back.)
`nentries`	When sending back the result code for an LDAP search operation, use this argument to specify the number of matching entries found.
`urls`	When sending back an `LDAP_PARTIAL_RESULTS` result code to an LDAP v2 client or an `LDAP_REFERRAL` result code to an LDAP v3 client, use this argument to specify the array of `berval` structures containing the referral URLs. (Pass `NULL` in other situations.)

Description

This callback lets you modify the result returned to the client. Register the callback with the server using `slapi_search_internal_callback_pb()`.

Returns

The callback should return `0` if successful, `-1` otherwise.

send_ldap_search_entry_fn_ptr_t

Specifies a callback triggered before the server sends an entry returned by a search to the client.

Syntax

```
#include "slapi-plugin.h"
typedef int (*send_ldap_search_entry_fn_ptr_t)( Slapi_PBlock *pb,
    Slapi_Entry *e, LDAPControl **ectrls, char **attrs,
    int attrsonly );
```

Parameters

The callback takes the following parameters.

Table 1-13 `send_ldap_search_entry_fn_ptr_t` Parameters

Parameter	Description
`pb`	Current parameter block for the operation.
`e`	Entry returned by the search.
`ectrls`	Array of controls for the operation.
`attrs`	Array of attribute types to return in the search results.
`attrsonly`	• 0 to return both values and types • 1 to return only attribute types

Description

This callback lets you modify what is returned to the client. Register the callback with the server using `slapi_search_internal_callback_pb()`.

Returns

The callback should return `0` if successful, `-1` otherwise.

Slapi_Attr

Opaque structure representing a directory entry attribute.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_attr Slapi_Attr;
```

See Also

Slapi_Entry

Slapi_Backend

Opaque structure representing a server backend.

Syntax

```
#include "slapi-plugin.h"
typedef struct backend Slapi_Backend;
```

See Also

“Handling Slapi_Backend Structures,” on page 48

Slapi_ComponentId

Opaque structure representing a component identifier used by the server to identify the plug-in when processing internal operations.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_componentid Slapi_ComponentId;
```

Example

The following code excerpt sets a plug-in component identifier used during an internal search.

Code Example 1-2 Setting a Plug-In Identifier for Use with Internal Operations

```
#include "slapi-plugin.h"
/* Declare the identifier as global to the plug-in. */
static Slapi_ComponentId * postop_id;

/* Assign a value as part of plug-in initialization. */
int
testpostop_init(Slapi_PBlock * pb)
{
    int rc = 0;
    /* Register description, other functions, and so forth. */
    rc |= slapi_pblock_set(
        pb,
        SLAPI_PLUGIN_START_FN,
        (void *) testpostop_set_log
    );
    /* Assign a value to the identifier. */
```

Code Example 1-2 Setting a Plug-In Identifier for Use with Internal Operations *(Continued)*

```
#include "slapi-plugin.h"
    rc |= slapi_pblock_get(pb, SLAPI_PLUGIN_IDENTITY, &postop_id);
    return (rc);
}

/* The server uses the identifier when processing
 * internal operations, such as an internal search. */
int
testpostop_set_log(Slapi_PBlock * pb)
{
    Slapi_DN    * confdn = NULL;       /* DN for configuration entry */
    Slapi_Entry * config = NULL;       /* Configuration entry */
    int rc = 0;

    confdn = slapi_sdn_new_dn_byval("cn=config");
    rc |= slapi_search_internal_get_entry(confdn, NULL, &config, postop_id);
    /* Use the results of the search. */
    return(rc);
}
```

slapi_compute_callback_t

Specifies a function to determine which function should be called to provide a value for a computed attribute.

Syntax

```
typedef int (*slapi_compute_callback_t)(computed_attr_context *c,
    char* type,Slapi_Entry *e,slapi_compute_output_t outputfn);
```

Parameters

The callback takes the following parameters.

Table 1-14 `slapi_compute_callback_t` Parameters

Parameter	Description
`c`	Context of the callback.
`type`	Attribute type for which the `outputfn` computes values.
`e`	Entry including computed attributes, returned to client after this callback returns.
`outputfn`	Callback to compute the attribute value.

Description

This callback selects the function that provides a computed value for an attribute of type `type` on an entry `e`, given context `c`. The server calls this function to get a a function for calculating attribute values before returning the entry `e` to the client having requested the operation.

This function is registered with the server using `slapi_compute_add_evaluator()` in the plug-in initialization function.

Returns

This function should return `0` on success. It should return `-1` if does not handle the attribute type passed in `type`. Otherwise, it should return an LDAP error code.

slapi_compute_output_t

Specifies a callback function to determine the value of a computed attribute.

Syntax

```
typedef int (*slapi_compute_output_t)(computed_attr_context *c,
    Slapi_Attr *a , Slapi_Entry *e);
```

Parameters

The callback takes the following parameters.

Table 1-15 `slapi_compute_output_t` Parameters

Parameter	Description
c	Context of the callback.
a	Attribute for which the callback computes values.
e	Entry including computed attributes, returned to client after this callback returns.

Description

This callback provides a computed value for attribute `a` of entry `e`, given context `c`. The `slapi_compute_callback_t` function you register using `slapi_compute_add_evaluator()` calls this function to compute a value for `a` before returning the entry `e` to the server.

Returns

This function should return `0` on success. It should return `-1` if does not handle the attribute type passed in `a`. Otherwise, it should return an LDAP error code.

Slapi_CondVar

Opaque structure representing a conditional variable used by the plug-in to handle thread synchronization.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_condvar Slapi_CondVar;
```

See Also

`slapi_destroy_condvar()`

`slapi_notify_condvar()`

`slapi_new_condvar()`

`slapi_wait_condvar()`

Slapi_Connection

Opaque structure representing a connection to the server.

Syntax

```
#include "slapi-plugin.h"
typedef struct conn Slapi_Connection;
```

Slapi_DN

Opaque structure representing a Distinguished Name (DN). The structure retains the original DN and can also hold a normalized version after `slapi_sdn_get_ndn()` is called.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_dn Slapi_DN;
```

See Also

"Handling DN Strings," on page 39 and "Handling Slapi_DN Structures," on page 49

Slapi_Entry

Opaque structure representing a directory entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_entry Slapi_Entry;
```

See Also

`Slapi_Attr`

“Handling Entries,” on page 39

slapi_extension_constructor_fnptr

Specifies a callback function for an object extension.

Syntax

```
#include "slapi-plugin.h"
typedef void *(*slapi_extension_constructor_fnptr)
    (void *object, void *parent);
```

Parameters

The callback takes the following parameters.

Table 1-16 `slapi_extension_constructor_fnptr` Parameters

Parameter	Description
`object`	Extended object
`parent`	Parent object for the extension

Description

This callback registers an object extension that can be retrieved using `slapi_get_object_extension()`.

Ensure that this callback only creates the object extension if it does not already exist.

The callback is registered with the server using `slapi_register_object_extension()` as part of the actual plug-in initialization function.

Returns

This callback returns a pointer to the extension. Otherwise it returns `NULL`.

slapi_extension_destructor_fnptr

Specifies a callback function to free memory allocated for an object extension.

Syntax

```
#include "slapi-plugin.h"
typedef void (*slapi_extension_destructor_fnptr)(void *extension,
    void *object, void *parent);
```

Parameters

The callback takes the following parameters.

Table 1-17 `slapi_extension_destructor_fnptr` Parameters

Parameter	Description
`extension`	Object extension
`object`	Extended object
`parent`	Parent for the extension

Description

This callback releases memory allocated for an object extension. The function is registered with the server using `slapi_register_object_extension()` in the plug-in initialization function.

Slapi_Filter

Opaque structure representing a search filter.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_filter Slapi_Filter;
```

See Also

"Handling Filters," on page 42

Slapi_MatchingRuleEntry

Opaque structure representing an LDAP v3 matching rule handled by the plug-in.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_matchingRuleEntry Slapi_MatchingRuleEntry;
```

See Also

"Handling Matching Rules," on page 44

Slapi_Mod

Opaque structure representing a modification of a directory entry attribute.

Parameter blocks use `LDAPMod` structures rather than `Slapi_Mod` structures. The latter type is provided as a convenience for plug-ins dealing extensively with modifications.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_mod Slapi_Mod;
```

See Also

`LDAPMod`

"Handling Modification Structures," on page 44

Slapi_Mods

Opaque structure representing a set of modifications to a directory entry.

Parameter blocks use arrays of `LDAPMod` structures rather than `Slapi_Mods` structures. The latter type is provided as a convenience for plug-ins dealing extensively with modifications.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_mods Slapi_Mods;
```

See Also

`LDAPMod`

"Handling Modification Structures," on page 44

Slapi_Mutex

Opaque structure representing a mutex lock used by the plug-in.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_mutex Slapi_Mutex;
```

See Also

`slapi_destroy_mutex()`

`slapi_lock_mutex()`

`slapi_new_mutex()`

`slapi_unlock_mutex()`

Slapi_Operation

Opaque structure representing a pending operation from an LDAP client.

The structure records, among other data, the type of operation requested.

Syntax

```
#include "slapi-plugin.h"
typedef struct op Slapi_Operation;
```

See Also

`slapi_op_abandoned()`, `slapi_op_get_type()`

Slapi_PBlock

Opaque structure representing, called a parameter block, containing name-value pairs updated in the context of a server operation.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_pblock Slapi_PBlock;
```

Description

For most types of plug-in functions, the server passes in a parameter block (`Slapi_PBlock`) including data relevant to the operation being processed. Access the data using `slapi_pblock_get()`.

Plug-in initialization functions register at least the plug-in API version, plug-in descriptions, and other plug-in functions using `slapi_pblock_set()`.

The specific parameters available in a `Slapi_PBlock` structure depend on the type of plug-in function and the context of the LDAP operation. Refer to the "Parameter Block Reference," on page 335 for details on the name-value pairs available to specific types of plug-in functions.

See Also

"Accessing and Modifying Parameter Blocks," on page 35

For examples of `Slapi_PBlock` use, refer to the sample plug-ins under *`ServerRoot`*`/plugins/slapd/slapi/examples/`.

Slapi_PluginDesc

Represents a plug-in description you provide to identify your plug-in.

The plug-in initialization function must register this information with the server.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_plugindesc {
    char * spd_id;
    char * spd_vendor;
    char * spd_version;
    char * spd_description;
} Slapi_PluginDesc;
```

Fields

This structure has the following fields.

Table 1-18 `Slapi_PluginDesc` Fields

Field	Description
`spd_id`	Unique (server wide) identifier for the plug-in.
`spd_vendor`	Name of the vendor supplying the plug-in such as `Sun Microsystems, Inc.`
`spd_version`	Plug-in revision number such as `5.2`, not to be confused with `SLAPI_PLUGIN_VERSION`, which specifies the plug-in API version supported by the plug-in.

Table 1-18 `Slapi_PluginDesc` Fields *(Continued)*

`spd_description`	Short description of the plug-in such as `Sample post-operation plug-in`.

See Also

For examples of `Slapi_PluginDesc` use, refer to the sample plug-ins under *`ServerRoot`*`/plugins/slapd/slapi/examples/`.

slapi_plugin_init_fnptr

Specifies a callback function for registering other plug-ins.

Syntax

```
#include "slapi-plugin.h"
typedef int (*slapi_plugin_init_fnptr)( Slapi_PBlock *pb );
```

Parameters

The callback takes the following parameter.

Table 1-19 `slapi_plugin_init_fnptr` Parameter

Parameter	Description
`pb`	Parameter block passed to the initialization function.

Description

This callback mimics a plug-in initialization function, permitting one plug-in to register other plug-ins. The function is registered with the server using `slapi_register_plugin()` as part of the actual plug-in initialization function.

Returns

This callback returns `0` on success. Otherwise, it returns `-1`.

See Also

For examples of plug-in initialization functions, refer to the sample plug-ins under *`ServerRoot`*`/plugins/slapd/slapi/examples/`.

Slapi_RDN

Opaque structure representing a Relative Distinguished Name (RDN), the part of the DN that differentiates the entry from other entries having the same parent.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_rdn Slapi_RDN;
```

See Also

“Handling Slapi_RDN Structures,” on page 51

Slapi_Value

Opaque structure representing an individual attribute value.

Use `Slapi_ValueSet` instead when handling all the values of an attribute at once.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_value Slapi_Value;
```

See Also

“Handling Slapi_Value Structures,” on page 52

Slapi_ValueSet

Opaque structure representing the set of values of an attribute.

Use `Slapi_Value` instead when handling an individual attribute value.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_value_set Slapi_ValueSet;
```

See Also

“Handling Slapi_ValueSet Structures,” on page 53

vattr_type_thang

Opaque structure representing both real and virtual attributes of an entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct _vattr_type_thang vattr_type_thang;
```

See Also

“Handling Virtual Attributes,” on page 55

Chapter 2

Function Reference

This chapter contains a reference to the public functions for writing plug-ins. Along with a detailed description of each function, the function reference details the function header file and syntax, the function parameters, the function return value, and possible memory concerns.

The beginning of this chapter lists the functions by functional area.

Functions by Functional Area

This section categorizes plug-in functions by functional area.

Accessing and Modifying Parameter Blocks

Function	Description
`slapi_pblock_destroy()`	Frees a pblock from memory.
`slapi_pblock_get()`	Gets the value from a parameter block.
`slapi_pblock_new()`	Creates a new parameter block.
`slapi_pblock_set()`	Sets the value of a parameter block.

Allocating and Freeing Memory

Function	Description
`slapi_ch_array_free()`	Frees an existing array.
`slapi_ch_bvdup()`	Makes a copy of an existing `berval` structure.
`slapi_ch_bvecdup()`	Makes a copy of an array of existing `berval` structures.
`slapi_ch_calloc()`	Allocates space for an array of a number of elements of a specified size.
`slapi_ch_free()`	Frees space allocated by the `slapi_ch_malloc()`, `slapi_ch_realloc()`, and `slapi_ch_calloc()` functions.
`slapi_ch_free_string()`	Frees an existing string.
`slapi_ch_malloc()`	Allocates space in memory.
`slapi_ch_realloc()`	Changes the size of a block of allocated memory.
`slapi_ch_strdup()`	Makes a copy of an existing string.

Handling Access Control

Function	Description
`slapi_access_allowed()`	Determines if the user requesting the current operation has the access rights to perform an operation on a given entry, attribute, or value.
`slapi_acl_check_mods()`	Determines if a user has the rights to perform the specified modifications on an entry.
`slapi_acl_verify_aci_syntax()`	Determines whether or not the access control items (ACIs) on an entry are valid.

Handling Attributes

Function	Description
slapi_attr_add_value()	Adds a value to an attribute.
slapi_attr_basetype()	Returns the base type of an attribute.
slapi_attr_dup()	Duplicates an attribute.
slapi_attr_first_value()	Gets the first value of an attribute.
slapi_attr_flag_is_set()	Determines if certain flags are set.
slapi_attr_free()	Frees an attribute.
slapi_attr_get_bervals_copy()	Puts the values contained in an attribute into an array of berval structures.
slapi_attr_get_flags()	Gets the flags associated with an attribute.
slapi_attr_get_numvalues()	Puts the count of values of an attribute into an integer.
slapi_attr_get_oid_copy()	Searches for an attribute type and gives its OID string.
slapi_attr_get_type()	Gets the name of the attribute type.
slapi_attr_get_valueset()	Copies attribute values into a valueset.
slapi_attr_init()	Initializes an empty attribute.
slapi_attr_new()	Creates a new attribute.
slapi_attr_next_value()	Gets the next value of an attribute.
slapi_attr_syntax_normalize()	Returns a copy of the normalized attribute types.
slapi_attr_type2plugin()	Gets information about the plug-in responsible for handling an attribute type.
slapi_attr_type_cmp()	Compares two attributes.
slapi_attr_types_equivalent()	Compares two attribute names to determine if they represent the same attribute.
slapi_attr_value_cmp()	Compares two attribute values.

Function	Description
slapi_attr_value_find()	Determines if an attribute contains a given value.

Handling BER Values

Function	Description
slapi_berval_cmp()	Compares two berval structures.
slapi_ch_bvdup()	Makes a copy of an existing berval structure.
slapi_ch_bvecdup()	Makes a copy of an array of existing berval structures.

Handling Controls

Function	Description
slapi_build_control()	Creates an LDAPControl structure based on a BerElement, an OID, and a criticality flag.
slapi_get_supported_controls_copy()	Retrieves an allocated array of object identifiers (OIDs) representing the controls supported by the Directory Server.
slapi_control_present()	Determines whether or not the specified object identification (OID) identifies a control that is present in a list of controls.
slapi_dup_control()	Makes an allocated copy of an LDAPControl.
slapi_register_supported_control()	Registers the specified control with the server. This function associates the control with an object identification (OID).

Handling DN Strings

Function	Description
`slapi_dn_beparent()`	Gets a copy of the DN of the parent of an entry.
`slapi_dn_ignore_case()`	Converts all characters in a DN to lowercase.
`slapi_dn_isbesuffix()`	Determines if a DN is the suffix of the local database.
`slapi_dn_isparent()`	Determines if a DN is the parent of a specific DN.
`slapi_dn_isroot()`	Determines if a DN is the root DN for the local database.
`slapi_dn_issuffix()`	Determines if a DN is equal to a specified suffix.
`slapi_dn_normalize()`	Converts a DN to canonical format.
`slapi_dn_normalize_case()`	Converts a DN to canonical format and all characters to lower case.
`slapi_dn_normalize_to_end()`	Normalizes part of a DN value.
`slapi_dn_parent()`	Gets the DN of the parent of an entry.
`slapi_dn_plus_rdn()`	Adds an RDN to a DN.

Handling Entries

Function	Description
`slapi_entry2str()`	Generates an LDIF string description.
`slapi_entry2str_with_options()`	Generates an LDIF string descriptions with options.
`slapi_entry_add_rdn_values()`	Add components in an entry's RDN.
`slapi_entry_add_string()`	Adds a string value to an attribute in an entry.
`slapi_entry_add_value()`	Adds a data value to an attribute in an entry.

Function	Description
slapi_entry_add_valueset()	Adds a data value to an attribute in an entry.
slapi_entry_alloc()	Allocates memory for a new entry.
slapi_entry_attr_delete()	Deletes an attribute from an entry.
slapi_entry_attr_find()	Checks if an entry contains a specific attribute.
slapi_entry_attr_get_charptr()	Gets the first value as a string.
slapi_entry_attr_get_int()	Gets the first value as an integer.
slapi_entry_attr_get_long()	Gets the first value as a long.
slapi_entry_attr_get_uint()	Gets the first value as an unsigned integer.
slapi_entry_attr_get_ulong()	Gets the first value as an unsigned long.
slapi_entry_attr_hasvalue()	Checks if an attribute in an entry contains a value.
slapi_entry_attr_merge_sv()	Adds an array to the attribute values in an entry.
slapi_entry_attr_set_charptr()	Replaces the values of an attribute with a string.
slapi_entry_attr_set_int()	Replaces the value of an attribute with an integer.
slapi_entry_attr_set_long()	Replaces the value of an attribute with a long.
slapi_entry_attr_set_uint()	Replaces the value of an attribute with an unsigned integer.
slapi_entry_attr_set_ulong()	Replaces the value of an attribute with an unsigned long.
slapi_entry_delete_string()	Deletes a string from an attribute.
slapi_entry_delete_values_sv()	Removes a Slapi_Value array from an attribute.
slapi_entry_dup()	Copies an entry, its DN, and its attributes.
slapi_entry_first_attr()	Finds the first attribute in an entry.
slapi_entry_free()	Frees an entry from memory.
slapi_entry_get_dn()	Gets the DN from an entry.
slapi_entry_get_dn_const()	Returns the DN of an entry as a constant.

Function	Description
slapi_entry_get_ndn()	Returns the NDN of an entry.
slapi_entry_get_sdn()	Returns the Slapi_DN from an entry.
slapi_entry_get_sdn_const()	Returns a Slapi_DN from an entry as a constant.
slapi_entry_get_uniqueid()	Gets the unique ID from an entry.
slapi_entry_has_children()	Determines if the specified entry has child entries.
slapi_entry_init()	Initializes the values of an entry.
slapi_entry_merge_values_sv()	Adds an array of data values to an attribute in an entry.
slapi_entry_next_attr()	Finds the next attribute in an entry.
slapi_entry_rdn_values_present()	Checks if values present in an entry's RDN are also present as attribute values.
slapi_entry_schema_check()	Determines if an entry complies with the schema for its object class.
slapi_entry_set_dn()	Sets the DN of an entry.
slapi_entry_set_sdn()	Sets the Slapi_DN value in an entry.
slapi_entry_size()	Returns the size of an entry.
slapi_is_rootdse()	Determines if an entry is the root DSE.
slapi_str2entry()	Converts an LDIF description into an entry.

Handling Extended Operations

Function	Description
slapi_get_supported_extended_ops_copy()	Gets a copy of the object IDs (OIDs) of the extended operations.

Handling Filters

Function	Description
`slapi_filter_compare()`	Determines if two filters are identical.
`slapi_filter_free()`	Frees the specified filter.
`slapi_filter_get_attribute_type()`	Gets the attribute type for all simple filter choices.
`slapi_filter_get_ava()`	Gets the attribute type and the value from the filter.
`slapi_filter_get_choice()`	Gets the type of the specified filter.
`slapi_filter_get_subfilt()`	Gets the substring values from the filter.
`slapi_filter_get_type()`	Gets the attribute type specified in the filter.
`slapi_filter_join()`	Joins two specified filters.
`slapi_filter_list_first()`	Gets the first filter that makes up the specified filter.
`slapi_filter_list_next()`	Gets the next filter.
`slapi_filter_test()`	Determines if the specified entry matches a particular filter.
`slapi_filter_test_ext()`	Determines if an entry matches a given filter.
`slapi_filter_test_simple()`	Determines if an entry matches a filter.
`slapi_str2filter()`	Converts a string description of a search filter into a filter of the `Slapi_Filter` type.

Handling Internal Operations

Function	Description
`slapi_add_entry_internal_set_pb()`	Prepare a `Slapi_PBlock` structure for an internal add operation involving a `Slapi_Entry` structure.

Function	Description
slapi_add_internal_pb()	Adds an LDAP add operation based on a parameter block to add a new directory entry.
slapi_add_internal_set_pb()	Prepare a Slapi_PBlock structure for an internal add operation.
slapi_delete_internal_pb()	Performs an LDAP delete operation based on a parameter block to remove a directory entry
slapi_delete_internal_set_pb()	Prepare a Slapi_PBlock structure for an internal delete operation.
slapi_free_search_results_internal()	Frees search results.
slapi_modify_internal_pb()	Performs an LDAP modify operation based on a parameter block to modify a directory entry.
slapi_modify_internal_set_pb()	Prepare a Slapi_PBlock structure for an internal modify operation.
slapi_modrdn_internal_pb()	Performs an LDAP modify RDN operation based on a parameter block to rename a directory entry.
slapi_rename_internal_set_pb()	Prepare a Slapi_PBlock structure for an internal modify RDN operation.
slapi_search_internal_callback_pb()	Performs an LDAP search operation based on a parameter block to search the directory.
slapi_search_internal_get_entry()	Performs an internal search operation to read one entry
slapi_search_internal_pb()	Performs an LDAP search operation based on a parameter block to search the directory.
slapi_search_internal_set_pb()	Prepare a Slapi_PBlock structure for an internal search operation.

Handling Matching Rules

Function	Description
slapi_matchingrule_free()	Free a Slapi_MatchingRuleEntry after registering the matching rule.
slapi_matchingrule_get()	Access a Slapi_MatchingRuleEntry.
slapi_matchingrule_new()	Allocate a Slapi_MatchingRuleEntry structure.
slapi_matchingrule_register()	Register a matching rule with the server.
slapi_matchingrule_set()	Modify a Slapi_MatchingRuleEntry.
slapi_mr_filter_index()	Call a matching rule filter index function.
slapi_mr_indexer_create()	Get a pointer to the indexer factory function for a matching rule.

Handling Modification Structures

Function	Description
slapi_entry2mods()	Creates an array of LDAPMod from a Slapi_Entry.
slapi_mod_add_value()	Adds a value to a Slapi_Mod structure.
slapi_mod_done()	Frees internals of Slapi_Mod structure.
slapi_mod_dump()	Dumps the contents of an LDAPMod to the server log.
slapi_mod_free()	Frees a Slapi_Mod structure.
slapi_mod_get_first_value()	Initializes a Slapi_Mod iterator and returns the first attribute value.
slapi_mod_get_ldapmod_byref()	Gets a reference to the LDAPMod in a Slapi_Mod structure.
slapi_mod_get_ldapmod_passout()	Retrieves the LDAPMod contained in a Slapi_Mod structure.
slapi_mod_get_next_value()	Increments the Slapi_Mod iterator and returns the next attribute value.

Function	Description
slapi_mod_get_num_values()	Gets the number of values in a Slapi_Mod structure.
slapi_mod_get_operation()	Gets the operation type of Slapi_Mod structure.
slapi_mod_get_type()	Gets the attribute type of a Slapi_Mod structure.
slapi_mod_init()	Initializes a Slapi_Mod structure.
slapi_mod_init_byref()	Initializes a Slapi_Mod structure that is a wrapper for an existing LDAPMod.
slapi_mod_init_byval()	Initializes a Slapi_Mod structure with a copy of an LDAPMod.
slapi_mod_init_passin()	Initializes a Slapi_Mod from an LDAPMod.
slapi_mod_isvalid()	Determines whether a Slapi_Mod structure is valid.
slapi_mod_new()	Allocates a new Slapi_Mod structure.
slapi_mod_remove_value()	Removes the value at the current Slapi_Mod iterator position.
slapi_mod_set_operation()	Sets the operation type of a Slapi_Mod structure.
slapi_mod_set_type()	Sets the attribute type of a Slapi_Mod.
slapi_mods2entry()	Creates a Slapi_Entry from an array of LDAPMod.
slapi_mods_add()	Appends a new mod with a single attribute value to Slapi_Mods structure.
slapi_mods_add_ldapmod()	Appends an LDAPMod to a Slapi_Mods structure.
slapi_mods_add_mod_values()	Appends a new mod to a Slapi_Mods structure, with attribute values provided as an array of Slapi_Value.
slapi_mods_add_modbvps()	Appends a new mod to a Slapi_Mods structure, with attribute values provided as an array of berval.
slapi_mods_add_smod()	Appends a Slapi_Mod to a Slapi_Mods structure.

Function	Description
slapi_mods_add_string()	Appends a new mod to Slapi_Mods structure with a single attribute value provided as a string.
slapi_mods_done()	Frees internals of a Slapi_Mods structure.
slapi_mods_dump()	Dumps the contents of a Slapi_Mods structure to the server log.
slapi_mods_free()	Frees a Slapi_Mods structure.
slapi_mods_get_first_mod()	Initializes a Slapi_Mods iterator and returns the first LDAPMod.
slapi_mods_get_first_smod()	Initializes a Slapi_Mods iterator and returns the first mod wrapped in a Slapi_Mods structure.
slapi_mods_get_ldapmods_byref()	Gets a reference to the array of LDAPMod in a Slapi_Mods structure.
slapi_mods_get_ldapmods_passout()	Retrieves the array of LDAPMod contained in a Slapi_Mods structure.
slapi_mods_get_next_mod()	Increments the Slapi_Mods iterator and returns the next LDAPMod.
slapi_mods_get_next_smod()	Increments the Slapi_Mods iterator and returns the next mod wrapped in a Slapi_Mods.
slapi_mods_get_num_mods()	Gets the number of mods in a Slapi_Mods structure.
slapi_mods_init()	Initializes a Slapi_Mods.
slapi_mods_init_byref()	Initializes a Slapi_Mods that is a wrapper for an existing array of LDAPMod.
slapi_mods_init_passin()	Initializes a Slapi_Mods structure from an array of LDAPMod.
slapi_mods_insert_after()	Inserts an LDAPMod into a Slapi_Mods structure after the current iterator position.
slapi_mods_insert_at()	Inserts an LDAPMod anywhere in a Slapi_Mods.

Function	Description
slapi_mods_insert_before()	Inserts an LDAPMod into a Slapi_Mods structure before the current iterator position.
slapi_mods_insert_smod_at()	Inserts a Slapi_Mod anywhere in a Slapi_Mods.
slapi_mods_insert_smod_before()	Inserts a Slapi_Mod into a Slapi_Mods structure before the current iterator position.
slapi_mods_iterator_backone()	Decrements the Slapi_Mods current iterator position.
slapi_mods_new()	Allocates a new uninitialized Slapi_Mods structure.
slapi_mods_remove()	Removes the mod at the current Slapi_Mods iterator position.

Handling Operations

Function	Description
slapi_op_abandoned()	Determines if the client has abandoned the current operation.
slapi_op_get_type()	Gets the type of a Slapi_Operation.

Handling Passwords

Function	Description
slapi_pw_find_sv()	Determines whether or not a specified password matches one of the encrypted values of an attribute.
slapi_pw_find_valueset()	Determines whether or not a specified password matches one of the encrypted values of an attribute.

Handling Roles

Function	Description
`slapi_register_role_get_scope()`	Register a callback to determine the scope of a role.
`slapi_role_check()`	Checks if the entry pointed to contains the role indicated.
`slapi_role_get_scope()`	Determine the scope of a role.

Handling SASL Bind Mechanisms

Function	Description
`slapi_get_supported_saslmechanisms_copy()`	Gets an array of the names of the supported Simple Authentication and Security Layer (SASL) mechanisms.
`slapi_register_supported_saslmechanism()`	Registers the specified Simple Authentication and Security Layer (SASL) mechanism with the server.

Handling Slapi_Backend Structures

Function	Description
`slapi_be_exist()`	Checks if the backend that contains the specified DN exists.
`slapi_be_get_name()`	Returns the name of the specified backend.
`slapi_be_get_readonly()`	Indicates if the database associated with the backend is in read-only mode.
`slapi_be_getsuffix()`	Returns the n+1 suffix associated with the specified backend.

Function	Description
slapi_be_gettype()	Returns the type of the backend.
slapi_be_is_flag_set()	Checks if a flag is set in the backend configuration.
slapi_be_issuffix()	Verifies that the specified suffix matches a registered backend suffix.
slapi_be_logchanges()	Indicates if the changes applied to the backend should be logged in the changelog.
slapi_be_private()	Verifies if the backend is private.
slapi_be_select()	Finds the backend that should be used to service the entry with the specified DN.
slapi_be_select_by_instance_name()	Find the backend used to service the database.
slapi_get_first_backend()	Returns a pointer of the backend structure of the first backend.
slapi_get_next_backend()	Returns a pointer to the next backend.
slapi_is_root_suffix()	Checks if a suffix is a root suffix of the DIT.

Handling Slapi_DN Structures

Function	Description
slapi_moddn_get_newdn()	Builds the new DN of an entry.
slapi_sdn_compare()	Compares two DNs.
slapi_sdn_copy()	Copies a DN.
slapi_sdn_done()	Clears a Slapi_DN structure.
slapi_sdn_dup()	Duplicates a Slapi_DN structure.
slapi_sdn_free()	Frees a Slapi_DN structure.
slapi_sdn_get_backend_parent()	Gets the DN of the parent within a specific backend.
slapi_sdn_get_dn()	Gets the DN from a Slapi_DN structure.

Function	Description
slapi_sdn_get_ndn()	Gets the normalized DN of a Slapi_DN structure.
slapi_sdn_get_ndn_len()	Gets the length of the normalized DN of a Slapi_DN structure.
slapi_sdn_get_parent()	Get the parent DN of a given Slapi_DN structure.
slapi_sdn_get_rdn()	Gets the RDN from an NDN.
slapi_sdn_isempty()	Checks if there is a DN value stored in a Slapi_DN structure.
slapi_sdn_isgrandparent()	Checks if a DN is the parent of the parent of a DN.
slapi_sdn_isparent()	Checks if a DN is the parent of a DN.
slapi_sdn_issuffix()	Checks if a Slapi_DN structure contains a suffix of another.
slapi_sdn_new()	Allocates new Slapi_DN structure.
slapi_sdn_new_dn_byref()	Creates a new Slapi_DN structure pointing to an existing DN string.
slapi_sdn_new_dn_byval()	Creates a new Slapi_DN structure copying an existing DN string.
slapi_sdn_new_dn_passin()	Creates a new Slapi_DN structure pointing to a new copy of a DN string.
slapi_sdn_new_ndn_byref()	Creates a new Slapi_DN structure pointing to an existing normalized DN.
slapi_sdn_new_ndn_byval()	Creates a new Slapi_DN structure copying an existing normalized DN.
slapi_sdn_scope_test()	Checks if an entry is in the scope of a certain base DN.
slapi_sdn_set_dn_byref()	Sets a DN value in a Slapi_DN structure pointing to an existing DN string.
slapi_sdn_set_dn_byval()	Sets a DN value in a Slapi_DN structure copying an existing DN string.
slapi_sdn_set_dn_passin()	Sets a DN value in a Slapi_DN structure pointing to a new copy of a DN string.

Function	Description
slapi_sdn_set_ndn_byref()	Sets a normalized DN in a Slapi_DN structure pointing to an existing normalized DN string.
slapi_sdn_set_ndn_byval()	Sets a normalized DN in a Slapi_DN structure copying an existing normalized DN string.
slapi_sdn_set_parent()	Sets a new parent in an entry.
slapi_sdn_set_rdn()	Sets a new RDN for an entry.

Handling Slapi_RDN Structures

Function	Description
slapi_rdn_add()	Adds a new RDN to an existing RDN structure.
slapi_rdn_compare()	Compares two RDNs.
slapi_rdn_contains()	Checks if a Slapi_RDN structure holds any RDN matching a given type/value pair.
slapi_rdn_contains_attr()	Checks if a Slapi_RDN structure contains any RDN matching a given type.
slapi_rdn_done()	Clears a Slapi_RDN structure.
slapi_rdn_free()	Frees a Slapi_RDN structure.
slapi_rdn_get_first()	Gets the type/value pair of the first RDN.
slapi_rdn_get_index()	Gets the index of the RDN.
slapi_rdn_get_index_attr()	Gets the position and the attribute value of the first RDN.
slapi_rdn_get_next()	Gets the RDN type/value pair from the RDN.
slapi_rdn_get_num_components()	Gets the number of RDN type/value pairs.
slapi_rdn_get_rdn()	Gets the RDN from a Slapi_RDN structure.
slapi_rdn_init()	Initializes a Slapi_RDN structure with NULL values.

Function	Description
slapi_rdn_init_dn()	Initializes a Slapi_RDN structure from an existing DN string.
slapi_rdn_init_rdn()	Initializes a Slapi_RDN structure from an existing Slapi_RDN structure.
slapi_rdn_init_sdn()	Initializes a Slapi_RDN structure from an existing Slapi_SDN structure.
slapi_rdn_isempty()	Checks if an RDN value is stored in a Slapi_RDN structure.
slapi_rdn_new()	Creates a new Slapi_RDN structure.
slapi_rdn_new_dn()	Creates a new Slapi_RDN structure.
slapi_rdn_new_rdn()	Creates a new Slapi_RDN structure.
slapi_rdn_new_sdn()	Creates a new Slapi_RDN structure.

Handling Slapi_Value Structures

Function	Description
slapi_value_compare()	Compares two values.
slapi_value_dup()	Duplicates a value.
slapi_value_free()	Frees a Slapi_Value structure from memory.
slapi_value_get_berval()	Gets the berval structure of the value.
slapi_value_get_int()	Converts the value of an integer.
slapi_value_get_length()	Gets the length of a value.
slapi_value_get_long()	Converts a value into a long integer.
slapi_value_get_string()	Returns the value as a string.
slapi_value_get_uint()	Converts the value into an unsigned integer.
slapi_value_get_ulong()	Converts the value into an unsigned long.
slapi_value_init()	Initializes a Slapi_Value structure with no values.

Function	Description
slapi_value_init_berval()	Initializes a Slapi_Value structure from the berval structure.
slapi_value_init_string()	Initializes a Slapi_Value structure from a string.
slapi_value_init_string_passin()	Initializes a Slapi_Value structure with a value contained in a string.
slapi_value_new()	Allocates a new Slapi_Value structure.
slapi_value_new_berval()	Allocates a new Slapi_Value structure from a berval structure.
slapi_value_new_string()	Allocates a new Slapi_Value structure from a string.
slapi_value_new_string_passin()	Allocates a new Slapi_Value structure and initializes it from a string.
slapi_value_new_value()	Allocates a new Slapi_Value from another Slapi_Value structure.
slapi_value_set()	Sets the value.
slapi_value_set_berval()	Copies the value from a berval structure into a Slapi_Value structure.
slapi_value_set_int()	Sets the integer value of a Slapi_Value structure.
slapi_value_set_string()	Copies a string into the value.
slapi_value_set_string_passin()	Sets the value.
slapi_value_set_value()	Copies the value of a Slapi_Value structure into another Slapi_Value structure.

Handling Slapi_ValueSet Structures

Function	Description
slapi_valueset_add_value()	Adds a Slapi_Value in the Slapi_ValueSet structure.

Function	Description
slapi_valueset_count()	Returns the number of values contained in a Slapi_ValueSet structure.
slapi_valueset_done()	Frees the values contained in the Slapi_ValueSet structure.
slapi_valueset_find()	Finds the value in a valueset using the syntax of an attribute.
slapi_valueset_first_value()	Gets the first value of a Slapi_ValueSet structure.
slapi_valueset_free()	Frees the specified Slapi_ValueSet structure and its members from memory.
slapi_valueset_init()	Resets a Slapi_ValueSet structure to no values.
slapi_valueset_new()	Allocates a new Slapi_ValueSet structure.
slapi_valueset_next_value()	Gets the next value from a Slapi_ValueSet structure.
slapi_valueset_set_from_smod()	Copies the values of Slapi_Mod structure into a Slapi_ValueSet structure.
slapi_valueset_set_valueset()	Initializes a Slapi_ValueSet structure from another Slapi_ValueSet structure.

Handling UTF8

Function	Description
slapi_has8thBit()	Checks if a string has an 8-bit character.
slapi_UTF8CASECMP()	Compares two UTF8 strings.
slapi_UTF8NCASECMP()	Compares a specified number of UTF8 characters.
slapi_UTF8ISLOWER()	Verifies if a UTF8 character is lower case.
slapi_UTF8ISUPPER()	Verifies if a single UTF8 character is upper case.

Function	Description
slapi_UTF8STRTOLOWER()	Converts a UTF8 string to lower case.
slapi_UTF8STRTOUPPER()	Converts a string made up of UTF8 characters and converts it to upper case.
slapi_UTF8TOLOWER()	Converts a UTF8 character to lower case.
slapi_UTF8TOUPPER()	Converts a lower case UTF8 character to an upper case character.

Logging

Function	Description
slapi_log_error_ex()	Writes an error message to the server error log
slapi_log_info_ex()	Writes an informational message to the server error log
slapi_log_warning_ex()	Writes a warning message to the server error log

Handling Virtual Attributes

Function	Description
slapi_vattr_value_compare()	Compares attribute type and name in a given entry.
slapi_vattr_values_free()	Frees the valueset and type names.
slapi_vattr_values_get_ex()	Returns the values for an attribute type from an entry.

Sending Entries, Referrals, and Results

Function	Description
slapi_send_ldap_referral()	Processes an entry's LDAP v3 referrals.

Function	Description
`slapi_send_ldap_result()`	Sends a result code back to the client.
`slapi_send_ldap_search_entry()`	Sends an entry found by a search back to the client.

Function Descriptions

slapi_access_allowed()

Determines if the user requesting the current operation has the access rights to perform an operation on a given entry, attribute, or value.

Syntax

```
#include "slapi-plugin.h"
int slapi_access_allowed( Slapi_PBlock *pb, Slapi_Entry *e,
    char *attr, struct berval *val, int access );
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block passed into this function.
`e`	Entry for which you want to check the access rights.
`attr`	Attribute for which you want to check the access rights.
`val`	Pointer to the `berval` structure containing the value for which you want to check the access rights.
`access`	Type of access rights that you want to check. For example, to check for write access, pass `SLAPI_ACL_WRITE` as the value of this argument.

The value of the `access` argument can be one of the following:

Argument	Description
`SLAPI_ACL_ADD`	Permission to add a specified entry.

SLAPI_ACL_COMPARE	Permission to compare the specified values of an attribute in an entry.
SLAPI_ACL_DELETE	Permission to delete a specified entry.
SLAPI_ACL_READ	Permission to read a specified attribute.
SLAPI_ACL_SEARCH	Permission to search on a specified attribute or value.
SLAPI_ACL_WRITE	Permission to write a specified attribute or value or permission to rename a specified entry.

Returns

This function returns one of the following values:

- LDAP_SUCCESS if the user has the specified rights to the entry, attribute, or value.
- LDAP_INSUFFICIENT_ACCESS if the user does not have the specified rights to the entry, attribute, or value.
- If a problem occurs during processing, the function will return one of the following error codes:

Return Code	Description
LDAP_OPERATIONS_ERROR	An error occurred while executing the operation. This error can occur if, for example, the type of access rights specified are not recognized by the server. In other words, you did not pass a value from the previous table.
LDAP_INVALID_SYNTAX	Invalid syntax was specified. This error can occur if the ACL associated with an entry, attribute, or value uses the wrong syntax.
LDAP_UNWILLING_TO_PERFORM	The DSA (this Directory Server) is unable to perform the specified operation. This error can occur if, for example, you are requesting write access to a read-only database.

Description

Call this function to determine if a user has access rights to a specified entry, attribute, or value. The function performs this check for users who request the operation that invokes this plug-in.

For example, suppose you are writing a pre-operation plug-in for the add operation. You can call this function to determine if users have the proper access rights before they can add an entry to the directory.

As part of the process of determining if the user has access rights, this function does the following:

- Checks if the user requesting the operation is the root DN.

  If so, the function returns `LDAP_SUCCESS`. (The root DN has permission to perform any operation.)

- Gets information about the operation being requested, the connection to the client, and the back-end database where directory information is stored.

  - If for some reason the function cannot determine which operation is being requested, the function returns `LDAP_OPERATIONS_ERROR`.
  - If no connection to a client exists — in other words, if the request for the operation was made by the server or its backend — the function returns `LDAP_SUCCESS`. (The server and its backend are not restricted by access control lists.)
  - If the back-end database is read-only and the request is checking for write access (`SLAPI_ACL_WRITE`), the function returns `LDAP_UNWILLING_TO_PERFORM`.

- Determines if the user requesting the operation is attempting to modify his or her own entry.

ACLs can be set up to allow users the rights to modify their own entries. The `slapi_access_allowed()` function checks for this condition.

The caller must ensure that the backend specified in the parameter block is set prior to calling this function. For example:

```
be = slapi_be_select(slapi_entry_get_sdn_const(seObjectEntry));
if (NULL == be) {
    cleanup("backend selection failed for entry: \"%s\"\n", szObjectDN);
    slapi_send_ldap_result(pb,LDAP_NO_SUCH_OBJECT,NULL,"Obj not found",0,NULL);
    return(SLAPI_PLUGIN_EXTENDED_SENT_RESULT);
```

```
    }

slapi_pblock_set(pb, SLAPI_BACKEND, be);
nAccessResult = slapi_access_allowed(pb,seObjectEntry,"*",bval,SLAPI_ACL_DELETE);
```

slapi_acl_check_mods()

Determines if a user has the rights to perform the specified modifications on an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_acl_check_mods( Slapi_PBlock *pb, Slapi_Entry *e,
    LDAPMod **mods, char **errbuf );
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block passed into this function.
`e`	Entry for which you want to check the access rights.
`mods`	Array of `LDAPMod` structures that represent the modifications to be made to the entry.
`errbuf`	Pointer to a string containing an error message if an error occurs during the processing of this function.

Returns

Returns one of the following values:

- `LDAP_SUCCESS` if the user has write permission to the values in the specified attributes.
- `LDAP_INSUFFICIENT_ACCESS` if the user does not have write permission to the values of the specified attribute.
- If a problem occurs during processing, the function will return one of the following error codes:

Argument	Description

LDAP_OPERATIONS_ERROR	An error occurred while executing the operation.
LDAP_INVALID_SYNTAX	Invalid syntax was specified. This error can occur if the ACL associated with an entry, attribute, or value uses the wrong syntax.
LDAP_UNWILLING_TO_PERFORM	The DSA (this directory server) is unable to perform the specified operation. This error can occur if, for example, you are requesting write access to a read-only database.

Description

Call this function to determine if a user has access rights to modify the specified entry. The function performs this check for users who request the operation that invokes this plug-in.

For example, if you are writing a database plug-in, you can call this function to determine if users have the proper access rights before they can add, modify, or delete entries from the database.

As part of the process of determining if the user has access rights, the `slapi_access_allowed()` function does the following:

- Checks to access control for the directory is disabled.

  If access control is disabled, the function returns `LDAP_SUCCESS`.

- For each value in each attribute specified in the `LDAPMod` array, the function determines if the user has permissions to write to that value. Specifically, the function calls `slapi_access_allowed()` with `SLAPI_ACL_WRITE` as the access right to check.

  - If for some reason the function cannot determine which operation is being requested, the function returns `LDAP_OPERATIONS_ERROR`.
  - If no connection to a client exists (in other words, if the request for the operation was made by the server or its backend), the function returns `LDAP_SUCCESS`. (The server and its backend are not restricted by access control lists.)
  - If the backend database is read-only and the request is checking for write access (`SLAPI_ACL_WRITE`), the function returns `LDAP_UNWILLING_TO_PERFORM`.

Memory Concerns

You must free the `errbuf` buffer by calling `slapi_ch_free()` when you are finished using the error message.

See Also

`slapi_access_allowed()`

`slapi_ch_free()`

slapi_acl_verify_aci_syntax()

Determines whether or not the access control items (ACIs) on an entry are valid.

Syntax

```
#include "slapi-plugin.h"
int slapi_acl_verify_aci_syntax (Slapi_Entry *e,
    char **errbuf);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry for which you want to check the ACIs.
`errbuf`	Pointer to the error message returned if the ACI syntax is invalid.

Returns

This function returns `0` if successful, or `-1` if an error occurs.

Memory Concerns

You must free the `errbuf` buffer by calling `slapi_ch_free()` when you are finished using the error message.

See Also

`slapi_ch_free()`

slapi_add_entry_internal_set_pb()

Prepares a parameter block for an internal add operation involving a `Slapi_Entry` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_add_entry_internal_set_pb(Slapi_PBlock *pb,
    Slapi_Entry *e, LDAPControl **controls,
    Slapi_ComponentId *plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block for the internal add operation
`e`	Entry to add
`controls`	Array of controls to request for the add operation; `NULL` if no controls
`plugin_identity`	Plug-in identifier obtained from `SLAPI_PLUGIN_IDENTITY` during plug-in initialization
`operation_flags`	Bitmask of flags such as `SLAPI_OP_FLAG_NEVER_CHAIN`

Returns

This function returns `0` if successful. Otherwise it returns an LDAP error code.

Description

This function prepares a parameter block for use with `slapi_add_internal_pb()` using the entry.

Memory Concerns

Allocate the parameter block using `slapi_pblock_new()` before calling this function.

The entry is consumed during the call to `slapi_add_internal_pb()`.

Free the parameter block after calling `slapi_add_internal_pb()`.

See Also

`slapi_add_internal_pb()`

`slapi_add_internal_set_pb()`

`slapi_pblock_new()`

slapi_add_internal_pb()

Performs an internal add operation of a new directory entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_add_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

Parameter	Description
`pb`	A parameter block that has been initialized using `slapi_add_internal_set_pb()`.

Returns

This function returns `-1` if the parameter passed is a `NULL` pointer. Otherwise, it returns `0`.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

Description

This function performs an internal add operation based on a parameter block. The parameter block should be initialized by calling `slapi_add_internal_set_pb()` or `slapi_add_entry_internal_set_pb()`.

Memory Concerns

None of the parameters that are passed to `slapi_add_internal_set_pb()` or `slapi_add_entry_internal_set_pb()` are altered or consumed by this function. The entry parameter that is passed to `slapi_add_entry_internal_set_pb()` is consumed by a successful call to this function.

slapi_add_internal_set_pb()

Prepares a parameter block for an internal add operation.

Syntax

```
#include "slapi-plugin.h"
int slapi_add_internal_set_pb(Slapi_PBlock *pb, const char *dn,
    LDAPMod **attrs, LDAPControl **controls,
    Slapi_ComponentId *plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block for the internal add operation
dn	Distinguished Name of the entry to add
attrs	Array of attributes of the entry to add
controls	Array of controls to request for the add operation
plugin_identity	Plug-in identifier obtained from SLAPI_PLUGIN_IDENTITY during plug-in initialization
operation_flags	Flags as SLAPI_OP_FLAG_NEVER_CHAIN

Returns

This function returns 0 if successful. Otherwise, it returns an LDAP error code.

Description

This function prepares a parameter block for use with slapi_add_internal_pb() using the components of the entry.

If the entry has already been prepared as a Slapi_Entry structure, use slapi_add_entry_internal_set_pb() instead.

Memory Concerns

Allocate the parameter block using slapi_pblock_new() before calling this function.

Free the parameter block after calling slapi_add_internal_pb().

See Also

slapi_add_entry_internal_set_pb()

slapi_add_internal_pb()

slapi_pblock_new()

slapi_attr_add_value()

Adds a value to an attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_add_value(Slapi_Attr *a, const Slapi_Value *v);
```

Parameters

This function takes the following parameters:

Parameter	Description
a	The attribute that will contain the values.
v	Values to be added to the attribute.

Returns

This function always returns `0`.

See Also

`slapi_attr_first_value()`

`slapi_attr_next_value()`

`slapi_attr_get_numvalues()`

`slapi_attr_value_cmp()`

`slapi_attr_value_find()`

slapi_attr_basetype()

Returns the base type of an attribute.

Syntax

```
#include "slapi-plugin.h"
char *slapi_attr_basetype( char *type, char *buf, size_t bufsiz );
```

Parameters

This function takes the following parameters:

Parameter	Description

type	Attribute type from which you wish to get the base type.
buf	Buffer to hold the returned base type.
bufsiz	Size of the buffer.

Returns

This function returns NULL if the base type fits in the buffer. If the base type is longer than the buffer, the function allocates memory for the base type and returns a pointer to it.

Description

This function returns the base type of an attribute (for example, if given cn;lang-jp, returns cn).

Memory Concerns

You should free the returned base type when you are finished by calling slapi_ch_free().

See Also

slapi_attr_type2plugin()

slapi_attr_get_type()

slapi_attr_type_cmp()

slapi_attr_types_equivalent()

slapi_attr_dup()

Duplicates an attribute.

Syntax

```
#include "slapi-plugin.h"
Slapi_Attr *slapi_attr_dup(const Slapi_Attr *attr);
```

Parameters

This function takes the following parameter:

Parameter	Description
attr	The attribute to be duplicated.

Returns
This function returns the newly created copy of the attribute.

Description
Use this function to make a copy of an attribute.

Memory Concerns
You must free the returned attribute using `slapi_attr_free()`.

See Also
`slapi_attr_new()`

`slapi_attr_init()`

`slapi_attr_free()`

slapi_attr_first_value()
Gets the first value of an attribute.

Syntax
```
#include "slapi-plugin.h"
int slapi_attr_first_value( Slapi_Attr *a, Slapi_Value **v );
```

Parameters
This function takes the following parameters:

Parameter	Description
a	Attribute containing the desired value.
v	Holds the first value of the attribute.

Returns
This function returns one of the following values:

- `0`, which is the index of the first value.
- `-1` if NULL.

Description
Use this function to get the first value of an attribute. This is part of a set of functions to enumerate over an `Slapi_Attr` structure.

See Also

`slapi_attr_next_value()`

`slapi_attr_get_numvalues()`

slapi_attr_flag_is_set()

Determines if certain flags are set for a particular attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_flag_is_set( Slapi_Attr *attr, unsigned long flag );
```

Parameters

This function takes the following parameters:

Parameter	Description
`attr`	Attribute that you want to check.
`flag`	Flag that you want to check in the attribute.

The value of the `flag` argument can be one of the following:

Argument	Description
`SLAPI_ATTR_FLAG_SINGLE`	Flag that determines if the attribute is single-valued.
`SLAPI_ATTR_FLAG_OPATTR`	Flag that determines if the attribute is an operational attribute.
`SLAPI_ATTR_FLAG_READONLY`	Flag that determines if the attribute is read-only.

Returns

This function returns one of the following values:

- `1` if the specified flag is set.
- `0` if the specified flag is not set.

Description

This function determines if certain flags are set for a particular attribute. These flags can identify an attribute as a single-valued attribute, an operational attribute, or as a read-only attribute.

See Also

`slapi_attr_get_flags()`

slapi_attr_free()

Frees an attribute.

Syntax

```
#include "slapi-plugin.h"
void slapi_attr_free( Slapi_Attr **a );
```

Parameters

This function takes the following parameters:

Parameter	Description
a	Attribute to be freed.

Description

Use this function to free an attribute when you are finished with it.

See Also

`slapi_attr_new()`

`slapi_attr_init()`

`slapi_attr_dup()`

slapi_attr_get_bervals_copy()

Puts the values contained in an attribute into an array of `berval` structures.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_bervals_copy(Slapi_Attr *a,
    struct berval ***vals );
```

Parameters

This function takes the following parameters:

Parameter	Description
a	Attribute that contains the desired values.
vals	Pointer to an array of berval structure pointers to hold the desired values.

Returns

This function returns one of the following values:

- 0 if values are found.
- -1 if NULL.

Description

This function copies the values from an attribute into an array of berval structure pointers.

Memory Concerns

Free this array using ber_bvecfree(3LDAP).

slapi_attr_get_flags()

Gets the flags associated with the specified attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_flags( Slapi_Attr *attr, unsigned long *flags );
```

Parameters

This function takes the following parameters:

Parameter	Description
attr	Attribute for which you wish to get the flags.
flags	When you call slapi_attr_get_flags(), this parameter is set to a pointer to the flags of the specified attribute. Do not free the flags; the flags are part of the actual data in the attribute, not a copy of the data.

To determine which flags have been set, you can use bitwise AND on the value of the `flags` argument with one or more of the following:

Arguments	Description
`SLAPI_ATTR_FLAG_SINGLE`	Flag that determines if the attribute is single-valued.
`SLAPI_ATTR_FLAG_OPATTR`	Flag that determines if the attribute is an operational attribute.
`SLAPI_ATTR_FLAG_READONLY`	Flag that determines if the attribute is read-only.

Returns

This function returns `0` if successful.

Description

This function gets the flags associated with the specified attribute. These flags can identify an attribute as a single-valued attribute, an operational attribute, or as a read-only attribute.

See Also

`slapi_attr_flag_is_set()`

slapi_attr_get_numvalues()

Puts the count of values of an attribute into a provided integer.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_numvalues( const Slapi_Attr *a, int *numValues);
```

Parameters

This function takes the following parameters:

Parameter	Description
`a`	Attribute containing the values to be counted.
`numValues`	Integer to hold the counted values.

Returns

This function always returns `0`.

Description

This function counts the number of values in an attribute and places that count in an integer.

See Also

`slapi_attr_first_value()`

`slapi_attr_next_value()`

slapi_attr_get_oid_copy()

Searches the syntaxes for an attribute type, and returns a copy of its OID string.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_oid_copy( const Slapi_Attr *attr, char **oidp );
```

Parameters

This function takes the following parameters:

Parameter	Description
`attr`	Attribute that contains the desired type.
`oidp`	Destination string of the copied attribute type OID.

Returns

This function returns one of the following values:

- `0` if the attribute type is found.
- `-1` if it is not.

Description

Use this function to search the syntaxes for an attribute type's OID.

Memory Concerns

You should free this string using `slapi_ch_free()`.

slapi_attr_get_type()

Gets the name of the attribute type from a specified attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_type( Slapi_Attr *attr, char **type );
```

Parameters

This function takes the following parameters:

Parameter	Description
`attr`	Attribute of which you wish to get the type.
`type`	When you call this function, this parameter is set to a pointer to the type of the specified attribute. Do not free this attribute type; the type is part of the actual data in the attribute, not a copy of the data.

Returns

This function returns `0` if successful.

See Also

`slapi_attr_type2plugin()`

`slapi_attr_type_cmp()`

`slapi_attr_types_equivalent()`

`slapi_attr_basetype()`

slapi_attr_get_valueset()

Copies existing values contained in an attribute into a valueset.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_valueset(const Slapi_Attr *a,
    Slapi_ValueSet **vs);
```

Parameters

This function takes the following parameters:

Parameter	Description
a	Attribute containing the values to be placed into a valueset. This must be a valid attribute, not NULL.
vs	Receives values from the first parameter.

Returns

This function always returns 0.

See Also

slapi_entry_add_valueset()

slapi_valueset_new()

slapi_valueset_free()

slapi_valueset_init()

slapi_valueset_done()

slapi_valueset_add_value()

slapi_valueset_first_value()

slapi_valueset_next_value()

slapi_valueset_count()

slapi_attr_init()

Initializes an empty attribute with a base type.

Syntax

```
#include "slapi-plugin.h"
Slapi_Attr *slapi_attr_init(Slapi_Attr *a, const char *type);
```

Parameters

This function takes the following parameters:

Parameter	Description
a	The empty attribute to be initialized.
type	Attribute type to be initialized.

Returns
This function returns the newly initialized attribute.

Description
Use this function to initialize an empty attribute with an attribute type.

See Also
`slapi_attr_new()`

`slapi_attr_free()`

`slapi_attr_dup()`

slapi_attr_new()
Creates a new attribute.

Syntax
```
#include "slapi-plugin.h"
Slapi_Attr *slapi_attr_new( void );
```

Parameters
This function takes no parameters.

Returns
This function returns the newly created attribute

Description
Use this function to create an empty attribute.

See Also
`slapi_attr_free()`

`slapi_attr_dup()`

slapi_attr_next_value()
Gets the next value of an attribute.

Syntax
```
#include "slapi-plugin.h"
int slapi_attr_next_value( Slapi_Attr *a, int hint,
    Slapi_Value **v );
```

Parameters

This function takes the following parameters:

Parameter	Description
a	Attribute contained the desired value.
hint	Index of the value to be returned.
v	Holds the value of the attribute.

Returns

This function returns one of the following values:

- `hint` plus `1` if the value is found
- `-1` if `NULL`, or if a value at `hint` is not found

Description

Use this function to get the next value of an attribute. The value of an attribute associated with an index is placed into a value. This is part of a set of functions to enumerate over a `Slapi_Attr` structure.

See Also

`slapi_attr_first_value()`

`slapi_attr_get_numvalues()`

slapi_attr_syntax_normalize()

Searches for an attribute type in the syntaxes, and returns a copy of the normalized attribute types.

Syntax

```
#include "slapi-plugin.h"
char * slapi_attr_syntax_normalize( const char *s );
```

Parameters

This function takes the following parameter:

Parameter	Description
s	Attribute type for which you wish to search.

Returns

This function returns the copy of the desired normalized attribute, or a normalized copy of what was passed in.

Description

Use this function to search the syntaxes for an attribute type and return its normalized form.

Memory Concerns

You should free the returned string using `slapi_ch_free()`.

See Also

`slapi_ch_free()`

slapi_attr_type2plugin()

Gets a pointer to information about the syntax plug-in responsible for handling the specified attribute type.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_type2plugin( char *type, void **pi );
```

Parameters

This function takes the following parameters:

Parameter	Description
`type`	Type of attribute for which you wish to get the plug-in.
`pi`	Pointer to the plug-in structure.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if the corresponding plug-in is not found.

Description

Syntax plug-ins are plug-ins that you can write to index and search for specific attribute types.

See Also

`slapi_attr_get_type()`

`slapi_attr_type_cmp()`

`slapi_attr_types_equivalent()`

`slapi_attr_basetype()`

slapi_attr_type_cmp()

Compares two attribute types to determine if they are the same.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_type_cmp( char *t1, char *t2, int opt );
```

Parameters

This function takes the following parameters:

Parameter	Description
`t1`	Name of the first attribute type that you want to compare.
`t2`	Name of the second attribute type that you want to compare.
`opt`	One of the following values: • `0` - Compare the types as is. • `1` - Compare only the base names of the types (for example, if the type is `cn;lang-en`, the function compares only the `cn` part of the type). • `2` - Ignore any options in the second type that are not in the first type. For example, if the first type is `cn` and the second type is `cn;lang-en`, the `lang-en` option in the second type is not part of the first type. In this case, the function considers the two types to be the same.

Returns

This function returns 0 if the type names are equal, or a non-zero value if they are not equal.

See Also

`slapi_attr_type2plugin()`

`slapi_attr_get_type()`

`slapi_attr_types_equivalent()`

`slapi_attr_basetype()`

slapi_attr_types_equivalent()

Compares two attribute names to determine if they represent the same attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_types_equivalent( const char *t1, const char *t2 );
```

Parameters

This function takes the following parameters:

Parameter	Description
`t1`	Pointer to the first attribute type that you want to compare.
`t2`	Pointer to the second attribute type that you want to compare.

Returns

This function returns the one of the following values:

- 1 if `t1` and `t2` represent the same attribute.
- 0 if `t1` and `t2` do not represent the same attribute.

See Also

`slapi_attr_add_value()`

`slapi_attr_first_value()`

`slapi_attr_next_value()`

`slapi_attr_get_numvalues()`

`slapi_attr_value_find()`

slapi_attr_value_cmp()

Compares two values for a given attribute to determine if they are equal.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_value_cmp( Slapi_Attr *attr, struct berval *v1,
    struct berval *v2 );
```

Parameters

This function takes the following parameters:

Parameter	Description
`attr`	Attribute used to determine how these values are compared (for example, if the attribute contains case-insensitive strings, the strings are compared without regard to case).
`v1`	Pointer to the `berval` structure containing the first value that you want to compare.
`v2`	Pointer to the `berval` structure containing the second value that you want to compare.

Returns

This function returns one of the following values:

- `0` if the values are equal.
- `-1` if they are not equal.

See Also

`slapi_attr_add_value()`

`slapi_attr_first_value()`

`slapi_attr_next_value()`

`slapi_attr_get_numvalues()`

`slapi_attr_value_find()`

slapi_attr_value_find()

Determines if an attribute contains a given value.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_value_find( Slapi_Attr *a, struct berval *v );
```

Parameters

This function takes the following parameters:

Parameter	Description
a	Attribute that you wish to check.
v	Pointer to the `berval` structure containing the value for which you wish to search.

Returns

This function returns one of the following values:

- `0` if the attribute contains the specified value.
- `-1` if the attribute does not contain the specified value.

See Also

`slapi_attr_add_value()`

`slapi_attr_first_value()`

`slapi_attr_next_value()`

`slapi_attr_get_numvalues()`

`slapi_attr_value_cmp()`

slapi_be_exist()

Checks if the backend that contains the specified DN exists.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_exist(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
sdn	Pointer to the DN in the backends for which you are looking.

Returns

This function returns one of the following values:

- 1 if the backend containing the specified DN exists.
- 0 if the backend does not exist.

See Also

`slapi_be_select()`

slapi_be_get_name()

Returns the name of the specified backend.

Syntax

```
#include "slapi-plugin.h"
char * slapi_be_get_name(Slapi_Backend * be);
```

Parameters

This function takes the following parameter:

Parameter	Description
be	Pointer to the structure containing the backend configuration.

Returns

This function returns the name associated to the specified backend.

Memory Concerns

You should not free the returned pointer.

slapi_be_get_readonly()

Indicates if the database associated with the backend is in read-only mode.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_get_readonly(Slapi_Backend *be);
```

Parameters

This function takes the following parameter:

Parameter	Description
be	Pointer to the structure containing the backend configuration.

Returns

This function returns one of the following values:

- `0` if the database is not in read-only mode.
- `1` if the database is in read-only mode.

slapi_be_getsuffix()

Returns the n+1 suffix associated with the specified backend.

Syntax

```
#include "slapi-plugin.h"
const Slapi_DN *slapi_be_getsuffix(Slapi_Backend *be, int n);
```

Parameters

This function takes the following parameters:

Parameter	Description
be	Pointer to the structure containing the backend configuration.
n	Index.

Returns

This function returns the DN of the suffix if it exists, or `NULL` if there is no n+1 suffix in the backend.

Description

This function returns the n+1 suffix associated with the specified backend. This function is still present for compatibility purposes with previous versions of the Directory Server Plug-In API.

Memory Concerns

You should not free the returned pointer.

slapi_be_gettype()

Returns the type of the backend.

Syntax

```
#include "slapi-plugin.h"
const char * slapi_be_gettype(Slapi_Backend *be);
```

Parameters

This function takes the following parameter:

Parameter	Description
be	Pointer to the structure containing the backend configuration.

Returns

This function returns the type of the backend.

Memory Concerns

You should not free the returned pointer.

slapi_be_is_flag_set()

Checks if a flag is set in the backend configuration.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_is_flag_set(Slapi_Backend * be, int flag);
```

Parameters

This function takes the following parameters:

Parameter	Description
be	Pointer to the structure containing the backend configuration.
flag	Flag to check (for example, SLAPI_BE_FLAG_REMOTE_DATA).

Returns

This function returns one of the following values:

- 0 if a flag is not set in the backend configuration.
- 1 if a flag is set.

slapi_be_issuffix()

Verifies that the specified suffix matches a registered backend suffix.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_issuffix(const Slapi_Backend *be,
    const Slapi_DN *suffix);
```

Parameters

This function takes the following parameters:

Parameter	Description
`be`	Pointer to the structure containing the backend configuration.
`suffix`	DN of the suffix for which you are looking.

Returns

This function returns one of the following values:

- `0` if the suffix is not part of the specified backend.
- `1` if the suffix is part of the specified backend.

Description

This function checks if the specified suffix exactly matches a registered suffix on a specified backend.

slapi_be_logchanges()

Indicates if the changes applied to the backend should be logged in the changelog.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_logchanges(Slapi_Backend *be);
```

Parameters

This function takes the following parameter:

Parameter	Description
be	Pointer to the structure containing the backend configuration.

Returns

This function returns one of the following values:

- `0` if the changes applied to the specific backend should not be logged in the changelog.
- `1` if the changes should be logged in the changelog.

slapi_be_private()

Verifies if the backend is private.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_private( Slapi_Backend * be );
```

Parameters

This function takes the following parameter:

Parameter	Description
be	Pointer to the structure containing the backend configuration.

Returns

This function returns one of the following values:

- `0` if the backend is not hidden from the user.
- `1` if the backend is hidden from the user (for internal use only).

slapi_be_select()

Finds the backend that should be used to service the entry with the specified DN.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend * slapi_be_select( const Slapi_DN * sdn );
```

Parameters

This function takes the following parameter:

Parameter	Description
sdn	Pointer to the DN of which you wish to get the backend.

Returns

This function returns a pointer to the default backend, if no backend with the appropriate suffix is configured. Otherwise, it returns a pointer to the backend structure.

Memory Concerns

You should not free the returned pointer.

See Also

slapi_be_select_by_instance_name()

slapi_be_select_by_instance_name()

Find the backend used to service the database.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend *slapi_be_select_by_instance_name( const char *name );
```

Parameters

This function takes the following parameter:

Parameter	Description
name	Pointer to the name of the backend of which you wish to get the structure.

Returns

This function returns NULL if no backend with the appropriate name is configured. Otherwise, it returns a pointer to the backend structure.

Description

This function finds the backend that should used to service the database named as the parameter.

Memory Concerns

You should not free the returned pointer.

See Also

`slapi_be_select()`

slapi_berval_cmp()

Compare two `berval` structures.

Syntax

```
#include "slapi-plugin.h"
int slapi_berval_cmp(const struct berval* L,const struct berval* R);
```

Parameters

This function takes the following parameters:

Parameter	Description
`L`	One of the `berval` structures
`R`	The other `berval` structure

Description

This function checks whether two `berval` structures are equivalent.

Returns

This function returns `0` if the two `berval` structures are equivalent. It returns a negative value if `L` is shorter than `R`, and a positive value if `L` is longer than `R`.

slapi_build_control()

Creates an LDAPControl structure based on a BerElement, an OID, and a criticality flag.

Syntax

```
#include "slapi-plugin.h"
int slapi_build_control( char *oid, BerElement *ber,
    char iscritical, LDAPControl **ctrlp );
```

Parameters

This function takes the following parameters:

Parameter	Description
oid	The OID (object identifier) for the control that is to be created.
ber	A BerElement that contains the control value. Pass NULL if the control has no value.
iscritical	The criticality flag. If non-zero, the control will be marked as critical. If 0, it will not be marked as critical.
ctrlp	Pointer that will receive the allocated LDAPControl structure.

Returns

This function returns LDAP_SUCCESS (LDAP result code) if successful.

Description

This function creates an LDAPControl structure based on a BerElement, an OID, and a criticality flag. The LDAPControl that is created can be used in LDAP client requests or internal operations.

Memory Concerns

The contents of the ber parameter are consumed by this function. Because of this, the caller should not free the BerElement once a successful call has been made to slapi_build_control().

The LDAPControl pointer that is returned in ctrlp should be freed by calling ldap_control_free(), which is an LDAP API function.

See Also

slapi_build_control_from_berval()

slapi_build_control()

ldap_control_free(3LDAP)

slapi_build_control_from_berval()

Creates an LDAPControl structure based on a struct berval, an OID, and a criticality flag.

Syntax

```
#include "slapi-plugin.h"
int slapi_build_control_from_berval( char *oid,
    struct berval *bvp, char iscritical, LDAPControl **ctrlp );
```

Parameters

This function takes the following parameters:

Parameter	Description
oid	The OID (object identifier) for the control that is to be created.
bvp	A struct berval that contains the control value. Pass NULL if the control has no value.
iscritical	The criticality flag. If non-zero, the control will be marked as critical. If 0, it will not be marked as critical.
ctrlp	Pointer that will receive the allocated LDAPControl structure.

Returns

This function always returns LDAP_SUCCESS (LDAP result code).

Description

This function creates an LDAPControl structure based on a struct berval, an OID, and a criticality flag. The LDAPControl that is created can be used in LDAP client requests or internal operations.

Memory Concerns

The contents of the bvp parameter are consumed by this function. Because of this, the caller should not free the bvp->bv_val pointer once a successful call to this function has been made.

The LDAPControl pointer that is returned in ctrlp should be freed by calling ldap_control_free(), which is an LDAP API function.

See Also

`slapi_build_control()`

`slapi_dup_control()`

`ldap_control_free`(3LDAP)

slapi_ch_array_free()

Frees an existing array.

Syntax

```
#include "slapi-plugin.h"
void slapi_ch_array_free( char **arrayp );
```

Parameters

This function takes the following parameter:

Parameter	Description
`arrayp`	Pointer to the array to be freed. This parameter can be NULL.

Description

This function frees the `char **` pointed to by `arrayp`.

slapi_ch_bvdup()

Makes a copy of an existing `berval` structure.

Syntax

```
#include "slapi-plugin.h"
extern struct berval* slapi_ch_bvdup( const struct berval *v );
```

Parameters

This function takes the following parameter:

Parameter	Description
`v`	Pointer to the `berval` structure that you want to copy.

Returns

This function returns a pointer to the new copy of the `berval` structure. If the structure cannot be duplicated (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

The contents of the `v` parameter are not altered by this function. The returned `berval` structure should be freed by calling `ber_bvfree()`, which is an LDAP API function.

See Also

`slapi_ch_bvecdup()`

`ber_bvfree`(3LDAP)

slapi_ch_bvecdup()

Makes a copy of an array of existing `berval` structures.

Syntax

```
#include "slapi-plugin.h"
extern struct berval** slapi_ch_bvecdup (const struct berval **v);
```

Parameters

This function takes the following parameters:

Parameter	Description
`v`	Pointer to the array of `berval` structures that you want to copy.

Returns

This function returns a pointer to an array of the new copy of the `berval` structures. If the structures cannot be duplicated (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

The contents of the `v` parameter are not altered by this function. The returned `berval` structure should be freed by calling `ber_bvfree()`, an LDAP API function.

See Also

`slapi_ch_bvecdup()`

`ber_bvfree`(3LDAP)

slapi_ch_calloc()

Allocates space for an array of a number of elements of a specified size.

Syntax

```
#include "slapi-plugin.h"
char * slapi_ch_calloc( unsigned long nelem, unsigned long size );
```

Parameters

This function takes the following parameters:

Parameter	Description
`nelem`	Number of elements for which you wish to allocate memory.
`size`	Size in bytes of the element for which you wish to allocate memory.

Returns

This function returns a pointer to the newly allocated space of memory. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

This function should be called instead of the standard `calloc()` C function, and terminates the `slapd` server with an “out of memory” error message if memory cannot be allocated.

You should free the returned pointer by calling `slapi_ch_free()`.

See Also

`slapi_ch_free()`

`slapi_ch_malloc()`

`slapi_ch_realloc()`

`slapi_ch_strdup()`

slapi_ch_free()

Frees space allocated by the `slapi_ch_malloc()`, `slapi_ch_realloc()`, and `slapi_ch_calloc()` functions and sets the pointer to NULL. Call this function instead of the standard `free()` C function.

Syntax

```
#include "slapi-plugin.h"
void slapi_ch_free( void **ptr );
```

Parameters

This function takes the following parameter:

Parameter	Description
`ptr`	Address of the pointer to the block of memory that you wish to free. If NULL, no action occurs.

Memory Concerns

The `ptr` passed to `slapi_ch_free()` should be the address of a pointer that was allocated using a slapi call such as `slapi_ch_malloc()`, `slapi_ch_calloc()`, `slapi_ch_realloc()`, or `slapi_ch_strdup()`.

See Also

`slapi_ch_malloc()`

`slapi_ch_calloc()`

`slapi_ch_strdup()`

slapi_ch_free_string()

Frees an existing string allocated by the `slapi_ch_malloc()`, `slapi_ch_realloc()`, and `slapi_ch_calloc()`. Call this function instead of the standard `free()` C function.

Syntax

```
#include "slapi-plugin.h"
void slapi_ch_free_string( char **s );
```

Parameters

This function takes the following parameter:

Parameter	Description
s	Address of the string that you wish to free.

Description

This function frees an existing string, and should be used in favor of `slapi_ch_free()` when using strings. It will perform compile-time error checking for incorrect error arguments, as opposed to `slapi_ch_free()`, which defeats the compile-time checking because you must cast the argument to `(void**)`.

See Also

`slapi_ch_free()`

`slapi_ch_malloc()`

`slapi_ch_calloc()`

`slapi_ch_realloc()`

`slapi_ch_strdup()`

slapi_ch_malloc()

Allocates space in memory.

Syntax

```
#include "slapi-plugin.h"
char * slapi_ch_malloc( unsigned long size );
```

Parameters

This function takes the following parameter:

Parameter	Description
size	Size in bytes of the space for which you wish to get the memory.

Returns

This function returns a pointer to the newly allocated space of memory. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

This function should be called instead of the standard `malloc()` C function, and terminates the `slapd` server with an “out of memory” error message if memory cannot be allocated.

The returned pointer should be freed by calling `slapi_ch_free()`.

See Also

`slapi_ch_free()`

`slapi_ch_calloc()`

`slapi_ch_realloc()`

`slapi_ch_strdup()`

slapi_ch_realloc()

Changes the size of a block of allocated memory.

Syntax

```
#include "slapi-plugin.h"
char * slapi_ch_realloc( char *block, unsigned long size );
```

Parameters

This function takes the following parameters:

Parameter	Description
`block`	Pointer to an existing block of allocated memory.
`size`	New size (in bytes) of the block of memory you want allocated.

Returns

This function returns a pointer to the reallocated space of memory. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

This function should be called instead of the standard `realloc()` C function, and terminates the `slapd` server with an “out of memory” error message if memory cannot be allocated.

The block parameter passed to slapi_ch_realloc() should be the address of a pointer that was allocated using a slapi call such as slapi_ch_malloc(), slapi_ch_calloc(), or slapi_ch_strdup().

The returned pointer should be freed by calling slapi_ch_free().

See Also

slapi_ch_free()

slapi_ch_calloc()

slapi_ch_strdup()

slapi_ch_strdup()

Makes a copy of an existing string.

Syntax

```
#include "slapi-plugin.h"
char * slapi_ch_strdup( char *s );
```

Parameters

This function takes the following parameter:

Parameter	Description
s	Pointer to the string you want to copy.

Returns

This function returns a pointer to a copy of the string. If space cannot be allocated (for example, if no more virtual memory exists), the slapd program terminates.

Memory Concerns

This function should be called instead of the standard strdup() C function, and terminates the slapd server with an “out of memory” error message if memory cannot be allocated.

The returned pointer should be freed by calling slapi_ch_free().

See Also

slapi_ch_free()

slapi_ch_calloc()

`slapi_ch_malloc()`

`slapi_ch_realloc()`

slapi_compute_add_evaluator()

Sets a callback for use by the server in evaluating which computed attributes to generate and include in an entry before returning a result to a client.

Syntax

```
#include "slapi-plugin.h"
int slapi_compute_add_evaluator(slapi_compute_callback_t fcn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`fcn`	Function to call when evaluating computed attributes

Returns

This function returns `0` if successful. Otherwise, it returns `ENOMEM` indicating that no memory could be allocated for the callback.

Description

For a description of the callback, refer to the “Data Type and Structure Reference,” on page 11. Register the callback as part of plug-in initialization.

See Also

`computed_attr_context`

`slapi_compute_callback_t`

`slapi_pblock_new()`

slapi_compute_add_search_rewriter_ex()

Sets callbacks for use by the server in searching against computed attributes.

Syntax

```
#include "slapi-plugin.h"
int slapi_compute_add_search_rewriter_ex(
    slapi_search_rewrite_callback_t function,
    slapi_search_rewrite_callback_t cleanup_function);
```

Parameters

This function takes the following parameters:

Parameter	Description
`function`	Function to call to rewrite a filter for the search
`cleanup_function`	Function to call to cleanup after performing the rewritten search

Returns

This function returns `0` if successful. Otherwise, it returns `ENOMEM` indicating that no memory could be allocated for the callback.

Description

For a description of the callback, refer to the “Data Type and Structure Reference,” on page 11. Register the callback as part of plug-in initialization.

See Also

`slapi_search_rewrite_callback_t`

slapi_control_present()

Determines whether or not the specified object identification (OID) identifies a control that is present in a list of controls.

Syntax

```
#include "slapi-plugin.h"
int slapi_control_present( LDAPControl **controls, char *oid,
    struct berval **val, int *iscritical );
```

Parameters

This function takes the following parameters:

Parameter	Description

`controls`	List of controls that you want to check.
`oid`	OID of the control that you want to find.
`val`	If the control is present in the list of controls, this function specifies the pointer to the `berval` structure containing the value of the control. If you do not want to receive a pointer to the control value, pass NULL for this parameter.
`iscritical`	If the control is present in the list of controls, this function specifies whether or not the control is critical to the operation of the server: • 0 means that the control is not critical to the operation. • 1 means that the control is critical to the operation. • If you do not want to receive an indication of whether the control is critical or not, pass NULL for this parameter.

Returns

This function returns one of the following values:

- `1` if the specified control is present in the list of controls.
- `0` if the control is not present in the list of controls.

Memory Concerns

The `val` output parameter is set to point into the controls array. A copy of the control value is not made.

See Also

`slapi_get_supported_controls_copy()`

`slapi_register_supported_control()`

slapi_delete_internal_pb()

Performs an LDAP delete operation based on a parameter block to remove a directory entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_delete_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

Parameter	Description
pb	A parameter block that has been initialized using slapi_delete_internal_set_pb().

Returns

This function returns -1 if the parameter passed is a NULL pointer. Otherwise, it returns 0.

After your code calls this function, the server sets SLAPI_PLUGIN_INTOP_RESULT in the parameter block to the appropriate LDAP result code. You can therefore check SLAPI_PLUGIN_INTOP_RESULT in the parameter block to determine whether an error has occurred.

Description

This function performs an internal delete operation based on a parameter block. The parameter block should be initialized by calling slapi_delete_internal_set_pb().

Memory Concerns

None of the parameters that are passed to slapi_delete_internal_set_pb() are altered or consumed by this function.

See Also

slapi_delete_internal_set_pb()

slapi_delete_internal_set_pb()

Prepares a parameter block for an internal delete operation.

Syntax

```
#include "slapi-plugin.h"
int slapi_delete_internal_set_pb(Slapi_PBlock *pb, const char *dn,
    LDAPControl **controls, const char *uniqueid,
    Slapi_ComponentId *plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

Parameter	Description

pb	Parameter block for the internal add operation
dn	Distinguished Name of the entry to add
controls	Array of controls to request for the add operation
uniqueid	Unique identifier for the entry if using this rather than DN.
plugin_identity	Plug-in identifier obtained from SLAPI_PLUGIN_IDENTITY during plug-in initialization
operation_flags	Flags such as SLAPI_OP_FLAG_NEVER_CHAIN

Returns

This function returns 0 if successful. Otherwise, it returns an LDAP error code.

Description

This function prepares a parameter block for use with slapi_delete_internal_pb() using the components of the entry.

Memory Concerns

Allocate the parameter block using slapi_pblock_new() before calling this function.

Free the parameter block after calling slapi_delete_internal_pb().

See Also

slapi_delete_internal_pb()

slapi_pblock_new()

slapi_destroy_condvar()

Frees a Slapi_CondVar structure from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_destroy_condvar( Slapi_CondVar *cvar );
```

Parameters

This function takes the following parameters:

Parameter	Description

`cvar`	Pointer to the `Slapi_CondVar` structure that you want to free from memory.

Description

This function frees a `Slapi_CondVer` structure from memory. Before calling this function, you should make sure that this condition variable is no longer in use.

slapi_destroy_mutex()

Frees a `Slapi_Mutex` structure from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_destroy_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

Parameter	Description
`mutex`	Pointer to the `Slapi_Mutex` structure that you want to free from memory.

Description

This function frees a `Slapi_Mutex` structure from memory. The calling function must ensure that no thread is currently in a lock-specific function. Locks do not provide self-referential protection against deletion.

slapi_dn_beparent()

Gets a copy of the distinguished name (DN) of the parent of an entry, unless the specified entry's DN is the suffix of the local database.

If you do not want to check if the entry's DN is the suffix of the local database, call the `slapi_dn_parent()` function instead.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_beparent( Slapi_PBlock *pb, char *dn );
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block.
dn	DN of the entry for which you want to find the parent.

Returns

This function returns the DN of the parent entry; or NULL if the specified DN is NULL, if the DN is an empty string, if the DN has no parent (for example, o=example.com), or if the specified DN is the suffix of the local database.

See Also

slapi_dn_parent()

slapi_dn_ignore_case()

Converts all characters in a distinguished name (DN) to lowercase.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_ignore_case( char *dn );
```

Parameters

This function takes the following parameters:

Parameter	Description
dn	DN that you want to convert to lowercase.

Returns

This function returns the DN with lowercase characters. Notice that the variable passed in as the dn argument is also converted in place.

See Also

slapi_dn_normalize()

slapi_dn_isbesuffix()

Determines whether or not the specified distinguished name (DN) is the suffix of the local database. Before calling this function, you should call `slapi_dn_normalize_case()` to normalize the DN and convert all characters to lowercase.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_isbesuffix( Slapi_PBlock *pb, char *dn );
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block.
`dn`	DN that you want to check.

Returns

This function returns 1 if the specified DN is the suffix for the local database, or 0 if the DN is not the suffix.

See Also

`slapi_dn_isroot()`

slapi_dn_isparent()

Determines whether or not a particular DN is the parent of another specified DN. Before calling this function, you should call `slapi_dn_normalize_case()` to normalize the DNs and convert all characters to lowercase.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_isparent( const char *parentdn, char *childdn );
```

Parameters

This function takes the following parameters:

Parameter	Description

parentdn	Determine if this DN is the parent of childdn.
childdn	Determine if this DN is the child of parentdn.

Returns

This function returns a non-zero value if parentdn is the parent of childdn, or 0 if the parentdn is not the parent of childdn.

See Also

slapi_dn_issuffix()

slapi_dn_isroot()

Determines if the specified DN is the root DN for this local database. Before calling this function, you should call slapi_dn_normalize_case() to normalize the DN and convert all characters to lowercase.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_isroot( Slapi_PBlock *pb, char *dn );
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block.
dn	DN that you want to check.

Returns

This function returns 1 if the specified DN is the root DN of the local database, or 0 if the DN is not the root DN.

See Also

slapi_dn_isbesuffix()

slapi_dn_issuffix()

Determines if a DN is equal to the specified suffix. Before calling this function, you should call `slapi_dn_normalize_case()` to normalize the DN and convert all characters to lowercase.

If you want to determine if a DN is the same as the suffix for the local database, call the `slapi_dn_isbesuffix()` function instead.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_issuffix( const char *dn, const char *suffix );
```

Parameters

This function takes the following parameters:

Parameter	Description
`dn`	DN that you want to check.
`suffix`	Suffix that you want compared against the DN.

Returns

This function returns 1 if the specified DN is the same as the specified suffix, or 0 if the DN is not the same as the suffix.

See Also

`slapi_dn_isparent()`

slapi_dn_normalize()

Converts a distinguished name (DN) to canonical format (no leading or trailing spaces, no spaces between components, and no spaces around the equals sign). For example, given the following DN:

```
cn = Moxie Cross , ou = Engineering , dc = example , dc = com
```

the function returns:

```
cn=Moxie Cross,ou=Engineering,dc=example,dc=com
```

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_normalize( char *dn );
```

Parameters

This function takes the following parameters:

Parameter	Description
dn	DN that you want to normalize.

Returns

This function returns the normalized DN. Notice that the variable passed in as the dn argument is also converted in place.

See Also

`slapi_dn_normalize_to_end()`

`slapi_dn_normalize_case()`

slapi_dn_normalize_case()

Converts a distinguished name (DN) to canonical format and converts all characters to lowercase. Calling this function has the same effect as calling the `slapi_dn_normalize()` function followed by the `slapi_dn_ignore_case()` function.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_normalize_case( char *dn );
```

Parameters

This function takes the following parameters:

Parameter	Description
dn	DN that you want to normalize and convert to lowercase.

Returns

This function returns the normalized DN with all lowercase characters. Notice that variable passed in as the dn argument is also converted in-place.

See Also

`slapi_dn_normalize()`

`slapi_dn_ignore_case()`

slapi_dn_normalize_to_end()

Normalizes part of a DN value, specifically, the part going from what is pointed to by dn to that pointed to by end.

Notice that this routine does not NULL terminate the normalized bit pointed to by dn at the return of the function.

If the argument end happens to be NULL, this routine does basically the same thing as `slapi_dn_normalize()`, except for NULL terminating the normalized DN.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_normalize_to_end( char *dn, char *end );
```

Parameters

This function takes the following parameters:

Parameter	Description
dn	DN value to be normalized.
end	Pointer to the end of what will be normalized from the DN value in dn. If this argument is NULL, the DN value in dn will be wholly normalized.

Returns

This function returns a pointer to the end of the dn that has been normalized. For example, if the RDN is `cn=Moxie` and the DN is `c=France,o=example.com`, the new DN will be `cn=Moxie,c=France,o=example.com`.

See Also

`slapi_dn_normalize()`

slapi_dn_parent()

Gets a copy of the distinguished name (DN) of the parent of an entry. Before calling this function, you should call `slapi_dn_normalize_case()` to normalize the DN and convert all characters to lowercase.

If you want to check if the DN is the suffix of the local database, call the `slapi_dn_beparent()` function instead.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_parent( char *dn );
```

Parameters

This function takes the following parameter:

Parameter	Description
`dn`	DN of the entry for which you want to find the parent.

Returns

This function returns the DN of the parent entry. If the specified DN is `NULL`, if the DN is an empty string, or if the DN has no parent (for example, `o=example.com`), the function returns `NULL`.

slapi_dn_plus_rdn()

Adds an RDN to DN.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_plus_rdn( const char *dn, const char *rdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`dn`	DN value to which a new RDN is to be added.
`rdn`	RDN value that is to be added to the DN value in `dn`.

Returns

This function returns the new DN formed by adding the RDN value in `rdn` to the DN value in `dn`.

See Also

`slapi_sdn_add_rdn()`

slapi_dup_control()

Makes an allocated copy of an `LDAPControl`.

Syntax

```
#include "slapi-plugin.h"
LDAPControl * slapi_dup_control( LDAPControl *ctrl );
```

Parameters

This function takes the following parameter:

Parameter	Description
`ctrl`	Pointer to an `LDAPControl` structure whose contents are to be duplicated.

Returns

This function returns a pointer to an allocated `LDAPControl` structure if successful, or `NULL` if an error occurs.

Description

This function duplicates the contents of an `LDAPControl` structure. All fields within the `LDAPControl` are copied to a new, allocated structure, and a pointer to the new structure is returned.

Memory Concerns

The structure that is returned should be freed by calling `ldap_control_free()`, an LDAP API function.

See Also

`ldap_control_free`(3LDAP)

slapi_entry2mods()

Creates an array of LDAPMod from a Slapi_Entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry2mods(const Slapi_Entry *e,
    char **dn, LDAPMod ***attrs);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Pointer to a Slapi_Entry.
dn	Address of a char* that will be set on return to the entry DN.
attrs	Address of an array of LDAPMod that will be set on return to a copy of the entry attributes.

Returns

This function returns one of the following values:

- 0 if successful.
- non-0 if not successful.

Description

This function creates an array of LDAPMod of type LDAP_MOD_ADD from a Slapi_Entry.

See Also

slapi_mods2entry()

slapi_entry2str()

Generates an LDIF string description of an LDAP entry.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry2str( Slapi_Entry *e, int *len );
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry that you want to convert into an LDIF string.
`len`	Length of the returned LDIF string.

Returns

Returns the LDIF string representation of the entry you specify. If an error occurs, the function returns `NULL`.

Description

This function generates an LDIF string value conforming to the following format:

```
dn: dn\n
[attr: value\n]*
```

For example:

```
dn: uid=jdoe, ou=People, o=example.com
cn: Jane Doe
sn: Doe
...
```

To convert a string description in LDIF format to an entry of the `Slapi_Entry` data type, call the `slapi_str2entry()` function.

Memory Concerns

When you no longer need to use the string, you should free it from memory by calling the `slapi_ch_free_string()` function.

See Also

`slapi_entry2str_with_options()`

`slapi_str2entry()`

slapi_entry2str_with_options()

Generates a description of an entry as an LDIF string. This function behaves much like `slapi_str2entry()`; however, you can specify output options with this function.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry2str_with_options( Slapi_Entry *e,
  int *len, int options );
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry that you want to convert into an LDIF string.
len	Length of the LDIF string returned by this function.
options	An option set that specifies how you want the string converted.

The Options Parameter

You can OR together any of the following options when you call this function:

Flag Value	Description
SLAPI_DUMP_STATEINFO	This is only used internally by replication. This allows access to the internal data used by multimaster replication.
SLAPI_DUMP_UNIQUEID	This option is used when creating an LDIF file to be used to initialize a replica. Each entry will contain the nsuniqueID operational attribute.
SLAPI_DUMP_NOOPATTRS	By default, certain operational attributes (such as creatorName, modifiersName, createTimestamp, modifyTimestamp) may be included in the output. With this option, no operational attributes will be included.
SLAPI_DUMP_NOWRAP	By default, lines will be wrapped as defined in the LDIF specification. With this option, line wrapping is disabled.

Returns

This function returns the LDIF string representation of the entry you specify or NULL if an error occurs.

Description

This function generates an LDIF string value conforming to the following syntax:

```
dn: dn\n
[attr: value\n]*
```

For example:

```
dn: uid=jdoe, ou=People, o=example.com
cn: Jane Doe
sn: Doe
...
```

To convert an entry described in LDIF string format to an LDAP entry using the `Slapi_Entry` data type, call the `slapi_str2entry()` function.

Memory Concerns

When you no longer need to use the string, you should free it from memory by calling the `slapi_ch_free_string()` function.

See Also

`slapi_entry2str()`

`slapi_str2entry()`

slapi_entry_add_rdn_values()

Adds the components in an entry's relative distinguished name (RDN) to the entry as attribute values. (For example, if the entry's RDN is `uid=bjensen`, the function adds `uid=bjensen` to the entry as an attribute value.)

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_rdn_values( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry to which you want to add the RDN attributes.

Returns

This function returns one of the following values:

- `LDAP_SUCCESS` if the values were successfully added to the entry. The function also returns `LDAP_SUCCESS` if the entry is `NULL`, if the entry's DN is `NULL`, or if the entry's RDN is `NULL`.
- `LDAP_INVALID_DN_SYNTAX` if the DN of the entry cannot be parsed.

Memory Concerns

Free the entry from memory by using `slapi_entry_free()` if the entry was allocated by the user.

See Also

`slapi_entry_free()`

slapi_entry_add_string()

Adds a string value to an attribute in an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_string (Slapi_Entry *e, const char *type,
    const char *value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry to which you want to add a string value.
`type`	Attribute to which you want to add a string value.
`value`	String value you want to add.

Returns

This function returns `0` when successful; any other value returned signals failure.

Description

This function adds a string value to the existing attribute values in an entry. If the specified attribute does not exist in the entry, the attribute is created with the string value specified.

Memory Concerns

This routine makes a copy of the parameter `value`. `value` can be `NULL`.

slapi_entry_add_value()

Adds a specified `Slapi_Value` data value to an attribute in an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_value (Slapi_Entry *e, const char *type,
    const Slapi_Value *value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry to which you want to add a value.
`type`	Attribute to which you want to add a value.
`value`	The `Slapi_Value` data value you want to add to the entry.

Returns

Returns `0` when successful; any other value returned signals failure.

Description

This function adds a `Slapi_Value` data value to the existing attribute values in an entry. If the specified attribute does not exist in the entry, the attribute is created with the `Slapi_Value` specified.

Memory Concerns

This routine makes a copy of the parameter `value`. `value` can be `NULL`.

slapi_entry_add_values_sv()

Adds an array of `Slapi_Value` data values to the specified attribute in an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_values_sv( Slapi_Entry *e, const char *type,
    Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry to which you want to add values.
type	Attribute type to which you want to add values.
vals	Array of Slapi_Value data values that you want to add.

Returns

Returns one of the following values:

- LDAP_SUCCESS if the Slapi_Value array is successfully added to the attribute.
- LDAP_TYPE_OR_VALUE_EXISTS if any values you are trying to add duplicate an existing value in the attribute.
- LDAP_OPERATIONS_ERROR if there are pre-existing duplicate values in the attribute.

Description

This function adds an array of Slapi_Value data values to an attribute. If the attribute does not exist, it is created and given the value contained in the Slapi_Value array.

This function replaces the deprecated slapi_entry_add_values() function. This function uses Slapi_Value attribute values instead of the berval attribute values.

Memory Concerns

This routine makes a copy of the parameter vals. vals can be NULL.

slapi_entry_add_valueset()

Add a Slapi_ValueSet data value to the specified attribute in an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_valueset(Slapi_Entry *e, const char *type,
    Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry to which you want to add values.
`type`	Attribute type to which you want to add values.
`vs`	`Slapi_ValueSet` data value that you want to add to the entry.

Returns

Returns `0` when successful; any other value returned signals failure.

Description

This function adds a set of values to an attribute in an entry. The values added are in the form of a `Slapi_ValueSet` data type. If the entry does not contain the attribute specified, it is created with the specified `Slapi_ValueSet` value.

Memory Concerns

This routine makes a copy of the parameter `vs`. `vs` can be `NULL`.

slapi_entry_alloc()

Allocates memory for a new entry of the data type `Slapi_Entry`.

Syntax

```
#include "slapi-plugin.h"
Slapi_Entry *slapi_entry_alloc();
```

Returns

Returns a pointer to the newly allocated entry of the data type `Slapi_Entry`. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Description

This function returns an empty `Slapi_Entry` structure. You can call other front-end functions to set the DN and attributes of this entry.

Memory Concerns

When you are no longer using the entry, you should free it from memory by calling the `slapi_entry_free()` function.

See Also

`slapi_entry_dup()`

`slapi_entry_free()`

slapi_entry_attr_delete()

Deletes an attribute (and all its associated values) from an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_delete( Slapi_Entry *e, const char *type );
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry from which you want to delete the attribute.
`type`	Attribute type that you want to delete.

Returns

This function returns one of the following values:

- 0 if successful.
- 1 if the specified attribute is not part of the entry.
- -1 if an error occurred.

slapi_entry_attr_find()

Determines if an entry contains the specified attribute. If the entry contains the attribute, the function returns a pointer to the attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_find( const Slapi_Entry *e, const char *type,
    Slapi_Attr **attr );
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry that you want to check.
`type`	Name of the attribute that you want to check.
`attr`	Pointer to the attribute, if the attribute is in the entry.

Returns

This function returns `0` if the entry contains the specified attribute; otherwise it returns `-1`.

Memory Concerns

Do not free the returned `attr`. It is a pointer to the internal entry data structure. It is usually wise to make a copy of the returned `attr`, using `slapi_attr_dup()`, to avoid dangling pointers if the entry is freed while the pointer to `attr` is still being used.

See Also

`slapi_entry_dup()`

slapi_entry_attr_get_charptr()

Gets the first value of an attribute in an entry as a string.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry_attr_get_charptr(const Slapi_Entry* e,
    const char *type);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry from which you want to get the string value.
type	Attribute type from which you want to get the value.

Returns

This function returns a copy of the first value in the attribute, or NULL if the entry does not contain the attribute.

Memory Concerns

When you are done working with this value, you should free it from memory by calling the slapi_ch_free() function.

slapi_entry_attr_get_int()

Gets the first value of an attribute in an entry as an integer.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_get_int(const Slapi_Entry* e,const char *type);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry from which you want to get the integer value.
type	Attribute type from which you want to get the value.

Returns

Returns the first value in the attribute converted to an integer or 0 if the entry does not contain the attribute.

slapi_entry_attr_get_long()

Gets the first value of an attribute in an entry as a long data type.

Syntax

```
#include "slapi-plugin.h"
long slapi_entry_attr_get_long( const Slapi_Entry* e,
    const char *type);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry from which you want to get the long value.
type	Attribute type from which you want to get the value.

Returns

This function returns the first value in the attribute converted to a long type. The function returns 0 if the entry does not contain the attribute specified.

slapi_entry_attr_get_uint()

Gets the first value of an attribute in an entry as a unsigned integer data type.

Syntax

```
#include "slapi-plugin.h"
unsigned int slapi_entry_attr_get_uint( const Slapi_Entry* e,
    const char *type);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry from which you want to get the value.
type	Attribute type from which you want to get the value.

Returns

This function returns the first value in the attribute converted to an `unsigned integer`. The function returns `0` if the entry does not contain the attribute specified.

slapi_entry_attr_get_ulong()

Gets the first value of an attribute in an entry as a unsigned long data type.

Syntax

```
#include "slapi-plugin.h"
unsigned long slapi_entry_attr_get_ulong( const Slapi_Entry* e,
    const char *type);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry from which you want to get the value.
`type`	Attribute type from which you want to get the value.

Returns

This function returns the first value in the attribute converted to an `unsigned long`. The function returns `0` if the entry does not contain the attribute specified.

slapi_entry_attr_hasvalue()

Determines if an attribute in an entry contains a specified value.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_hasvalue(Slapi_Entry *e, const char *type,
    const char *value);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry that you want to check.
type	Attribute type that you want to test for the value specified.
value	Value that you want to find in the attribute.

Returns

Returns one of the following values:

- 1 if the attribute contains the specified value.
- 0 if the attribute does not contain the specified value.

Memory Concerns

value must not be NULL.

slapi_entry_attr_merge_sv()

Adds an array of Slapi_Value data values to the existing attribute values in an entry. If the attribute does not exist, it is created with the Slapi_Value specified.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_merge_sv( Slapi_Entry *e, const char *type,
    Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry to which you want to add values.
type	Attribute to which you want to add values.
vals	Array of Slapi_Value data values you want to add.

Returns

Returns 0 if successful; any other value returned signals failure.

Description

This function replaces the deprecated `slapi_entry_attr_merge()` function. This function uses `Slapi_Value` attribute values instead of the `berval` attribute values.

Memory Concerns

This function makes a copy of the parameter `vals`. `vals` can be `NULL`.

slapi_entry_attr_replace_sv()

Replaces the values of an attribute with the `Slapi_Value` data value you specify.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_replace_sv( Slapi_Entry *e, const char *type,
    Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry in which you want to replace values.
`type`	Attribute type which will receive the replaced values.
`vals`	Array containing the `Slapi_Value` values that should replace the existing values of the attribute.

Returns

This function returns `0` when successful; any other value returned signals failure.

Description

This function replaces existing attribute values in a specified entry with a single `Slapi_Value` data value. The function first deletes the existing attribute from the entry, then replaces it with the new value specified.

This function replaces the deprecated `slapi_entry_attr_replace()` function. This function uses `Slapi_Value` attribute values instead of the `berval` attribute values.

Memory Concerns

This function makes a copy of the parameter `vals`. `vals` can be `NULL`.

slapi_entry_attr_set_charptr()

Replaces the value or values of an attribute in an entry with a specified string value.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_charptr(Slapi_Entry* e, const char *type,
    const char *value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry in which you want to set the value.
`type`	Attribute type in which you want to set the value.
`value`	String value that you want to assign to the attribute.

Memory Concerns

This function makes a copy of the parameter `values`. `vaulues` can be `NULL`, and if so, this function is roughly equivalent to `slapi_entry_attr_delete`().

See Also

slapi_entry_attr_delete()

slapi_entry_attr_set_int()

Replaces the value or values of an attribute in an entry with a specified integer data value.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_int(Slapi_Entry* e, const char *type,
    int l);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry in which you want to set the value.
type	Attribute type in which you want to set the value.
l	Integer value that you want assigned to the attribute.

Description

This function will replace the value or values of an attribute with the integer value that you specify. If the attribute does not exist, it is created with the integer value that you specify.

slapi_entry_attr_set_long()

Replaces the value or values of an attribute in an entry with a specified long data type value.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_long(Slapi_Entry* e, const char *type,
    unsigned long l);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry in which you want to set the value.
type	Attribute type in which you want to set the value.
l	Long integer value that you want assigned to the attribute.

slapi_entry_attr_set_uint()

Replaces the value or values of an attribute in an entry with a specified unsigned integer data type value.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_uint(Slapi_Entry* e, const char *type,
    unsigned int l);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry in which you want to set the value.
`type`	Attribute type in which you want to set the value.
`l`	Unsigned integer value that you want assigned to the attribute.

Description

This function will replace the value or values of an attribute with the `unsigned integer` value that you specify. If the attribute does not exist, it is created with the unsigned integer value you specify.

slapi_entry_attr_set_ulong()

Replaces the value or values of an attribute in an entry with a specified unsigned long data type value.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_ulong(Slapi_Entry* e, const char *type,
    unsigned long l);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry in which you want to set the value.
`type`	Attribute type in which you want to set the value.
`l`	Unsigned long value that you want assigned to the attribute.

Description

This function will replace the value or values of an attribute with the `unsigned long` value that you specify. If the attribute does not exist, it is created with the unsigned long value that you specify.

slapi_entry_delete_string()

Deletes a string value from an attribute in an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_delete_string(Slapi_Entry *e, const char *type,
    const char *value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry from which you want the string deleted.
`type`	Attribute type from which you want the string deleted.
`value`	Value of string to delete.

Returns

Returns `0` when successful; any other value returned signals failure.

slapi_entry_delete_values_sv()

Removes an array of `Slapi_Value` data values from an attribute in an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_delete_values_sv( Slapi_Entry *e, const char *type,
    Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry from which you want to delete values.
type	Attribute from which you want to delete values.
vals	Array of Slapi_Value data values that you want to delete.

Returns

Returns LDAP_SUCCESS if the specified attribute and the array of Slapi_Value data values are deleted from the entry.

If the specified attribute contains a NULL value, the attribute is deleted from the attribute list and the function returns LDAP_NO_SUCH_ATTRIBUTE. Additionally, if the attribute is not found in the list of attributes for the specified entry, the function returns LDAP_NO_SUCH_ATTRIBUTE.

If there is an operational error during the processing of this call (such as a duplicate value found), the function will return LDAP_OPERATIONS_ERROR. If this occurs, please report the problem to Sun support services.

Description

This function removes an attribute / value set from an entry. Notice that both the attribute and its Slapi_Value data values are removed from the entry. If you supply a Slapi_Value whose value is NULL, the function will delete the specified attribute from the entry. In either case, the function returns LDAP_SUCCESS.

This function replaces the deprecated slapi_entry_delete_values() function. This function uses Slapi_Value attribute values instead of the berval attribute values.

Memory Concerns

The vals parameter can be NULL, in which case, this function does nothing.

See Also

slapi_entry_delete_values_sv()

slapi_entry_dup()

Makes a copy of an entry, its DN, and its attributes.

Syntax

```
#include "slapi-plugin.h"
Slapi_Entry *slapi_entry_dup( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry that you want to copy.

Returns

This function returns the new copy of the entry. If the structure cannot be duplicated (for example, if no more virtual memory exists), the `slapd` program terminates.

Description

This function returns a copy of an existing `Slapi_Entry` structure. You can call other front-end functions to change the DN and attributes of this entry.

Memory Concerns

When you are no longer using the entry, you should free it from memory by calling the `slapi_entry_free()` function.

See Also

`slapi_entry_alloc()`

`slapi_entry_free()`

slapi_entry_first_attr()

Finds the first attribute in an entry. If you want to iterate through the attributes in an entry, use this function in conjunction with the `slapi_entry_next_attr()` function.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_first_attr( Slapi_Entry *e, Slapi_Attr **attr );
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry from which you want to get the attribute.
attr	Pointer to the first attribute in the entry.

Returns

Returns 0 when successful; any other value returned signals failure.

Memory Concerns

Do not free the returned attr. This is a pointer into the internal entry data structure. If you need a copy, use slapi_attr_dup().

See Also

slapi_attr_dup()

slapi_entry_free()

Frees an entry, its DN, and its attributes from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_free( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry that you want to free. If NULL, no action occurs.

Description

Call this function to free an entry that you have allocated by using the slapi_entry_alloc() function or the slapi_entry_dup() function.

Memory Concerns

To free entries, always use this function, as opposed to using slapi_ch_free(), or free().

See Also

`slapi_entry_alloc()`

`slapi_entry_dup()`

slapi_entry_get_dn()

Gets the distinguished name (DN) of the specified entry.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry_get_dn( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry from which you want to get the DN.

Returns

This function returns the DN of the entry. Notice that this returns a pointer to the actual DN in the entry, not a copy of the DN. You should not free the DN unless you plan to replace it by calling `slapi_entry_set_dn()`.

Memory Concerns

Use `slapi_ch_free()` if you are replacing the DN with `slapi_entry_set_dn()`.

See Also

`slapi_ch_free()`

`slapi_entry_set_dn()`

slapi_entry_get_dn_const()

Returns as a `const` the `DN` value of the entry that you specify.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_entry_get_dn_const( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry from which you want to get the DN as a constant.

Returns

This function returns the DN of the entry that you specify. The DN is returned as a const; you are not able to modify the DN value. If the DN of the Slapi_DN object is NULL, the NDN value of Slapi_DN is returned.

Memory Concerns

Never free this value.

slapi_entry_get_ndn()

Returns the normalized DN from the entry that you specify.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry_get_ndn( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry from which you want to obtain the normalized DN.

Returns

This function returns the normalized DN from the entry that you specify. If the entry you specify does not contain a normalized DN, one is created through the processing of this function.

Memory Concerns

Never free this value.

slapi_entry_get_sdn()

Returns the Slapi_DN object from the entry that you specify.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_entry_get_sdn( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry from which you want to get the Slapi_DN object.

Returns

This function returns the Slapi_DN object from the entry that you specify.

Memory Concerns

Never free this value. If you need a copy, use slapi_sdn_dup().

See Also

slapi_sdn_dup()

slapi_entry_get_sdn_const()

Returns as a const the value of the Slapi_DN object from the entry that you specify.

Syntax

```
#include "slapi-plugin.h"
const Slapi_DN *slapi_entry_get_sdn_const ( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry from which you want to get the Slapi_DN object.

Returns

Returns as a `const` the value of the `Slapi_DN` object from the entry that you specify.

Memory Concerns

Never free this value. If you need a copy, use `slapi_sdn_dup()`.

See Also

`slapi_sdn_dup()`

slapi_entry_get_uniqueid()

Gets the unique ID value of the entry.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_entry_get_uniqueid( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry from which you want obtain the unique ID.

Returns

This function returns the unique ID value of the entry specified.

Memory Concerns

Never free this value. If you need a copy, use `slapi_ch_strdup()`.

See Also

`slapi_ch_strdup()`

slapi_entry_has_children()

This function determines if the specified entry has child entries.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_has_children( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry that you want to test for child entries.

Returns

This function returns `1` if the entry you supply has children entries; otherwise it returns `0`.

slapi_entry_init()

Initializes the values of an entry with the DN and attribute value pairs you supply.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_init(Slapi_Entry *e, char *dn, Slapi_Attr *a);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	The entry you want to initialize.
dn	The DN of the entry you are initializing.
a	Initialization list of attribute value pairs, supplied as a `Slapi_Attr` data value.

Description

This function initializes the attributes and the corresponding attribute values of an entry. Also, during the course of processing, the unique ID of the entry is set to `NULL` and the flag value is set to `0`.

Use this function to initialize a Slapi_Entry pointer.

Memory Concerns

This function should always be used after slapi_entry_alloc(), and never otherwise. For example:

```
Slapi_Entry *e = slapi_entry_alloc();
slapi_entry_init(e, NULL, NULL);
```

To set the DN in the entry:

```
slapi_sdn_set_dn_passin(slapi_entry_get_sdn(e), dn);
```

In this case, the dn argument is not copied, but is consumed by the function. To copy the argument, see the following example:

```
char *dn = slapi_ch_strdup(some_dn);
Slapi_Entry *e = slapi_entry_alloc();
slapi_entry_init(e, dn, NULL);
```

dn is not freed in this context, but is eventually be freed when slapi_entry_free() is called.

See Also

slapi_entry_free()

slapi_entry_alloc()

slapi_ch_strdup()

slapi_entry_merge_values_sv()

Merges (adds) an array of Slapi_Value data values to a specified attribute in an entry. If the entry does not contain the attribute specified, the attribute is created with the value supplied.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_merge_values_sv( Slapi_Entry *e, const char *type,
    Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

Parameter	Description

e	Entry into which you want to merge values.
type	Attribute type that contains the values you want to merge.
vals	Values that you want to merge into the entry. Values are of type Slapi_Value.

Returns

This function returns either LDAP_SUCCESS or LDAP_NO_SUCH_ATTRIBUTE.

Description

This function adds additional Slapi_Value data values to the existing values contained in an attribute. If the attribute type does not exist, it is created.

If the specified attribute exists in the entry, the function merges the value specified and returns LDAP_SUCCESS. If the attribute is not found in the entry, the function creates it with the Slapi_Value specified and returns LDAP_NO_SUCH_ATTRIBUTE.

Notice that if this function fails, it leaves the values for type within a pointer to e in an indeterminate state. The present value set may be truncated.

Memory Concerns

This function makes a copy of vals. vals can be NULL.

slapi_entry_next_attr()

Finds the next attribute after prevattr in an entry. To iterate through the attributes in an entry, use this function in conjunction with the slapi_entry_first_attr() function.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_next_attr( Slapi_Entry *e, Slapi_Attr *prevattr,
    Slapi_Attr **attr );
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry from which you want to get the attribute.
prevattr	Previous attribute in the entry.

attr	Pointer to the next attribute after prevattr in the entry.

Returns

This function returns 0 if successful or -1 if prevattr was the last attribute in the entry.

Memory Concerns

Never free the returned attr. Use slapi_attr_dup() to make a copy if a copy is needed.

See Also

slapi_attr_dup()

slapi_entry_rdn_values_present()

Determines if the values in an entry's relative distinguished name (RDN) are also present as attribute values. (For example, if the entry's RDN is cn=Barbara Jensen, the function determines if the entry has the cn attribute with the value Barbara Jensen.)

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_rdn_values_present( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry from which you want to get the attribute.

Returns

This function returns 1 if the values in the RDN are present in attributes of the entry or 0 if the values are not present.

slapi_entry_schema_check()

Determines whether or not the specified entry complies with the schema for its object class.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_schema_check( Slapi_PBlock *pb, Slapi_Entry *e );
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block.
e	Entry of which you want to check the schema.

Returns

Returns one of the following values:

- `0` if the entry complies with the schema or if schema checking is turned off. The function also returns `0` if the entry has additional attributes not allowed by the schema and has the object class `extensibleObject`.
- `1` if the entry is missing the `objectclass` attribute, if it is missing any required attributes, if it has any attributes not allowed by the schema (but does not have the object class `extensibleObject`), or if the entry has multiple values for a single-valued attribute.

Memory Concerns

The `pb` argument can be `NULL`. It is used only to get the `SLAPI_IS_REPLACED_OPERATION` flag. If that flag is present, no schema checking is done.

slapi_entry_set_dn()

Sets the distinguished name (DN) of an entry.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_set_dn( Slapi_Entry *e, char *dn );
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry to which you want to assign the DN.
dn	Distinguished name you want assigned to the entry.

Description

This function sets a pointer to the DN supplied in the specified entry.

Memory Concerns

dn is freed when slapi_entry_free() is called.

A copy of dn should be passed, for example:

```
char *dn = slapi_ch_strdup(some_dn);
slapi_entry_set_dn(e, dn);
```

The old dn is freed as a result of this call. Do not pass in a NULL value.

See Also

slapi_entry_free()

slapi_entry_set_dn()

slapi_entry_get_dn()

slapi_entry_set_sdn()

Sets the Slapi_DN value in an entry.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_set_sdn( Slapi_Entry *e, const Slapi_DN *sdn );
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Entry to which you want to set the value of the Slapi_DN.

sdn	The specified Slapi_DN value that you want to set.

Description

This function sets the value for the Slapi_DN object in the entry you specify.

Memory Concerns

This function makes a copy of the sdn argument.

slapi_entry_size()

This function returns the approximate size of an entry, rounded to the nearest 1k. This can be useful for checking cache sizes, estimating storage needs, and so on.

Syntax

```
#include "slapi-plugin.h"
size_t slapi_entry_size(Slapi_Entry *e);
```

Parameters

This function takes the following parameter:

Parameter	Description
e	Entry from which you want the size returned.

Returns

This function returns the size of the entry, rounded to the nearest 1k. The value returned is a size_t data type, with is a u_long value. If the entry is empty, a size of 1k is returned.

Description

When determining the size of an entry, only the sizes of the attribute values are counted; the size of other entry values (such as the size of attribute names, variously-normalized DNs, or any metadata) are not included in the size returned. It is assumed that the size of the metadata is well enough accounted for by the rounding of the size to the next largest 1k (this holds true especially in larger entries, where the actual size of the attribute values far outweighs the size of the metadata).

Notice that when determining the size of the entry, both deleted values and deleted attributes are included in the count.

slapi_filter_compare()

Determines if two filters are identical.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_compare(struct slapi_filter *f1,
    struct slapi_filter *f2);
```

Parameters

This function takes the following parameters:

Parameter	Description
`f1`	First filter to compare.
`f2`	Second filter to compare.

Returns

This function returns `0` if the two filters are identical, or a value other than `0` if they are not.

Description

This function allows you to determine if two filters are identical, and/or are allowed to be in a different order.

slapi_filter_free()

Frees the specified filter and (optionally) the set of filters that comprise it (for example, the set of filters in an `LDAP_FILTER_AND` type filter).

Syntax

```
#include "slapi-plugin.h"
void slapi_filter_free( Slapi_Filter *f, int recurse );
```

Parameters

This function takes the following parameters:

Parameter	Description
f	Filter that you want to free.
recurse	If 1, recursively frees all filters that comprise this filter. If 0, only frees the filter specified by f.

Description

This function frees the filter in parameter f.

Memory Concerns

Filters created using slapi_str2filter() must be freed after using this function. Filters extracted from a parameter block using:

```
slapi_pblock_get( pb, SLAPI_SEARCH_FILTER, &filter );
```

must not be freed.

See Also

slapi_str2filter()

slapi_pblock_get()

slapi_filter_get_attribute_type()

Gets the attribute type for all simple filter choices.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_attribute_type( Slapi_Filter *f, char **type );
```

Parameters

This function takes the following parameters:

Parameter	Description
f	Filter from which you wish to get the substring values.
type	Pointer to the attribute type of the filter.

Returns

This function returns the attribute type of the filter.

Description

This function gets the attribute type for all simple filter choices:

- LDAP_FILTER_GE
- LDAP_FILTER_LE
- LDAP_FILTER_APPROX
- LDAP_FILTER_EQUALITY
- LDAP_FILTER_SUBSTRINGS
- LDAP_FILTER_PRESENT
- LDAP_FILTER_EXTENDED
- LDAP_FILTER_AND
- LDAP_FILTER_OR
- LDAP_FILTER_NOT

A filter such as (mail-foo), will return the type mail.

Memory Concerns

The attribute type is returned in type and should not be freed after calling this function. It will be freed at the same time as the Slapi_Filter structure when slapi_filter_free() is called.

See Also

slapi_filter_get_choice()

slapi_filter_get_ava()

slapi_filter_get_type()

slapi_filter_free()

slapi_filter_get_ava()

(Applies only to filters of the types LDAP_FILTER_EQUALITY, LDAP_FILTER_GE, LDAP_FILTER_LE, LDAP_FILTER_APPROX) Gets the attribute type and the value from the filter.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_ava( Slapi_Filter *f, char **type,
    struct berval **bval );
```

Parameters

This function takes the following parameters:

Parameter	Description
f	Filter from which you wish to get the attribute and value.
type	Pointer to the attribute type of the filter.
bval	Pointer to the address of the berval structure containing the value of the filter.

Returns

This function returns 0 if successful, or -1 if the filter is not one of the types listed above.

Description

Filters of the type LDAP_FILTER_EQUALITY, LDAP_FILTER_GE, LDAP_FILTER_LE, and LDAP_FILTER_APPROX generally compare a value against an attribute. For example:

```
(cn=Barbara Jensen)
```

This filter finds entries in which the value of the cn attribute is equal to Barbara Jensen.

The attribute type is returned in the parameter type, and the value is returned in the parameter bval.

Memory Concerns

The strings within the parameters type and bval are direct pointers to memory inside the Slapi_Filter, and therefore should not be freed after usage. They will be freed when a server entity calls slapi_filter_free() after usage of the Slapi_Filter structure.

See Also

slapi_filter_get_choice()

slapi_filter_get_type()

`slapi_filter_get_attribute_type()`

slapi_filter_get_choice()

Gets the type of the specified filter (for example, `LDAP_FILTER_EQUALITY`).

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_choice( Slapi_Filter *f );
```

Parameters

This function takes the following parameter:

Parameter	Description
`f`	Filter type that you wish to get.

Returns

This function returns one of the following values:

- `LDAP_FILTER_AND` (AND filter)

  For example: `(&(ou=Accounting)(l=Sunnyvale))`

- `LDAP_FILTER_OR` (OR filter)

  For example: `(|(ou=Accounting)(l=Sunnyvale))`

- `LDAP_FILTER_NOT` (NOT filter)

  For example: `(!(l=Sunnyvale))`

- `LDAP_FILTER_EQUALITY` (equals filter)

  For example: `(ou=Accounting)`

- `LDAP_FILTER_SUBSTRINGS` (substring filter)

  For example: `(ou=Account*Department)`

- `LDAP_FILTER_GE` ("greater than or equal to" filter)

  For example: `(supportedLDAPVersion>=3)`

- `LDAP_FILTER_LE` ("less than or equal to" filter)

For example: `(supportedLDAPVersion<=2)`

- `LDAP_FILTER_PRESENT` (presence filter)

  For example: `(mail=*)`

- `LDAP_FILTER_APPROX` (approximation filter)

  For example: `(ou~=Sales)`

- `LDAP_FILTER_EXTENDED` (extensible filter)

  For example: `(o:dn:=Example)`

See Also

`slapi_filter_get_type()`

`slapi_filter_get_attribute_type()`

`slapi_filter_get_ava()`

slapi_filter_get_subfilt()

(Applies only to filters of the type `LDAP_FILTER_SUBSTRINGS`) Gets the substring values from the filter.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_subfilt( Slapi_Filter *f, char **type,
    char **initial, char ***any, char **final );
```

Parameters

This function takes the following parameters:

Parameter	Description
`f`	Filter from which you wish to get the substring values.
`type`	Pointer to the attribute type of the filter.
`initial`	Pointer to the initial substring (“starts with”) of the filter.
`any`	Pointer to an array of the substrings (“contains”) for the filter.
`final`	Pointer to the final substring (“ends with”) of the filter.

Returns

This function returns one of the following values:

- `0` if successful.
- `-1` if the filter is not one of the types listed above.

Description

Filters of the type `LDAP_FILTER_SUBSTRINGS` generally compare a set of substrings against an attribute. For example:

```
(cn=John*Q*Public)
```

This filter finds entries in which the value of the `cn` attribute starts with `John`, contains `Q`, and ends with `Public`.

Call this function to get these substring values as well as the attribute type from this filter. In the case of the example above, calling this function gets the `initial` substring `John`, the `any` substring `Q`, and the `final` substring `Public` in addition to the attribute type `cn`.

See Also

`slapi_filter_get_attribute_type()`

`slapi_filter_get_ava()`

`slapi_filter_get_choice()`

slapi_filter_get_type()

(Applies only to filters of the type `LDAP_FILTER_PRESENT`) Gets the attribute type specified in the filter.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_type( Slapi_Filter *f, char **type );
```

Parameters

This function takes the following parameters:

Parameter	Description
`f`	Filter from which you want to get the substring values.
`type`	Pointer to the attribute type of the filter.

Returns

This function returns 0 if successful, or -1 if the filter is not one of the types listed above.

Description

Filters of the type `LDAP_FILTER_PRESENT` generally determine if a specified attribute is assigned a value. For example:

```
(mail=*)
```

This filter finds entries that have a value assigned to the `mail` attribute.

Call this function to get the attribute type from this filter. In the case of the example above, calling this function gets the attribute type `mail`.

Memory Concerns

The string returned in the parameter `type` must not be freed after calling this function. It will be freed when the structure `Slapi_Filter` is freed by calling `slapi_filter_free()`

See Also

`slapi_filter_get_attribute_type()`

`slapi_filter_get_ava()`

`slapi_filter_get_choice()`

slapi_filter_join()

Joins the two specified filters using one of the following filter types: `LDAP_FILTER_AND`, `LDAP_FILTER_OR`, or `LDAP_FILTER_NOT`. (When specifying the filter type `LDAP_FILTER_NOT`, the second filter should be NULL.)

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_join( int ftype, Slapi_Filter *f1,
    Slapi_Filter *f2 );
```

Parameters

This function takes the following parameters:

Parameter	Description
`ftype`	Type of composite filter you want to create.

f1	First filter that you want to join.
f2	Second filter that you want to join. If ftype is LDAP_FILTER_NOT, specify NULL for this argument.

Returns

This function returns the new filter constructed from the other two filters.

Description

Filters of the type LDAP_FILTER_AND, LDAP_FILTER_OR, and LDAP_FILTER_NOT generally consist of one or more other filters. For example:

```
(&(ou=Accounting)(l=Sunnyvale))
```

```
(|(ou=Accounting)(l=Sunnyvale))
```

```
(!(l=Sunnyvale))
```

Each of these examples contain one or more LDAP_FILTER_EQUALITY filters.

Call the slapi_filter_join() function to create a new filter of the type LDAP_FILTER_AND, LDAP_FILTER_OR, or LDAP_FILTER_NOT.

Memory Concerns

The f1 and f2 filters are not copied, nor freed, during the join process, but the resulting filter will have references pointing to these two filters.

slapi_filter_list_first()

(Applies only to filters of the types LDAP_FILTER_EQUALITY, LDAP_FILTER_GE, LDAP_FILTER_LE, LDAP_FILTER_APPROX) Gets the first filter that makes up the specified filter.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_list_first( Slapi_Filter *f );
```

Parameters

This function takes the following parameter:

Parameter	Description
f	Filter of which you wish to get the first component.

Returns

The first filter that makes up the specified filter `f`.

Description

To iterate through all filters that make up a specified filter, use this function in conjunction with the `slapi_filter_list_next()` function.

Filters of the type `LDAP_FILTER_AND`, `LDAP_FILTER_OR`, and `LDAP_FILTER_NOT` generally consist of one or more other filters. For example, if the filter is:

```
(&(ou=Accounting)(l=Sunnyvale))
```

the first filter in this list is:

```
(ou=Accounting)
```

Call the `slapi_filter_list_first()` function to get the first filter in the list.

Memory Concerns

No duplication of the filter is done, so this filter should not be freed independently of the original filter.

See Also

`slapi_filter_list_next()`

slapi_filter_list_next()

(Applies only to filters of the types `LDAP_FILTER_EQUALITY`, `LDAP_FILTER_GE`, `LDAP_FILTER_LE`, `LDAP_FILTER_APPROX`) Gets the next filter (following `fprev`) that makes up the specified filter `f`.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_list_next(Slapi_Filter *f,
    Slapi_Filter *fprev);
```

Parameters

This function takes the following parameters:

Parameter	Description
`f`	Filter from which you want to get the next component (after `fprev`).
`fprev`	Filter within the specified filter `f`.

Returns

The next filter (after `fprev`) that makes up the specified filter `f`.

Description

To iterate through all filters that make up a specified filter, use this function in conjunction with the `slapi_filter_list_first()` function.

Filters of the type `LDAP_FILTER_AND`, `LDAP_FILTER_OR`, and `LDAP_FILTER_NOT` generally consist of one or more other filters. For example, if the filter is:

```
(&(ou=Accounting)(l=Sunnyvale))
```

the next filter after `(ou=Accounting)` in this list is:

```
(l=Sunnyvale)
```

Call the `slapi_filter_list_next()` function to get the filters from this list.

Memory Concerns

No duplication of the filter is done, so the filter should not be freed independently of the original filter.

See Also

`slapi_filter_list_first()`

slapi_filter_test()

Determines if the specified entry matches a particular filter.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_test( Slapi_PBlock *pb, Slapi_Entry *e,
    Slapi_Filter *f, int verify_access );
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block.
`e`	Entry that you want to test.
`f`	Filter that you want to test the entry against.

verify_access	If 1, verifies that the current user has access rights to search the specified entry. If 0, bypasses any access control.

Returns

One of the following values:

- 0 if the entry matched the filter or if the specified filter is NULL.
- -1 if the filter type is unknown.
- A positive value (an LDAP error code) if an error occurred.

See Also

slapi_filter_test_simple()

slapi_filter_test_ext()

slapi_filter_test_ext()

Determines if an entry matches a given filter.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_test_ext( Slapi_PBlock *pb, Slapi_Entry *e,
    Slapi_Filter *f, int verify_access, int only_test_access)
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block from which the user is extracted
e	The entry on which filter matching must be verified.
f	The filter used for filter matching.
verify_access	0 when access checking is not to be done.
	1 when access checking must be done.
only_test_access	0 when filter matching must be done.
	1 when filter matching must not be done.

Returns

This function returns one of the following values:

- `0` if the entry matched the filter, or if the specified filter is NULL.
- `-1` if the filter type is unkown, or if the entry does not match the filter.
- A positive value (an LDAP error code) if an error occurred, or if the current user does not have access rights to search the specified entry.

Description

This function allows you to determine if an entry matches a given filter, and/or that the current user has the permission to access the entry.

See Also

`slapi_filter_test_simple()`

`slapi_filter_test()`

slapi_filter_test_simple()

Determines if an entry matches a filter.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_test_simple( Slapi_Entry *e, Slapi_Filter *f);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry that you wish to test.
`f`	Filter to match the entry against.

Returns

This function returns one of the following values:

- `0` if the entry matched the filter, or if the specified filter is `NULL`.
- `-1` if the filter type is unknown, or if the entry does not match the filter.
- A positive value (an LDAP error code) if an error occurred.

Description

This function allows you to check if entry `e` matches filter `f`.

See Also

`slapi_filter_test()`

`slapi_filter_test_ext()`

slapi_find_matching_paren()

Find a right parenthesis matching a left parenthesis.

Syntax

```
#include "slapi-plugin.h"
char *slapi_find_matching_paren( const char *str );
```

Parameters

This function takes the following parameters:

Parameter	Description
`str`	Pointer to a string starting with a left parenthesis

Returns

This function returns a pointer to the right parenthesis if successful. Otherwise, it returns `NULL` indicating that no matching right parenthesis was found.

Description

This function takes a pointer to a string starting with a left parenthesis, `(`, and returns a pointer to the matching right parenthesis, `)`. It may be useful when evaluating complex search filter strings.

slapi_free_search_results_internal()

Frees search results returned by the `slapi_search_internal_pb()` and `slapi_search_internal_callback_pb()` functions.

Syntax

```
#include "slapi-plugin.h"
void slapi_free_search_results_internal(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

Parameter	Description
`pb`	Parameter block returned by the `slapi_search_internal_pb()` and `slapi_search_internal_callback_pb()` functions.

Description

This function must be called when you are finished with the entries before freeing the parameter block.

slapi_free_suffix_list()

Free a list of directory suffixes.

Syntax

```
#include "slapi-plugin.h"
void slapi_free_suffix_list(Slapi_DN ** suffix_list);
```

Parameters

This function takes the following parameters:

Parameter	Description
`suffix_list`	Array of Distinguished Names for the suffixes to free

Description

This function frees each entry in `suffix_list`, and the `suffix_list` itself. It does not remove data from a database associated with the suffix.

slapi_get_first_backend()

Returns a pointer of the backend structure of the first backend.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend* slapi_get_first_backend(char **cookie);
```

Parameters

This function takes the following parameter:

Parameter	Description
cookie	Output parameter containing the index of the returned backed. This is useful for calls to slapi_get_next_backend(). Contains 0 in output if no backend is returned.

Returns

This function returns a pointer to the backend structure of the first backend, and its index, in the cookie parameter, or NULL if there is no backend.

Description

This function returns a pointer to the backend structure of the first backend. If you wish to iterate through all of the backends, use this function in conjunction with slapi_get_next_backend(). For example:

```
Slapi_Backend *be = NULL;
char *cookie = NULL;
be = slapi_get_first_backend (&cookie);
while (be)
    {
    ...
    be = slapi_get_next_backend (cookie);
    }
slapi_ch_free ((void**)&cookie);
```

Memory Concerns

Free the cookie parameter after the iteration using slapi_ch_free().

See Also

slapi_get_next_backend()

slapi_get_object_extension()

Access an object extension.

Syntax

```
#include "slapi-plugin.h"
void *slapi_get_object_extension(int objecttype, void *object,
    int extensionhandle);
```

Parameters

This function takes the following parameters:

Parameter	Description
`objecttype`	Type set by the server
`object`	Pointer to the object you extended
`extensionhandle`	Handle set by the server

Description

This function returns a pointer to the object extension registered using `slapi_register_object_extension()`.

See Also

`slapi_register_object_extension()`

slapi_get_next_backend()

Returns a pointer to the next backend.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend* slapi_get_next_backend(char *cookie);
```

Parameters

This function takes the following parameters:

Parameter	Description
`cookie`	Upon input, contains the index from which the search for the next backend is done. Upon output, contains the index of the returned backend.

Returns

This function returns a pointer to the next backend, if it exists, and updates the `cookie` parameter. Otherwise, it returns `NULL` and `cookie` is not changed.

Description

This function returns a pointer to the next backend. If you wish to iterate through all of the backends, use this function in conjunction with `slapi_get_first_backend()`. For example:

```
Slapi_Backend *be = NULL;
char *cookie = NULL;
be = slapi_get_first_backend (&cookie);
while (be )
    {
    ...
    be = slapi_get_next_backend (cookie);
    }
slapi_ch_free ((void**)&cookie);
```

Memory Concerns

Free the `cookie` parameter after the iteration using `slapi_ch_free()`.

See Also

`slapi_get_first_backend()`

`slapi_ch_free()`

slapi_get_supported_controls_copy()

Retrieves an allocated array of object identifiers (OIDs) representing the controls supported by the Directory Server. You can register new controls by calling `slapi_register_supported_control()`.

Syntax

```
#include "slapi-plugin.h"
int slapi_get_supported_controls_copy( char ***ctrloidsp,
    unsigned long **ctrlopsp );
```

Parameters

This function takes the following parameters:

Parameter	Description

`ctrloidsp`	Pointer to a character array that will receive the set of supported control OIDs. Pass `NULL` for this parameter if you do not wish to receive the OIDs.
`ctrlopsp`	Pointer to an unsigned long array that will receive the supported operation values for each control in the `ctrloidsp` array. Pass `NULL` for this parameter if you do not wish to receive the supported operation values.

Returns

This function returns `0` if successful, or a non-zero value if an error occurs.

Description

This function replaces the deprecated `slapi_get_supported_controls()` function from previous releases, as it was not multithread safe.

When you call `slapi_register_supported_control()` to register a control, you specify the OID of the control and the IDs of the operations that support the control. The server records this information in two arrays; an array of control OIDs, and an array of operations that support the control. You can get copies of these arrays by calling `slapi_get_supported_controls_copy()`.

For each OID returned in the `ctrloidsp` array, the corresponding array element (with the same index) in the `ctrlopsp` array identifies the operations that support the control. For a list of the possible IDs for the operations, see `slapi_register_supported_controls()`.

Memory Concerns

The returned `ctrloidsp` array should be freed by calling `slapi_ch_array_free()`. The returned `ctrlopsp` array should be freed by calling slapi_ch_free().

See Also

`slapi_register_supported_control()`

`slapi_ch_array_free()`

slapi_get_supported_extended_ops_copy()

Gets a copy of the object IDs (OIDs) of the extended operations.

Syntax

```
#include "slapi-plugin.h"
char **slapi_get_supported_extended_ops_copy ( void );
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to an array of the OIDs of the extended operations supported by the server.

Description

This function replaces the deprecated `slapi_get_supported_extended_ops()` function from earlier releases, as `slapi_get_supported_extended_ops()` was not multithread safe.

This function gets a copy of the object IDs (OIDs) of the extended operations supported by the server. You can register new extended operations by putting the OID in the `SLAPI_PLUGIN_EXT_OP_OIDLIST` parameter and calling `slapi_block_set()`.

Memory Concerns

The array returned by this function should be freed by calling the `slapi_ch_array_free()` function.

See Also

`slapi_pblock_set()`

`slapi_ch_array_free()`

slapi_get_supported_saslmechanisms_copy()

Gets an array of the names of the supported Simple Authentication and Security Layer (SASL) mechanisms. You can register new SASL mechanisms by calling the `slapi_vattr_values_free()` function.

Syntax

```
#include "slapi-plugin.h"
char ** slapi_get_supported_saslmechanisms_copy( void );
```

Returns

This function returns a pointer to an array of the names of SASL mechanisms supported by the server.

slapi_has8thBit()

Checks if a sting has an 8-bit character.

Syntax

```
#include "slapi-plugin.h"
int slapi_has8thBit(unsigned char *s);
```

Parameters

This function takes the following parameter:

Parameter	Description
s	Pointer to the null-terminated string to test.

Returns

This function returns `1` if the string contains an 8-bit character, or `0` if it does not.

slapi_is_rootdse()

This function determines if an entry is the root DSE. The root DSE is a special entry that contains information about the Directory Server, including its capabilities and configuration.

Syntax

```
#include "slapi-plugin.h"
int slapi_is_rootdse ( const char *dn );
```

Parameters

This function takes the following parameters:

Parameter	Description
dn	The DN that you want to test to see if it is the root DSE entry.

Returns

This function returns `1` if `dn` is the root DSE; otherwise the function returns `0`.

slapi_is_root_suffix()

Checks if a suffix is a root suffix of the DIT.

Syntax

```
#include "slapi-plugin.h"
int slapi_is_root_suffix(Slapi_DN * dn);
```

Parameters

This function takes the following parameter:

Parameter	Description
dn	DN that you wish to check.

Returns

This function returns one of the following values:

- 0 if the DN is not a root suffix
- 1 if the DN is a root suffix

slapi_ldap_init()

Get a thread-safe handle to an LDAP connection.

Syntax

```
#include "slapi-plugin.h"
LDAP *slapi_ldap_init(char *ldaphost, int ldapport, int secure,
    int shared);
```

Parameters

This function takes the following parameters:

Parameter	Description
ldaphost	Host on which the LDAP server is running
ldapport	Port on which the LDAP server is listening
secure	1 for a secure connection over SSL, NULL otherwise
shared	If not NULL, then the connection may be shared between threads

Description

This function allows a plug-in to retrieve a thread-safe handle to an LDAP connection. When done with the handle, call `slapi_ldap_unbind()`.

A timeout may be set for the connection using the Sun ONE Directory Server Resource Kit. Code Example 2-1 demonstrates how to set a timeout.

Code Example 2-1 Setting a Timeout

```
#include "slapi-plugin.h"
#include "ldap.h"

void
my_ldap_function(void)
{
    LDAP * ld;
    int    to = 5000;                  /* 5000 ms == 5 s timeout */

    if ((ld = slapi_ldap_init(host, port, 0, 1)) == NULL) {
        /* error trying to create an LDAP session */
        return -1;
    }

    if (ldap_set_option(ld, LDAP_X_OPT_CONNECT_TIMEOUT, &to) != 0) {
        /* error setting timeout */
        slapi_ldap_unbind(ld);
        return -1;
    }

    /* Use the handle for a search for example. */

    slapi_ldap_unbind(ld);
    return 0;
}
```

Refer to “Downloading Directory Server Tools,” on page 8 for information on how to obtain the Sun ONE Directory Server Resource Kit.

Returns

This function returns an LDAP connection handle if successful. Otherwise, it returns `NULL`.

See Also

`slapi_ldap_unbind()`

slapi_ldap_unbind()

Release an LDAP connection obtained using `slapi_ldap_init()`.

Syntax

```
#include "slapi-plugin.h"
void slapi_ldap_unbind( LDAP *ld );
```

Parameters

This function takes the following parameters:

Parameter	Description
`ld`	Handle to the LDAP connection

Description

This function allows a plug-in to release an LDAP connection obtained using `slapi_ldap_init()`.

See Also

`slapi_ldap_init()`

slapi_lock_mutex()

Lock a mutex.

Syntax

```
#include "slapi-plugin.h"
void slapi_lock_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

Parameter	Description
`mutex`	Mutex for thread synchronization

Description

This function allows thread synchronization. Once a thread has locked a mutex using this function, other threads attempting to acquire the lock are blocked until the thread holding the mutex calls `slapi_unlock_mutex()`.

See Also

`slapi_destroy_mutex()`

`slapi_new_mutex()`

`slapi_unlock_mutex()`

slapi_log_error_ex()

Write an error message to the server error log.

Syntax

```
#include "slapi-plugin.h"
int slapi_log_error_ex(long errorId,
    long msgId, int connId, int opId, char * context,
    char * message, char * format, /* args */ ...);
```

Parameters

This function takes the following parameters:

Parameter	Description
`errorId`	Unique identifier you provide for this error message
`msgId`	Identifier for the current message obtained using `SLAPI_OPERATION_MSGID`
`connId`	Identifier for the current connection obtained using `SLAPI_CONN_ID`
`opId`	Identifier for the current operation obtained using `SLAPI_OPERATION_ID`
`context`	String indicating the context in which the error arose such as the name of the plug-in function logging the message
`message`	String message identifying this error
`format`	Format specification in the style of `printf()`
`args`	Arguments for the format specification in `format`

Description

This function writes the specified error message to the server error log in synchronous fashion. This function does not return until the log message has been flushed to disk, thus blocking the server for the duration of the write operation. By default, the error log is *ServerRoot*/slapd-*serverID*/logs/errors.

Unlike slapi_log_info_ex(), this function cannot be turned off.

Error messages typically concern fatal errors. For warnings, use slapi_log_warning_ex(). For informational log messages, use slapi_log_info_ex().

Example

Code Example 2-2 shows a call to slapi_log_error_ex().

Code Example 2-2 Logging an Error

```
#include "slapi-plugin.h"
#include "example-com-error-ids.h" /* example.com unique
                                      error IDs file */
int
foobar(Slapi_PBlock * pb)
{
    char * error_cause;
    int    apocalypse = 1;         /* Expect the worst. */

    /* ... */

    if (apocalypse) {              /* Server to crash soon */
        slapi_log_error_ex(
            EXCOM_SERVER_MORIBUND, /* Unique error ID */
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "example.com: foobar in baz plug-in",
            "cannot write to file system: %s\n",
            error_cause
        );
        return -1;
    }
    return 0;
}
```

Returns

This function returns 0 if successful. Otherwise, it returns -1.

See Also

`slapi_log_info_ex()`

`slapi_log_warning_ex()`

slapi_log_info_ex()

Write an informational message to the server error log.

Syntax

```
#include "slapi-plugin.h"
int slapi_log_info_ex(slapi_log_info_area_t area,
    slapi_log_info_level_t level,
    long msgId, int connId, int opId, char * context,
    char * format, /* args */, ...);
```

Parameters

This function takes the following parameters:

Parameter	Description
`area`	Identifies the server component so logging can be turned on by adding the decimal value of the area to the value for `nsslapd-infolog-area`. Subtract from the value to turn informational logging off.
`level`	Identifies whether the server should log this message when informational logging is activated for `area`. When informational logging is activated, setting level to: • `SLAPI_LOG_INFO_LEVEL_DEFAULT` means always log the message. • `SLAPI_LOG_INFO_LEVEL_EXTRA` means only log if the value of `nsslapd-infolog-level` is greater than 0.
`msgId`	Identifier for the current message obtained using `SLAPI_OPERATION_MSGID`
`connId`	Identifier for the current connection obtained using `SLAPI_CONN_ID`
`opId`	Identifier for the current operation obtained using `SLAPI_OPERATION_ID`
`context`	String indicating the context in which the message arose such as the name of the plug-in function logging the message

`format`	Format specification in the style of `printf()`
`args`	Arguments for the format specification in `format`

Description

This function writes the specified error message to the server error log in synchronous fashion. This function does not return until the log message has been flushed to disk, thus blocking the server for the duration of the write operation. By default, the error log is *ServerRoot*`/slapd-`*serverID*`/logs/errors`.

This function is turned off by default. Activate logging of the message by adding the decimal representation of `area` to the value of `nsslapd-infolog-area` on `cn=config` for the directory. The following command enables logging for messages with `area SLAPI_LOG_INFO_AREA_PLUGIN` and `level SLAPI_LOG_INFO_LEVEL_DEFAULT`:

```
$ ldapmodify -p port -D "cn=directory manager" -w password
dn: cn=config
changetype: modify
replace: nsslapd-infolog-area
nsslapd-infolog-area: 65536
```

The directory console offers a less arcane interface for activating and deactivating informational logging.

Informational message are typically those that system administrators may ignore unless trying to debug server behavior. For errors, use `slapi_log_error_ex()`. For warnings, use `slapi_log_warning_ex()`.

Example

Code Example 2-3 shows a call to `slapi_log_info_ex()`.

Code Example 2-3 Logging an Informational Message

```
#include "slapi-plugin.h"

int
hello()
{
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        SLAPI_LOG_NO_MSGID,
        SLAPI_LOG_NO_CONNID,
        SLAPI_LOG_NO_OPID,
        "hello() from a plug-in",
        "Hello, World!\n"
```

Code Example 2-3 Logging an Informational Message

```
    );
    return 0;
}
```

Returns

This function returns `0` if successful. Otherwise, it returns `-1`.

See Also

`slapi_log_error_ex()`

`slapi_log_warning_ex()`

slapi_log_warning_ex()

Write a warning message to the server error log.

Syntax

```
#include "slapi-plugin.h"
int slapi_log_warning_ex(long warningId,
    long msgId, int connId, int opId, char * context,
    char * message, char * format, /* args */ ...);
```

Parameters

This function takes the following parameters:

Parameter	Description
`warningId`	Unique identifier you provide for this warning message
`msgId`	Identifier for the current message obtained using `SLAPI_OPERATION_MSGID`
`connId`	Identifier for the current connection obtained using `SLAPI_CONN_ID`
`opId`	Identifier for the current operation obtained using `SLAPI_OPERATION_ID`

`context`	String indicating the context in which the warning arose such as the name of the plug-in function logging the message
`message`	String message identifying this warning
`format`	Format specification in the style of `printf()`
`args`	Arguments for the format specification in `format`

Description

This function writes the specified error message to the server error log in synchronous fashion. This function does not return until the log message has been flushed to disk, thus blocking the server for the duration of the write operation. By default, the error log is *ServerRoot*`/slapd-`*serverID*`/logs/errors`.

Unlike `slapi_log_info_ex()`, this function cannot be turned off.

Warning messages typically concern potentially serious situations, but not fatal errors. For fatal errors, use `slapi_log_error_ex()`. For informational log messages, use `slapi_log_info_ex()`.

Example

Code Example 2-4 shows a call to `slapi_log_warning_ex()`.

Code Example 2-4 Logging a Warning

```
#include "slapi-plugin.h"
#include "example-com-warning-ids.h" /* example.com unique
                                        warning IDs file */
int
foobar()
{
    int disk_use_percentage;

    /* ... */

    if (disk_use_percentage >= 95){
        slapi_log_warning_ex(
            EXCOM_DISK_FULL_WARN,    /* unique warning ID */
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "example.com: foobar in baz plug-in",
            "disk %.0f%% full, find more space\n",
            disk_use_percentage
        );
    }
```

Code Example 2-4 Logging a Warning

```
    return 0;
}
```

Returns

This function returns `0` if successful. Otherwise, it returns `-1`.

See Also

`slapi_log_error_ex()`

`slapi_log_info_ex()`

slapi_matchingrule_free()

Free a `Slapi_MatchingRuleEntry` after registering the matching rule.

Syntax

```
#include "slapi-plugin.h"
void slapi_matchingrule_free(Slapi_MatchingRuleEntry **mrEntry,
    int freeMembers);
```

Parameters

This function takes the following parameters:

Parameter	Description
`mrEntry`	Matching rule registration object
`freeMembers`	Whether to free the members of the `Slapi_MatchingRuleEntry` If `0`, do not free the members of the `Slapi_MatchingRuleEntry`.

Description

This function frees memory allocated to a `Slapi_MatchingRuleEntry` after the structure has been used to register a matching rule.

See Also

`slapi_matchingrule_new()`

`slapi_matchingrule_register()`

slapi_matchingrule_get()

Access a `Slapi_MatchingRuleEntry` member.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_get(Slapi_MatchingRuleEntry *mr, int arg,
    void *value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`mr`	Matching rule registration object
`arg`	Identifier for the `Slapi_MatchingRuleEntry` member: • `SLAPI_MATCHINGRULE_DESC`, a string describing the matching rule • `SLAPI_MATCHINGRULE_NAME`, a string identifying the matching rule • `SLAPI_MATCHINGRULE_OID`, a string representing the matching rule object identifier • `SLAPI_MATCHINGRULE_SYNTAX`, the matching rule syntax OID string `1.3.6.1.4.1.1466.115.121.1.15` • `SLAPI_MATCHINGRULE_OBSOLETE`, an `int` identifying whether the rule is obsolete
`value`	Value retrieved from the member

Description

This function accesses a `Slapi_MatchingRuleEntry` member based on the identifier in `arg`.

Returns

This function returns `0` if successful. Otherwise, it returns `-1`.

See Also

`slapi_matchingrule_register()`

`slapi_matchingrule_set()`

slapi_matchingrule_new()

Allocate a `Slapi_MatchingRuleEntry`.

Syntax

```
#include "slapi-plugin.h"
Slapi_MatchingRuleEntry *slapi_matchingrule_new(void);
```

Description

This function allocates a `Slapi_MatchingRuleEntry` used to register a matching rule.

Returns

This function returns a pointer to the matching rule registration object if successful. Otherwise, it returns `NULL`.

See Also

`slapi_matchingrule_free()`

`slapi_matchingrule_register()`

slapi_matchingrule_register()

Register a matching rule with the server.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_register(Slapi_MatchingRuleEntry *mrEntry);
```

Parameters

This function takes the following parameters:

Parameter	Description
`mrEntry`	Matching rule registration object

Description

This function registers a `Slapi_MatchingRuleEntry` with the server. Register matching rules as part of the plug-in initialization function.

First, allocate the structure using `slapi_matchingrule_new()`. Next, set the members of the matching rule entry using `slapi_matchingrule_set()`. After setting the members, register the matching rule with the server using this function. Finally, free the memory allocated using `slapi_matchingrule_free()`.

Returns

This function returns `0` if successful. Otherwise, it returns `-1`.

See Also

`slapi_matchingrule_free()`

`slapi_matchingrule_get()`

`slapi_matchingrule_new()`

`slapi_matchingrule_set()`

slapi_matchingrule_set()

Modify a `Slapi_MatchingRuleEntry` member.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_set(Slapi_MatchingRuleEntry *mr, int arg,
    void *value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`mr`	Matching rule registration object

`arg`	Identifier for the `Slapi_MatchingRuleEntry` member: • `SLAPI_MATCHINGRULE_DESC`, a string describing the matching rule • `SLAPI_MATCHINGRULE_NAME`, a string identifying the matching rule • `SLAPI_MATCHINGRULE_OID`, a string representing the matching rule object identifier • `SLAPI_MATCHINGRULE_SYNTAX`, the matching rule syntax OID string `1.3.6.1.4.1.1466.115.121.1.15` • `SLAPI_MATCHINGRULE_OBSOLETE`, an `int` identifying whether the rule is obsolete
`value`	Value to affect to the member

Description

This function modifies a `Slapi_MatchingRuleEntry` member based on the identifier in `arg`.

Returns

This function returns `0` if successful. Otherwise, it returns `-1`.

See Also

`slapi_matchingrule_get()`

`slapi_matchingrule_register()`

slapi_mod_add_value()

Adds a value to a `Slapi_Mod` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_add_value(Slapi_Mod *smod, const struct berval *val);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smod`	Pointer to an initialized `Slapi_Mod`.

`val`	Pointer to a `berval` representing the attribute value.

Description

Adds a copy of a given attribute to the `Slapi_Mod`.

slapi_mod_done()

Frees the internals of `Slapi_Mod` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_done(Slapi_Mod *mod);
```

Parameters

This function takes the following parameter:

Parameter	Description
`mod`	Pointer to a `Slapi_Mod`.

Description

This function frees the internals of a `Slapi_Mod`, leaving it in the uninitialized state.

Memory Concerns

Use this function on a stack-allocated `Slapi_Mod` when you have finished with it, or wish to reuse it.

See Also

`slapi_mod_init()`

`slapi_mod_init_byval()`

`slapi_mod_init_byref()`

`slapi_mod_init_passin()`

slapi_mod_dump()

Dumps the contents of an `LDAPMod` to the server log.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_dump(LDAPMod *mod, int n);
```

Parameters

This function takes the following parameters:

Parameter	Description
mod	Pointer to an LDAPMod.
n	Numeric label that will be included in the log.

Description

This function uses the LDAP_DEBUG_ANY log level to dump the contents of an LDAPMod to the server log for debugging.

slapi_mod_free()

Frees a Slapi_Mod structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_free(Slapi_Mod **smod);
```

Parameters

This function takes the following parameter:

Parameter	Description
smod	Pointer to an initialized Slapi_Mod.

Description

This function frees a Slapi_Mod structure that was allocated by slapi_mod_new().

See Also

slapi_mod_new()

slapi_mod_get_first_value()

Initializes a Slapi_Mod iterator and returns the first attribute value.

Syntax

```
#include "slapi-plugin.h"
struct berval *slapi_mod_get_first_value(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

Parameter	Description
smod	Pointer to an initialized Slapi_Mod.

Returns

This function returns a pointer to the first attribute value in the Slapi_Mod, or NULL if no values exist.

Description

Use this function with slapi_mod_get_next_value() to iterate through the attribute values in a Slapi_Mod structure.

See Also

slapi_mod_get_next_value()

slapi_mod_get_ldapmod_byref()

Gets a reference to the LDAPMod in a Slapi_Mod structure.

Syntax

```
#include "slapi-plugin.h"
const LDAPMod *slapi_mod_get_ldapmod_byref(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

Parameter	Description
smod	Pointer to an initialized Slapi_Mod.

Returns
This function returns a pointer to a read-only `LDAPMod` owned by the `Slapi_Mod`.

Description
Use this function to get direct access to the `LDAPMod` contained in a `Slapi_Mod`.

Memory Concerns
Responsibility for the `LDAPMod` remains with the `Slapi_Mod`.

See Also
`slapi_mod_get_ldapmod_passout()`

slapi_mod_get_ldapmod_passout()
Retrieves the `LDAPMod` contained in a `Slapi_Mod` structure.

Syntax
```
#include "slapi-plugin.h"
LDAPMod *slapi_mod_get_ldapmod_passout(Slapi_Mod *smod);
```

Parameters
This function takes the following parameter:

Parameter	Description
`smod`	Pointer to an initialized `Slapi_Mod`.

Returns
This function returns a pointer to an `LDAPMod` owned by the caller.

Description
Use this function to get the `LDAPMod` out of a `Slapi_Mod`.

Memory Concerns
Responsibility for the `LDAPMod` transfers to the caller. The `Slapi_Mod` is left in the uninitialized state.

See Also
`slapi_mod_get_ldapmod_byref()`

slapi_mod_get_next_value()

Increments the Slapi_Mod iterator and returns the next attribute value.

Syntax

```
#include "slapi-plugin.h"
struct berval *slapi_mod_get_next_value(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

Parameter	Description
smod	Pointer to an initialized Slapi_Mod.

Returns

This function returns a pointer to the next attribute value in the Slapi_Mod, or NULL if there are no more.

Description

Use this function with slapi_mods_get_first_mod() to iterate through the attribute values in a Slapi_Mod.

See Also

slapi_mods_get_first_mod()

slapi_mod_get_num_values()

Gets the number of values in a Slapi_Mod structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_mod_get_num_values(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

Parameter	Description
smod	Pointer to an initialized Slapi_Mod.

Returns

This function returns the number of attribute values in the `Slapi_Mod`.

slapi_mod_get_operation()

Gets the operation type of `Slapi_Mod` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_mod_get_operation(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

Parameter	Description
`smod`	Pointer to an initialized `Slapi_Mod`.

Returns

This function returns one of `LDAP_MOD_ADD`, `LDAP_MOD_DELETE`, `LDAP_MOD_REPLACE`, combined using the bitwise or operator with `LDAP_MOD_BYVALUES`.

See Also

`slapi_mod_set_operation()`

slapi_mod_get_type()

Gets the attribute type of a `Slapi_Mod` structure.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_mod_get_type(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

Parameter	Description

smod	Pointer to an initialized Slapi_Mod.

Returns

This function returns a read-only pointer to the attribute type in the Slapi_Mod.

Description

Gets the LDAP attribute type of a Slapi_Mod.

See Also

slapi_mod_set_type()

slapi_mod_init()

Initializes a Slapi_Mod structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init(Slapi_Mod *smod, int initCount);
```

Parameters

This function takes the following parameters:

Parameter	Description
smod	Pointer to an uninitialized Slapi_Mod.
initCount	Suggested number of attribute values for which to make room. Minimum value is 0.

Description

This function initializes a Slapi_Mod so that it is empty, but initially has room for the given number of attribute values.

Memory Concerns

If you are unsure of the room you will need, you may use an initCount of 0. The Slapi_Mod expands as necessary.

See Also

slapi_mod_done()

slapi_mod_init_byval()

slapi_mod_init_byref()

slapi_mod_init_passin()

slapi_mod_init_byref()

Initializes a Slapi_Mod structure that is a wrapper for an existing LDAPMod.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_byref(Slapi_Mod *smod, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

Parameter	Description
smod	Pointer to an uninitialized Slapi_Mod.
mod	Pointer to an LDAPMod.

Description

This function initializes a Slapi_Mod containing a reference to an LDAPMod. Use this function when you have an LDAPMod and would like the convenience of the Slapi_Mod functions to access it.

See Also

slapi_mod_done()

slapi_mod_init()

slapi_mod_init_byval()

slapi_mod_init_passin()

slapi_mod_init_byval()

Initializes a Slapi_Mod structure with a copy of an LDAPMod.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_byval(Slapi_Mod *smod, const LDAPMod *mod);
```

Parameters

This function takes the following parameters:

Parameter	Description
smod	Pointer to an uninitialized Slapi_Mod.
mod	Pointer to an LDAPMod.

See Also

slapi_mod_done()

slapi_mod_init()

slapi_mod_init_byref()

slapi_mod_init_byval()

slapi_mod_init_passin()

Initializes a Slapi_Mod from an LDAPMod.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_passin(Slapi_Mod *smod, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

Parameter	Description
smod	Pointer to an uninitialized Slapi_Mod.
mod	Pointer to an LDAPMod.

Description

This function initializes a Slapi_Mod by passing in an LDAPMod. Use this function to convert an LDAPMod to a Slapi_Mod.

Memory Concerns

Responsibility for the LDAPMod is transferred to the Slapi_Mod. The LDAPMod is destroyed when the Slapi_Mod is destroyed.

See Also

slapi_mod_done()

slapi_mod_init()

slapi_mod_init_byval()

slapi_mod_init_byref()

slapi_mod_isvalid()

Determines whether a Slapi_Mod structure is valid.

Syntax

```
#include "slapi-plugin.h"
int slapi_mod_isvalid(const Slapi_Mod *mod);
```

Parameters

This function takes the following parameters:

Parameter	Description
smod	Pointer to a Slapi_Mod.

Returns

This function returns one of the following values:

- 1 if the Slapi_Mod is valid.
- 0 if the Slapi_Mod is not valid.

Description

Use this function to verify that the contents of Slapi_Mod are valid. It is considered valid if the operation type is one of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE, combined using the bitwise or operator with LDAP_MOD_BYVALUES; the attribute type is not NULL; and there is at least one attribute value for add and replace operations.

slapi_mod_new()

Allocates a new `Slapi_Mod` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mod* slapi_mod_new( void );
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to an allocated, uninitialized `Slapi_Mod`.

Description

This function allocates a new uninitialized `Slapi_Mod`. Use this function when you need to a `Slapi_Mod` allocated from the heap, rather than from the stack.

See Also

`slapi_mod_free()`

slapi_mod_remove_value()

Removes the value at the current `Slapi_Mod` iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_remove_value(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

Parameter	Description
`smod`	Pointer to an initialized `Slapi_Mod`.

See Also

`slapi_mod_get_first_value()`

`slapi_mod_get_next_value()`

slapi_mod_set_operation()

Sets the operation type of a `Slapi_Mod` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_set_operation(Slapi_Mod *smod, int op);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smod`	Pointer to an initialized `Slapi_Mod`.
`op`	One of `LDAP_MOD_ADD`, `LDAP_MOD_DELETE`, `LDAP_MOD_REPLACE`, combined using the bitwise or operator with `LDAP_MOD_BYVALUES`.

See Also

`slapi_mod_get_operation()`

slapi_mod_set_type()

Sets the attribute type of a `Slapi_Mod`.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_set_type(Slapi_Mod *smod, const char *type);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smod`	Pointer to an initialized `Slapi_Mod`.
`type`	An attribute type.

Description

Sets the attribute type of the `Slapi_Mod` to a copy of the given value.

See Also

`slapi_mod_get_type()`

slapi_moddn_get_newdn()

Builds the new DN of an entry.

Syntax

```
#include "slapi-plugin.h"
char * slapi_moddn_get_newdn(Slapi_DN *dn_olddn, char *newrdn,
    char *newsuperiordn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`dn_olddn`	The old DN value.
`newrdn`	The new RDN value.
`newsuperordn`	If not NULL, will be the DN of the future superior entry of the new DN, which will be worked out by adding the value in `newrdn` in front of the content of this parameter.

Returns

This function returns the new DN for the entry whose previous DN was `dn_olddn`.

Description

This function is used for `moddn` operations and builds a new DN out of a new RDN and the DN of the new parent.

The new DN is worked out by adding the new RDN in `newrdn` to a parent DN. The parent will be the value in `newsuperiordn`, if different from `NULL`, and will otherwise be taken from `dn_olddn` by removing the old RDN (the parent of the entry will still be the same as the new DN).

slapi_modify_internal_pb()

Performs an LDAP modify operation based on a parameter block to modify a directory entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_modify_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

Parameter	Description
pb	A parameter block that has been initialized using `slapi_modify_internal_set_pb()`.

Returns

This function returns `-1` if the parameter passed is a `NULL` pointer. Otherwise, it returns `0`.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

Description

This function performs an internal modify operation based on a parameter block. The parameter block should be initialized by calling `slapi_modify_internal_set_pb()`.

Memory Concerns

None of the parameters that are passed to `slapi_modify_internal_set_pb()` are altered or consumed by this function.

slapi_modify_internal_set_pb()

Prepares a parameter block for an internal modify operation.

Syntax

```
#include "slapi-plugin.h"
int slapi_modify_internal_set_pb(Slapi_PBlock *pb,
    const char *dn, LDAPMod **mods, LDAPControl **controls,
    const char *uniqueid, Slapi_ComponentId *plugin_identity,
    int operation_flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block for the internal modify operation
dn	Distinguished Name of the entry to modify
mods	Array of modifications to apply
controls	Array of controls to request for the modify operation
uniqueid	Unique identifier for the entry if using this rather than DN
plugin_identity	Plug-in identifier obtained from SLAPI_PLUGIN_IDENTITY during plug-in initialization
operation_flags	Flags such as SLAPI_OP_FLAG_NEVER_CHAIN

Returns

This function returns 0 if successful. Otherwise, it returns an LDAP error code.

Description

This function prepares a parameter block for use with slapi_modify_internal_pb().

Memory Concerns

Allocate the parameter block using slapi_pblock_new() before calling this function.

Free the parameter block after calling slapi_modify_internal_pb().

See Also

slapi_modify_internal_pb()

slapi_pblock_new()

slapi_modrdn_internal_pb()

Performs an LDAP modify RDN operation based on a parameter block to rename a directory entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_modrdn_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

Parameter	Description
pb	A parameter block that has been initialized using `slapi_rename_internal_set_pb()`.

Returns

This function returns `-1` if the parameter passed is a `NULL` pointer. Otherwise, it returns `0`.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

Description

This function performs an internal modify RDN operation based on a parameter block. The parameter block should be initialized by calling `slapi_rename_internal_set_pb()`.

Memory Concerns

None of the parameters that are passed to `slapi_modrdn_internal_set_pb()` are altered or consumed by this function.

See Also

`slapi_rename_internal_set_pb()`

slapi_mods2entry()

Creates a `Slapi_Entry` from an array of `LDAPMod`.

Syntax

```
#include "slapi-plugin.h"
int slapi_mods2entry(Slapi_Entry **e, const char *dn,
    LDAPMod **attrs);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Address of a pointer that will be set on return to the created entry.
dn	The LDAP DN of the entry.
attrs	An array of LDAPMod of type LDAP_MOD_ADD representing the entry attributes.

Returns

This function returns LDAP_SUCCESS if successful, or an LDAP return code if not successful.

Description

This function creates a Slapi_Entry from a copy of an array of LDAPMod of type LDAP_MODD_ADD.

See Also

slapi_entry2mods()

slapi_mods_add()

Appends a new mod with a single attribute value to Slapi_Mods structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add( Slapi_Mods *smods, int modtype,
    const char *type, unsigned long len, const char *val);
```

Parameters

This function takes the following parameters:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.
modtype	One of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE.
type	The LDAP attribute type.

len	The length in bytes of the attribute value.
val	The attribute value.

Description

This function appends a new mod with a single attribute value to a Slapi_Mods. The mod is constructed from copies of the values of modtype, type, len, and val.

Memory Concerns

This function must not be used on Slapi_Mods initialized with slapi_mods_init_byref().

See Also

slapi_mods_add_ldapmod()

slapi_mods_add_modbvps()

sslapi_mods_add_mod_values()

slapi_mods_add_string()

slapi_mods_add_ldapmod()

Appends an LDAPMod to a Slapi_Mods structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_ldapmod(Slapi_Mods *smods, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.
mod	Pointer to a the LDAPMod to be appended.

Description

Appends an LDAPMod to a Slapi_Mods.

Memory Concerns

Responsibility for the LDAPMod is transferred to the Slapi_Mods.

This function must not be used on a Slapi_Mods initialized with slapi_mods_init_byref().

See Also

slapi_mods_add()

slapi_mods_add_modbvps()

slapi_mods_add_mod_values()

slapi_mods_add_string()

slapi_mods_add_mod_values()

Appends a new mod to a Slapi_Mods structure, with attribute values provided as an array of Slapi_Value.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_mod_values( Slapi_Mods *smods, int modtype,
    const char *type, Slapi_Value **va );
```

Parameters

This function takes the following parameters:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.
modtype	One of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE.
type	The LDAP attribute type.
va	A null-terminated array of Slapi_Value representing the attribute values.

Description

This function appends a new mod to a Slapi_Mods. The mod is constructed from copies of the values of modtype, type and va. Use this function when you have the attribute values to hand as an array of Slapi_Value.

Memory Concerns

This function must not be used on a `Slapi_Mods` initialized with `slapi_mods_init_byref()`.

See Also

`slapi_mods_add()`

`slapi_mods_add_ldapmod()`

`slapi_mods_add_modbvps()`

`slapi_mods_add_string()`

slapi_mods_add_modbvps()

Appends a new `mod` to a `Slapi_Mods` structure, with attribute values provided as an array of `berval`.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_modbvps( Slapi_Mods *smods, int modtype,
    const char *type, struct berval **bvps );
```

Parameters

This function takes the following parameters:

Parameter	Description
`smods`	Pointer to an initialized `Slapi_Mods`.
`modtype`	One of `LDAP_MOD_ADD`, `LDAP_MOD_DELETE`, `LDAP_MOD_REPLACE`.
`type`	The LDAP attribute type.
`bvps`	A null-terminated array of `berval` representing the attribute values.

Description

This function appends a new `mod` to `Slapi_Mods`. The `mod` is constructed from copies of the values of `modtype`, `type` and `bvps`. Use this function when you have the attribute values to hand as an array of `berval`.

Memory Concerns

This function must not be used on a `Slapi_Mods` initialized with `slapi_mods_init_byref()`.

See Also

`slapi_mods_add()`

`slapi_mods_add_ldapmod()`

`slapi_mods_add_mod_values()`

`slapi_mods_add_string()`

slapi_mods_add_smod()

Appends a `Slapi_Mod` to a `Slapi_Mods` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_smod(Slapi_Mods *smods, Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smods`	Pointer to an initialized `Slapi_Mods`.
`smod`	Pointer to a the `Slapi_Mod` to be appended.

Description

Appends a `Slapi_Mod` to a `Slapi_Mods`.

Memory Concerns

Responsibility for the `Slapi_Mod` is transferred to the `Slapi_Mods`.

This function must not be used on a `Slapi_Mods` initialized with `slapi_mods_init_byref()`.

See Also

`slapi_mods_add()`

`slapi_mods_add_modbvps()`

`slapi_mods_add_mod_values()`

`slapi_mods_add_string()`

slapi_mods_add_string()

Appends a new mod to Slapi_Mods structure with a single attribute value provided as a string.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_string( Slapi_Mods *smods, int modtype,
    const char *type, const char *val);
```

Parameters

This function takes the following parameters:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.
modtype	One of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE.
type	The LDAP attribute type.
val	The attribute value represented as a NULL-terminated string.

Description

This function appends a new mod with a single string attribute value to a Slapi_Mods. The mod is constructed from copies of the values of modtype, type and val.

Memory Concerns

This function must not be used on a Slapi_Mods initialized with slapi_mods_init_byref().

See Also

slapi_mods_add()

slapi_mods_add_ldapmod()

slapi_mods_add_modbvps()

slapi_mods_add_mod_values()

slapi_mods_done()

Frees internals of a `Slapi_Mods` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_done(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

Parameter	Description
`smods`	Pointer to a `Slapi_Mods` structure.

Description

This function frees the internals of a `Slapi_Mods`, leaving it in the uninitialized state. Use this function on a stack-allocated `Slapi_Mods` when you are finished with it, or when you wish to reuse it.

See Also

`slapi_mods_init()`

`slapi_mods_init_byref()`

`slapi_mods_init_passin()`

slapi_mods_dump()

Dumps the contents of a `Slapi_Mods` structure to the server log.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_dump(const Slapi_Mods *smods, const char *text);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smods`	Pointer to a `Slapi_Mods`

`text`	Descriptive text that will be included in the log, preceding the `Slapi_Mods` content.

Description

This function uses the `LDAP_DEBUG_ANY` log level to dump the contents of a `Slapi_Mods` to the server log for debugging.

See Also

`slapi_mods_dump()`

slapi_mods_free()

Frees a `Slapi_Mods` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_free(Slapi_Mods **smods);
```

Parameters

This function takes the following parameter:

Parameter	Description
`smods`	Pointer to an allocated `Slapi_Mods`.

Description

This function frees a `Slapi_Mods` that was allocated by `slapi_mods_new()`.

See Also

slapi_mods_new()

slapi_mods_get_first_mod()

Initializes a `Slapi_Mods` iterator and returns the first `LDAPMod`.

Syntax

```
#include "slapi-plugin.h"
LDAPMod *slapi_mods_get_first_mod(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.

Returns

This function returns a pointer to the first LDAPMod in the Slapi_Mods, or NULL if there are no mods.

slapi_mods_get_first_smod()

Initializes a Slapi_Mods iterator and returns the first mod wrapped in a Slapi_Mods structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mod *slapi_mods_get_first_smod(Slapi_Mods *smods,
    Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

Parameter	Description
smods	A pointer to an initialized Slapi_Mods.
smod	Pointer to a Slapi_Mod that used to hold the mod.

Returns

This function returns a pointer to the Slapi_Mod, wrapping the first mod, or NULL if no mod exists.

Description

Use this function in conjunction with slapi_mods_get_next_smod() to iterate through the mods in a Slapi_Mods using a Slapi_Mods wrapper.

Memory Concerns

Only one thread may be iterating through a particular `Slapi_Mods` at any given time.

See Also

`slapi_mods_get_next_smod()`

slapi_mods_get_ldapmods_byref()

Gets a reference to the array of `LDAPMod` in a `Slapi_Mods` structure.

Syntax

```
#include "slapi-plugin.h"
LDAPMod **slapi_mods_get_ldapmods_byref(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

Parameter	Description
`smods`	Pointer to an initialized `Slapi_Mods`.

Returns

This function returns a null-terminated array of `LDAPMod` owned by the `Slapi_Mods`.

Description

Use this function to get direct access to the array of `LDAPMod` contained in a `Slapi_Mods`.

Memory Concerns

Responsibility for the array remains with the `Slapi_Mods`.

See Also

`slapi_mods_get_ldapmods_passout()`

slapi_mods_get_ldapmods_passout()

Retrieves the array of `LDAPMod` contained in a `Slapi_Mods` structure.

Syntax

```
#include "slapi-plugin.h"
LDAPMod **slapi_mods_get_ldapmods_passout(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.

Returns

A null-terminated array LDAPMod owned by the caller.

Description

Gets the array of LDAPMod out of a Slapi_Mods. Responsibility for the array transfers to the caller. The Slapi_Mods is left in the uninitialized state.

See Also

slapi_mods_get_ldapmods_byref()

slapi_mods_get_next_mod()

Increments the Slapi_Mods iterator and returns the next LDAPMod.

Syntax

```
#include "slapi-plugin.h"
LDAPMod *slapi_mods_get_next_mod(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.

Returns

This function returns a pointer to the next LDAPMod, or NULL if there are no more.

Description

Use this function in conjunction with `slapi_mods_get_first_mod()` to iterate through the `mods` in a `Slapi_Mods`. This will return an `LDAPMod` each time until the end.

Memory Concerns

Only one thread may be iterating through a particular `Slapi_Mods` at any given time.

See Also

`slapi_mods_get_first_mod()`

slapi_mods_get_next_smod()

Increments the `Slapi_Mods` iterator and returns the next `mod` wrapped in a `Slapi_Mods`.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mod *slapi_mods_get_next_smod(Slapi_Mods *smods,
    Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smods`	A pointer to a an initialized `Slapi_Mods`.
`smod`	Pointer to a `Slapi_Mod` that used to hold the `mod`.

Returns

This function returns a pointer to the `Slapi_Mod`, wrapping the next `mod`, or `NULL` if there are no more `mods`.

Description

Use this function in conjunction with `slapi_mods_get_first_smod()` to iterate through the `mods` in a `Slapi_Mods` using a `Slapi_Mods` wrapper.

Memory Concerns

Only one thread may be iterating through a particular Slapi_Mods at any given time.

See Also

slapi_mods_get_first_smod()

slapi_mods_get_num_mods()

Gets the number of mods in a Slapi_Mods structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_mods_get_num_mods(const Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.

Returns

The number of mods in the Slapi_Mods.

slapi_mods_init()

Initializes a Slapi_Mods.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_init(Slapi_Mods *smods, int initCount);
```

Parameters

This function takes the following parameters:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.

`initCount`	Number of modifications to allocate initially.

Description

Initializes a `Slapi_Mods` so that it is empty, but initially has room for the given number of `mods`.

Memory Concerns

If you are unsure of how much room you will need, you may use an `initCount` of `0`. The `Slapi_Mods` expands as necessary.

See Also

`slapi_mods_done()`

slapi_mods_init_byref()

Initializes a `Slapi_Mods` that is a wrapper for an existing array of `LDAPMod`.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_init_byref(Slapi_Mods *smods, LDAPMod **mods);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smods`	Pointer to an uninitialized `Slapi_Mods`.
`mods`	A null-terminated array of `LDAPMod`.

Description

Initializes a `Slapi_Mods` containing a reference to an array of `LDAPMod`. This function provides the convenience of using `Slapi_Mods` functions to access `LDAPMod` array items.

Memory Concerns

The array is not destroyed when the `Slapi_Mods` is destroyed. Notice that you cannot insert new `mods` in a `Slapi_Mods` that has been initialized by reference.

See Also

`slapi_mods_done()`

slapi_mods_init_passin()

Initializes a `Slapi_Mods` structure from an array of `LDAPMod`.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_init_passin(Slapi_Mods *smods, LDAPMod **mods);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smods`	Pointer to an uninitialized `Slapi_Mods`.
`mods`	A null-terminated array of `LDAPMod`.

Description

This function initializes a `Slapi_Mods` by passing in an array of `LDAPMod`. This function converts an array of `LDAPMod` to a `Slapi_Mods`.

Memory Concerns

The responsibility for the array and its elements is transferred to the `Slapi_Mods`. The array and its elements are destroyed when the `Slapi_Mods` is destroyed.

See Also

`slapi_mods_done()`

slapi_mods_insert_after()

Inserts an `LDAPMod` into a `Slapi_Mods` structure after the current iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_after(Slapi_Mods *smods, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smods`	Pointer to an initialized `Slapi_Mods` with a valid iterator position.
`mod`	Pointer to the `LDAPMod` to be inserted.

Description

This function inserts an `LDAPMod` in a `Slapi_Mods` immediately after the current position of the `Slapi_Mods` iterator. The iterator position is unchanged.

Memory Concerns

Responsibility for the `LDAPMod` is transferred to the `Slapi_Mods`.

This function must not be used on a `Slapi_Mods` initialized with `slapi_mods_init_byref()`.

See Also

`slapi_mods_get_first_mod()`

`sslapi_mods_get_next_mod()`

`slapi_mods_get_first_smod()`

`slapi_mods_get_next_smod()`

slapi_mods_insert_at()

Inserts an `LDAPMod` anywhere in a `Slapi_Mods`.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_at(Slapi_Mods *smods, LDAPMod *mod, int pos);
```

Parameters

This function takes the following parameters:

Parameter	Description
`smods`	Pointer to an initialized `Slapi_Mods`.

mod	Pointer to the LDAPMod to be inserted.
pos	Position at which to insert the new mod. Minimum value is 0. Maximum value is the current number of mods.

Description

This function inserts an LDAPMod at a given position in Slapi_Mods. Position 0 (zero) refers to the first mod. A position equal to the current number of mods causes an append. mods at and above the specified position are moved up by one, and the given position refers to the newly inserted mod.

Memory Concerns

Responsibility for the LDAPMod is transferred to the Slapi_Mods.

This function must not be used on a Slapi_Mods initialized with slapi_mods_init_byref().

slapi_mods_insert_before()

Inserts an LDAPMod into a Slapi_Mods structure before the current iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_before(Slapi_Mods *smods, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods with a valid iterator position.
mod	Pointer to the LDAPMod to be inserted.

Description

Inserts an LDAPMod into a Slapi_Mods immediately before the current position of the Slapi_Mods iterator. The iterator position is unchanged.

Memory Concerns

The responsibility for the LDAPMod is transferred to the Slapi_Mods.

This function must not be used on a Slapi_Mods initialized with slapi_mods_init_byref().

See Also

slapi_mods_get_first_mod()

slapi_mods_get_next_mod()

slapi_mods_insert_smod_at()

Inserts a Slapi_Mod anywhere in a Slapi_Mods.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_smod_at(Slapi_Mods *smods, Slapi_Mod *smod,
    int pos);
```

Parameters

This function takes the following parameters:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.
smod	Pointer to the Slapi_Mod to be inserted.
pos	Position at which to insert the Slapi_Mod. Minimum value is 0. Maximum value is the current number of mods.

Description

This function inserts a Slapi_Mod at a given position in Slapi_Mods. Position 0 (zero) refers to the first mod. A position equal to the current number of mods causes an append. mods at and above the specified position are moved up by one, and the given position refers to the newly inserted mod.

Memory Concerns

Responsibility for the Slapi_Mod is transferred to the Slapi_Mods.

This function must not be used on a Slapi_Mods initialized with slapi_mods_init_byref().

slapi_mods_insert_smod_before()

Inserts a Slapi_Mod into a Slapi_Mods structure before the current iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_smod_before(Slapi_Mods *smods,
    Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods with a valid iterator position.
smod	Pointer to the Slapi_Mod to be inserted.

Description

Inserts a Slapi_Mod into a Slapi_Mods immediately before the current position of the Slapi_Mods iterator. The iterator position is unchanged.

Memory Concerns

The responsibility for the Slapi_Mod is transferred to the Slapi_Mods.

This function must not be used on a Slapi_Mods initialized with slapi_mods_init_byref().

See Also

slapi_mods_get_first_mod()

slapi_mods_get_next_mod()

slapi_mods_iterator_backone()

Decrements the Slapi_Mods current iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_iterator_backone(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

Parameter	Description
`smods`	Pointer to an initialized `Slapi_Mods`.

Description

This function moves the iterator back one position.

See Also

`slapi_mods_get_first_mod()`

`sslapi_mods_get_next_mod()`

`slapi_mods_get_first_smod()`

`slapi_mods_get_next_smod()`

slapi_mods_new()

Allocates a new uninitialized `Slapi_Mods` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mods* slapi_mods_new( void );
```

Parameters

This function takes no parameters.

Returns

A pointer to an allocated uninitialized `Slapi_Mods`.

Description

This function allocates a new initialized `Slapi_Mods`.

Memory Concerns

Use this function when you need a `Slapi_Mods` allocated from the heap, rather than from the stack.

See Also

`slapi_mods_free()`

slapi_mods_remove()

Removes the mod at the current Slapi_Mods iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_remove(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

Parameter	Description
smods	Pointer to an initialized Slapi_Mods.

Description

This function removes the mod at the current iterator position.

See Also

slapi_mods_get_first_mod()

sslapi_mods_get_next_mod()

slapi_mods_get_first_smod()

slapi_mods_get_next_smod()

slapi_mr_filter_index()

Call a matching rule filter index function.

Syntax

```
#include "slapi-plugin.h"
int slapi_mr_filter_index(Slapi_Filter* f, Slapi_PBlock* pb);
```

Parameters

This function takes the following parameters:

Parameter	Description
f	Filter containing the matching rule OID
pb	Parameter block to pass to the filter index function

Description

This function enables plug-ins to call matching rule filter index functions.

Returns

This function returns `0` if successful. It returns `LDAP_UNAVAILABLE_CRITICAL_EXTENSION` if no filter index function is available for the rule in the filter. It returns `-1` if something goes horribly wrong setting the parameter block arguments.

slapi_mr_indexer_create()

Get a pointer to the indexer factory function for a matching rule.

Syntax

```
#include "slapi-plugin.h"
int slapi_mr_indexer_create(Slapi_PBlock * pb);
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block containing the matching rule object identifier affected to `SLAPI_PLUGIN_MR_OID`

Description

This function enables plug-ins to call matching rule indexer factory functions. If successful, the function sets `SLAPI_PLUGIN_MR_INDEXER_CREATE_FN` in `pb` to point to the indexer factory function.

Returns

This function returns `0` if successful. It returns `LDAP_UNAVAILABLE_CRITICAL_EXTENSION` if no indexer factory function is available for the matching rule. It returns `-1` if something goes horribly wrong getting or setting the parameter block arguments.

slapi_new_condvar()

Allocate a conditional variable.

Syntax

```
#include "slapi-plugin.h"
Slapi_CondVar *slapi_new_condvar( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameter:

Parameter	Description
`mutex`	Mutex used for the lock

Description

This function enables thread synchronization using a wait/notify mechanism.

Returns

This function returns a pointer to the new conditional variable if successful. Otherwise, it returns `NULL`.

Memory Considerations

Call `slapi_new_mutex()` to create the mutex and `slapi_destroy_mutex()` to free the mutex.

Call `slapi_destroy_condvar()` to free the conditional variable.

See Also

`slapi_destroy_condvar()`

`slapi_notify_condvar()`

`slapi_wait_condvar()`

slapi_new_mutex()

Allocate a mutex.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mutex *slapi_new_mutex( void );
```

Returns

This function returns a pointer to the new mutex if successful. Otherwise, it returns `NULL`.

Description

This function enables thread synchronization. Once a thread has locked the mutex using `slapi_lock_mutex()`, other threads attempting to acquire the lock are blocked until the thread holding the mutex calls `slapi_unlock_mutex()`.

Memory Considerations

Call `slapi_destroy_mutex()` to free the mutex.

See Also

`slapi_destroy_mutex()`

`slapi_lock_mutex()`

`slapi_unlock_mutex()`

slapi_notify_condvar()

Notify a change in a conditional variable.

Syntax

```
#include "slapi-plugin.h"
int slapi_notify_condvar( Slapi_CondVar *cvar, int notify_all );
```

Parameters

This function takes the following parameter:

Parameter	Description
`cvar`	Conditional variable changed
`notify_all`	`NULL` means notify only the next thread waiting for notification. Otherwise, all threads are notified.

Description

This function enables thread synchronization using a wait / notify mechanism.

Returns

This function returns `1` if successful. Otherwise, it returns `NULL`.

Memory Considerations

Call `slapi_wait_condvar()` to wait on a change to the conditional variable.

See Also

`slapi_destroy_condvar()`

`slapi_new_condvar()`

`slapi_wait_condvar()`

slapi_op_abandoned()

Determines whether or not the client has abandoned the current operation (the operation that passes in the parameter block).

Syntax

```
#include "slapi-plugin.h"
int slapi_op_abandoned( Slapi_PBlock *pb );
```

Parameters

This function takes the following parameter:

Parameter	Description
`pb`	Parameter block passed in from the current operation.

Description

This function allows you to verify if the operation associated to the pblock in the parameter has been abandoned. This function is useful to periodically check the operations status of long-running plug-ins.

Returns

This function returns one of the following values:

- 1 if the operation has been abandoned.
- 0 if the operation has not been abandoned.

slapi_op_get_type()

Gets the type of a `Slapi_Operation`.

Syntax

```
#include "slapi-plugin.h"
unsigned long slapi_op_get_type(Slapi_Operation * op);
```

Parameters

This function takes the following parameter:

Parameter	Description
op	The operation of which you wish to get the type.

Description

This function returns the type of an operation. The `Slapi_Operation` structure can be extracted from a pblock structure using `slapi_pblock_get()` with the `Slapi_Operation` parameter. For example:

```
slapi_pblock_get (pb, SLAPI_OPERATION, &op);
```

Returns

This function will return one of the following operation types:

- `SLAPI_OPERATION_BIND`
- `SLAPI_OPERATION_UNBIND`
- `SLAPI_OPERATION_SEARCH`
- `SLAPI_OPERATION_MODIFY`
- `SLAPI_OPERATION_ADD`
- `SLAPI_OPERATION_DELETE`
- `SLAPI_OPERATION_MODDN`
- `SLAPI_OPERATION_MODRDN`
- `SLAPI_OPERATION_COMPARE`
- `SLAPI_OPERATION_ABANDON`
- `SLAPI_OPERATION_EXTENDED`

See Also

`slapi_pblock_get()`

slapi_pblock_destroy()

Frees the specified parameter block from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_pblock_destroy( Slapi_PBlock *pb );
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block that you wish to free.

Memory Concerns

The parameter block that you wish to free must have been created using `slapi_pblock_new()`. Use of this function with pblocks allocated on the stack (e.g. `Slapi_PBlock pb;`), or using another memory allocator, is not supported and may lead to memory errors and memory leaks. For example:

```
Slapi_PBlock *pb = malloc(sizeof(Slapi_PBlock));
```

After calling this function, you should set the pblock pointer to `NULL` to avoid reusing freed memory in your function context, as in the following:

```
slapi_pblock_destroy(pb);
pb = NULL;
```

If you reuse the pointer in this way, it makes it easier to identify a Segmentation Fault, rather than using some difficult method to detect memory leaks or other abnormal behavior.

It is safe to call this function with a `NULL` pointer. For example:

```
Slapi_PBlock *pb = NULL;
slapi_pblock_destroy(pb);
```

This saves the trouble of checking for `NULL` before calling `slapi_pblock_destroy()`.

See Also

`slapi_pblock_new()`

slapi_pblock_get()

Gets the value of a name-value pair from a parameter block.

Syntax

```
#include "slapi-plugin.h"
int slapi_pblock_get( Slapi_PBlock *pb, int arg, void *value );
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block.
`arg`	ID of the name-value pair that you want to get. For a list of IDs that you can specify, see Chapter 3, “Parameter Block Reference."
`value`	Pointer to the value retrieved from the parameter block.

Returns

This function returns one of the following values:

- `0` if successful.
- `-1` if an error occurs (for example, if an invalid ID is specified).

Memory Concerns

The `void *value` argument should always be a pointer to the type of value you are retrieving:

```
int connid = 0;
...
retval = slapi_pblock_get(pb, SLAPI_CONN_ID, &connid);
```

`SLAPI_CONN_ID` is an integer value, so you will pass in a pointer to / address of an integer to get the value. Similarly, for a `char * value` (a string), pass in a pointer to / address of the value. For example:

```
char *binddn = NULL;
...
retval = slapi_pblock_get(pb, SLAPI_CONN_DN, &binddn);
```

With certain compilers on some platforms, you may have to cast the value to `(void *)`.

We recommend that you set the value to `0` or `NULL` before calling `slapi_pblock_get()` to avoid reading from uninitialized memory, in case the call to `slapi_pblock_get()` fails.

In most instances, the caller should not free the returned value. The value will usually be freed internally or through the call to `slapi_pblock_destroy()`. The exception is if the value is explicitly set by the caller through `slapi_pblock_set()`. In this case, the caller is responsible for memory management. If the value is freed, it is strongly recommended that the free is followed by a call to `slapi_pblock_set()` with a value of `NULL`. For example:

```
char *someparam = NULL;
...
someparam = slapi_ch_strdup(somestring);
slapi_pblock_set(pb, SOME_PARAM, someparam);
someparam = NULL; /* avoid dangling reference */
...
slapi_pblock_get(pb, SOME_PARAM, &someparam);
slapi_pblock_set(pb, SOME_PARAM, NULL); /* Make sure no one else
                                           references this. */
slapi_ch_free_string(&someparam);
...
```

Some internal functions may change the value passed in, so it is recommended to use `slapi_pblock_get()` to retrieve the value again, rather than relying on a potential dangling pointer. This is shown in the previous example, which sets `someparam` to `NULL` after setting it in the pblock.

See Also

`slapi_pblock_destroy()`

`slapi_pblock_set()`

slapi_pblock_new()

Creates a new parameter block.

Syntax

```
#include "slapi-plugin.h"
Slapi_PBlock *slapi_pblock_new();
```

Returns

Pointer to the new parameter block.

Memory Concerns

The pblock pointer allocated with `slapi_pblock_new()` must always be freed by `slapi_pblock_destroy()`. The use of other memory deallocators such as `free()` is not supported and may lead to crashes or memory leaks.

slapi_pblock_set()

Sets the value of a name-value pair in a parameter block.

Syntax

```
#include "slapi-plugin.h"
int slapi_pblock_set( Slapi_PBlock *pb, int arg, void *value );
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block.
`arg`	ID of the name-value pair that you want to set. For a list of IDs that you can specify, see Chapter 3, “Parameter Block Reference."
`value`	Pointer to the value that you want to set in the parameter block.

Returns

This function returns 0 if successful, or -1 if an error occurs (for example, if an invalid ID is specified).

Memory Concerns

The value to be passed in must always be a pointer, even for integer arguments. For example, if you wanted to do a search with the `ManageDSAIT` control:

```
int managedsait = 1;
...
slapi_pblock_set(pb, SLAPI_MANAGEDSAIT, &managedsait);
```

A call similar to the following example will cause a crash:

```
slapi_pblock_set(pb, SLAPI_MANAGEDSAIT, 1);
```

However, for values which are already pointers such as `char * strings`, `char **arrays`, `Slapi_Backend *`, and so forth, you can pass in the value directly. For example:

```
char *target_dn = slapi_ch_strdup(some_dn);
slapi_pblock_set(pb, SLAPI_TARGET_DN, target_dn);
```

or

```
slapi_pblock_set(pb, SLAPI_TARGET_DN, NULL);
```

With some compilers, you will have to cast the value argument to `(void *)`.

If the caller allocates the memory passed in, the caller is responsible for freeing that memory. Also, it is recommended to use `slapi_pblock_get()` to retrieve the value to free rather than relying on a potentially dangling pointer. See the `slapi_pblock_get()` example for more details.

When setting parameters to register a plug-in, the plug-in type must always be set first, since many of the plug-in parameters depend on the type. For example, set the `SLAPI_PLUGIN_TYPE` to extended operation before setting the list of extended operation OIDs for the plug-in.

See Also

`slapi_pblock_get()`

slapi_pw_find_sv()

Determines whether or not a specified password matches one of the encrypted values of an attribute. For example, you can call this function to determine if a given password matches a value in the `userpassword` attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_pw_find_sv( Slapi_Value **vals, const Slapi_Value *v );
```

Parameters

This function takes the following parameters:

Parameter	Description
`vals`	Pointer to the array of `Slapi_Value` structure pointers to hold the values of an attribute that stores passwords such as `userpassword`.
`v`	Pointer to the `Slapi_Value` structure containing the password that you wish to check (for example, you can get this value from the SLAPI_BIND_CREDENTIALS parameter in the parameter block and create the `Slapi_Value` using `slapi_value_init_berval()`).

Returns

This function returns `0` if the password specified by `v` was found in `vals`, or a non-zero value if the password `v` was not found in `vals`.

Memory Concerns

You must allocate and release an array of `Slapi_Value` structures for `vals` sized to hold the exact number of password values for the `userpassword` attribute on the entry to check. The resulting array is not `NULL` terminated. For a simpler memory allocation model, use `slapi_pw_find_valueset()` instead.

Description

When the server stores the password for an entry in the `userpassword` attribute, it encrypts the password using different schemes.

Use this function to determine if a given password is one of the values of the `userpassword` attribute. This function determines which password scheme was used to store the password and uses the appropriate comparison function to compare a given value against the encrypted values of the `userpassword` attribute.

See Also

`slapi_pw_find_valueset()`

slapi_pw_find_valueset()

Determines whether or not a specified password matches one of the encrypted values of an attribute. For example, you can call this function to determine if a given password matches a value in the `userpassword` attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_pw_find_valueset(Slapi_Valueset *valset,
    const Slapi_Value *v);
```

Parameters

This function takes the following parameters:

Parameter	Description
`valset`	Pointer to the `Slapi_Valueset` structure containing the values of an attribute that stores passwords such as `userpassword`.
`v`	Pointer to the `Slapi_Value` structure containing the password to check (for example, you can get this value from the SLAPI_BIND_CREDENTIALS parameter in the parameter block and create the `Slapi_Value` using `slapi_value_init_berval()`).

Returns

This function returns `0` if the password specified by `v` was found in `valset`, or a non-zero value if the password `v` was not found in `valset`.

Description

When the server stores the password for an entry in the `userpassword` attribute, it encrypts the password using different schemes.

Use this function to determine if a given password is one of the values of the `userpassword` attribute. This function determines which password scheme was used to store the password and uses the appropriate comparison function to compare a given value against the encrypted values of the `userpassword` attribute.

See Also

`slapi_pw_find_sv()`

slapi_rdn_add()

Adds a new RDN to an existing `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_add(Slapi_RDN *rdn, const char *type,
    const char *value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The target `Slapi_RDN` structure.
`type`	The type (`cn`, `o`, `ou`, etc.) of the RDN to be added. This parameter cannot be `NULL`.
`value`	The value of the RDN to be added. This parameter cannot be `NULL`.

Returns

This function always returns 1.

Description

This function adds a new type/value pair to an existing RDN, or sets the type/value pair as the new RDN if `rdn` is empty. This function resets the `FLAG_RDNS` flags, which means that the RDN array within the `Slapi_RDN` structure is no longer current with the new RDN.

See Also

`slapi_rdn_get_num_components()`

slapi_rdn_compare()

Compares two RDNs.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_compare(Slapi_RDN *rdn1, Slapi_RDN *rdn2);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn1`	The first RDN to compare.
`rdn2`	The second RDN to compare.

Returns

This function returns `0` if `rdn1` and `rdn2` have the same RDN components, or `-1` if they do not.

Description

This function compares `rdn1` and `rdn2`. For `rdn1` and `rdn2` to be considered equal RDNs, their components do not necessarily have to be in the same order.

slapi_rdn_contains()

Checks whether a `Slapi_RDN` structure holds any RDN matching a given type/value pair.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_contains(Slapi_RDN *rdn, const char *type,
    const char *value,size_t length);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The `Slapi_RDN` structure containing the RDN value(s).
`type`	The type (`cn`, `o`, `ou`, etc.) of the RDN searched.
`value`	The value of the RDN searched.
`length`	Gives the length of `value` that should be taken into account for the string operation when searching for the RDN.

Returns

This function returns `1` if `rdn` contains an RDN that matches the `type`, `value` and `length`, or `0` if no RDN matches the desired type/value.

Description

This function searches for an RDN inside of the `Slapi_RDN` structure `rdn` that matches both `type` and `value` as given in the parameters. This function makes a call to `slapi_rdn_get_index()` and verifies that the returned value is anything but `-1`.

See Also

`slapi_rdn_get_index()`

`slapi_rdn_contains_attr()`

slapi_rdn_contains_attr()

Checks whether a `Slapi_RDN` structure contains any RDN matching a given type, and if true, gets the corresponding attribute value.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_contains_attr(Slapi_RDN *rdn, const char *type,
    char **value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The `Slapi_RDN` structure containing the RDN value(s).
`type`	Type (`cn`, `o`, `ou`, etc.) of the RDN searched.
`value`	Repository that will hold the value of the first RDN whose type matches the content of the parameter `type`. If this parameter is `NULL` at the return of the function, no RDN with the desired type exists within `rdn`.

Returns

This function returns 1 if `rdn` contains a RDN that matches the given type, or 0 if there is no match.

Description

This function looks for an RDN inside the `Slapi_RDN` structure `rdn` that matches the type given in the parameters. This function makes a call to `slapi_rdn_get_index_attr()` and verifies that the returned value is anything but `-1`. If successful, it also returns the corresponding attribute value.

See Also

`slapi_rdn_get_index_attr()`

`slapi_rdn_contains()`

slapi_rdn_done()

Clears an instance of a `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_done(Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
`rdn`	Pointer to the structure to be cleared.

Description

This function clears the contents of a `Slapi_RDN` structure. It frees both the RDN value and the array of split RDNs. Those pointers are then set to `NULL`.

See Also

`slapi_rdn_free()`

`slapi_rdn_init()`

slapi_rdn_free()

Frees a `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_free(Slapi_RDN **rdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
`rdn`	Pointer to the pointer of the `Slapi_RDN` structure to be freed.

Description

This function frees both the contents of the `Slapi_RDN` structure and the structure itself pointed to by the content of `rdn`.

See Also

`slapi_rdn_new()`

`slapi_rdn_done()`

slapi_rdn_get_first()

Gets the type / value pair corresponding to the first RDN stored in a `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_first(Slapi_RDN *rdn, char **type, char **value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The `Slapi_RDN` structure containing the RDN value(s).
`type`	Repository that will hold the type of the first RDN. If this parameter is NULL at the return of the function, it means `rdn` is empty.
`value`	Repository that will hold the type of the first RDN. If this parameter is NULL at the return of the function, it means `rdn` is empty.

Returns

This function returns -1 if `rdn` is empty, or 1 if the operation is successful.

Description

This function gets the type / value pair corresponding to the first RDN stored in `rdn`. For example, if the RDN is `cn=Joey`, the function will place `cn` in the `type` return parameter, and `Joey` in `value`.

See Also

`slapi_rdn_get_next()`

`slapi_rdn_get_rdn()`

slapi_rdn_get_index()

Gets the index of the RDN that follows the RDN with a given type and value.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_index(Slapi_RDN *rdn, const char *type,
    const char *value, size_t length);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The `Slapi_RDN` structure containing the RDN value(s).
`type`	Type (`cn`, `o`, `ou`, etc.) of the RDN that is searched.

value	Value of the RDN searched.
length	Gives the length of value that should be taken into account for the string comparisons when searching for the RDN.

Returns

This function returns the index of the RDN that follows the RDN matching the contents of the parameters type and value. If no RDN stored in rdn matches the given type/value pair, -1 is returned.

Description

This function searches for an RDN inside the Slapi_RDN structure rdn that matches both type and value as given in the parameters. If it succeeds, the position of the matching RDN is returned.

See Also

slapi_rdn_get_index_attr()

slapi_rdn_contains()

slapi_rdn_get_index_attr()

Gets the position and the attribute value of the first RDN in a Slapi_RDN structure that matches a given type.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_index_attr(Slapi_RDN *rdn, const char *type,
    char **value);
```

Parameters

This function takes the following parameters:

Parameter	Description
rdn	The Slapi_RDN structure containing the RDN value(s).
type	Type (cn, o, ou, etc.) of the RDN searched.
value	Repository that will hold the value of the first RDN whose type matches the content of the parameter type. If this parameter is NULL at the return of the function, no RDN exists within rdn.

Returns

This function returns `-1` if there is no RDN that matches the content type, or the real position of the first RDN within RDN that matches the content of type.

Description

This function searches for an RDN inside of the `Slapi_RDN` structure `rdn` that matches the type given in the parameters. If successful, the position of the matching RDN, as well as the corresponding attribute value, is returned.

See Also

`slapi_rdn_get_index()`

slapi_rdn_get_next()

Gets a certain RDN type/value pair from within the RDNs stored in a `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_next(Slapi_RDN *rdn, int index, char **type,
    char **value);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The `Slapi_RDN` structure containing the RDN value(s).
`index`	Indicates the position of the RDN that precedes the currently desired RDN.
`type`	Repository that will hold the type (`cn`, `o`, `ou`, etc.) of the next (`index`+1) RDN. If this parameter is `NULL` at the return of the function, the RDN does not exist.
`value`	Repository that will hold the value of the next (`index`+1) RDN. If this parameter is `NULL`, the RDN does not exist.

Returns

This function returns `-1` if there is no RDN in the index position, or the real position of the retrieved RDN if the operation was successful.

Description

This function gets the type / value pair corresponding to the RDN stored in the next (index+1) position inside rdn. Notice that the index of an element within an array of values is always one unit below its real position in the array.

See Also

slapi_rdn_get_first()

slapi_rdn_get_rdn()

slapi_rdn_get_num_components()

Gets the number of RDN type / value pairs present in a Slapi_RDN structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_num_components(Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
rdn	The target Slapi_RDN structure.

Returns

This function returns the number of RDN type / value pairs present in rdn.

See Also

slapi_rdn_add()

slapi_rdn_get_rdn()

Gets the RDN from a Slapi_RDN structure.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_rdn_get_rdn(const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
`rdn`	The `Slapi_RDN` structure holding the RDN value.

Returns

This function returns the RDN value.

slapi_rdn_init()

Initializes a `Slapi_RDN` structure with `NULL` values.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init(Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
`rdn`	The `Slapi_RDN` structure to be initialized.

Description

This function initializes a given `Slapi_RDN` structure with `NULL` values (both the RDN value and the array of split RDNs are set to `NULL`).

See Also

`slapi_rdn_new()`

`slapi_rdn_free()`

`slapi_rdn_done()`

slapi_rdn_init_dn()

Initializes a `Slapi_RDN` structure with an RDN value taken from a given DN.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init_dn(Slapi_RDN *rdn,const char *dn);
```

Parameters

This function takes the following parameters:

Parameter	Description
rdn	The Slapi_RDN structure to be initialized.
dn	The DN value whose RDN will be used to initialize the new Slapi_RDN structure.

Description

This function initializes a given Slapi_RDN structure with the RDN value taken from the DN passed in the dn parameter.

See Also

slapi_rdn_init_sdn()

slapi_rdn_init_rdn()

slapi_rdn_init_rdn()

Initializes a Slapi_RDN structure with an RDN value.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init_rdn(Slapi_RDN *rdn,const Slapi_RDN *fromrdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
rdn	The Slapi_RDN structure to be initialized.
fromrdn	The RDN value to be set in the new Slapi_RDN structure.

Description

This function initializes a given `Slapi_RDN` structure with the RDN value in `fromrdn`.

See Also

`slapi_rdn_init_dn()`

`slapi_rdn_init_sdn()`

slapi_rdn_init_sdn()

Initializes a `Slapi_RDN` structure with an RDN value taken from the DN contained in a given `Slapi_DN` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init_sdn(Slapi_RDN *rdn,const Slapi_DN *sdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The `Slapi_RDN` structure to be initialized.
`sdn`	The `Slapi_DN` structure containing the DN value whose RDN will be used to initialize the new `Slapi_RDN` structure.

Description

This function initializes a given `Slapi_RDN` structure with the RDN value taken from the DN passed within the `Slapi_DN` structure of the `sdn` parameter.

See Also

`slapi_rdn_init_dn()`

`slapi_rdn_init_rdn()`

slapi_rdn_isempty()

Checks whether an RDN value is stored in a `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_isempty(const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
`rdn`	The target `Slapi_RDN` structure.

Returns

This function returns `1` if there is no RDN value present, or `0` if `rdn` contains a value.

See Also

`slapi_rdn_init()`

`slapi_rdn_done()`

`slapi_rdn_free()`

slapi_rdn_new()

Allocates a new `Slapi_RDN` structure and initializes the values to `NULL`.

Syntax

```
#include "slapi-plugin.h"
Slapi_RDN * slapi_rdn_new();
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to the newly allocated, and still empty, `Slapi_RDN` structure.

Description

This function creates a new `Slapi_RDN` structure by allocating the necessary memory and initializing both the RDN value and the array of split RDNs to `NULL`.

See Also

`slapi_rdn_init()`

`slapi_rdn_done()`

`slapi_rdn_free()`

slapi_rdn_new_dn()

Creates a new `Slapi_RDN` structure and sets an RDN value taken from a given DN.

Syntax

```
#include "slapi-plugin.h"
Slapi_RDN *slapi_rdn_new_dn(const char *dn);
```

Parameters

This function takes the following parameter:

Parameter	Description
`dn`	The DN value whose RDN will be used to initialize the new `Slapi_RDN` structure.

Returns

This function returns a pointer to the new `Slapi_RDN` structure initialized with the TDN taken from the DN value in `dn`.

Description

This function creates a new `Slapi_RDN` structure and initializes its RDN with the value taken from the DN passed in the `dn` parameter.

Memory Concerns

The memory is allocated by the function itself.

See Also

`slapi_rdn_new_sdn()`

`slapi_rdn_new_rdn()`

slapi_rdn_new_rdn()

Creates a new Slapi_RDN structure and sets an RDN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_RDN * slapi_rdn_new_rdn(const Slapi_RDN *fromrdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
fromrdn	The RDN value to be set in the new Slapi_RDN structure.

Returns

This function returns a pointer to the new Slapi_RDN structure with an RDN set to the content of fromrdn.

Description

This function creates a new Slapi_RDN structure and initializes its RDN with the value of fromrdn.

Memory Concerns

The memory is allocated by the function itself.

See Also

slapi_rdn_new_dn()

slapi_rdn_new_sdn()

slapi_rdn_new_sdn()

Creates a new Slapi_RDN structure and sets an RDN value taken from the DN contained in a given Slapi_RDN structure.

Syntax

```
#include "slapi-plugin.h"
vSlapi_RDN *slapi_rdn_new_sdn(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
sdn	Slapi_RDN structure containing the DN value whose RDN will be used to initialize the new Slapi_RDN structure.

Returns

This function returns a pointer to the new Slapi_RDN structure initialized with the RDN taken from the DN value in dn.

Description

This function creates a new Slapi_RDN structure and initializes its RDN with the value taken from the DN passed within the Slapi_RDN structure of the sdn parameter.

Memory Concerns

The memory is allocated by the function itself.

See Also

slapi_rdn_new_dn()

slapi_rdn_new_rdn()

slapi_rdn_remove()

Removes an RDN type/value pair from a Slapi_RDN structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_remove(Slapi_RDN *rdn, const char *type,
const char *value, size_t length);
```

Parameters

This function takes the following parameters:

Parameter	Description
rdn	The target Slapi_RDN structure.
type	Type (cn, o, ou, etc.) of the RDN searched.
value	The value of the RDN searched.

`length`	Gives the length of `value` that should be taken into account for the string comparisons when searching for the RDN.

Returns

This function returns `1` if the RND is removed from `rdn`, or `0` if no RDN is removed.

Description

This function removes the RDN from `rdn` that matches the given criteria (`type`, `value` and `length`).

See Also

`slapi_rdn_remove_index()`

`slapi_rdn_remove_attr()`

slapi_rdn_remove_attr()

Removes an RDN type/value pair from a `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_remove_attr(Slapi_RDN *rdn, const char *type);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The target `Slapi_RDN` structure.
`type`	Type (`cn`, `o`, `ou`, etc.) of the RDN searched.

Returns

This function returns `1` of the RDN is removed from `rdn`, or `0` if no RDN is removed.

Description

This function removes the first RDN from `rdn` that matches the given type.

See Also

`slapi_rdn_remove()`

`slapi_rdn_remove_index()`

slapi_rdn_remove_index()

Removes an RDN type/value pair from a `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_remove_index(Slapi_RDN *rdn, int atindex);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The target `Slapi_RDN` structure.
`atindex`	The index of the RDN type/value pair to remove.

Returns

This function returns `1` if the RDN is removed from `rdn`, or `0` if no RDN is removed because either `rdn` is empty, or the index goes beyond the number of RDNs present.

Description

This function removes the RDN from `rdn` with `atindex` index (placed in the `atindex`+1 position).

See Also

`slapi_rdn_remove()`

`slapi_rdn_remove_attr()`

slapi_rdn_set_dn()

Sets an RDN value in a `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_set_dn(Slapi_RDN *rdn,const char *dn);
```

Parameters

This function takes the following parameters:

Parameter	Description
rdn	The target Slapi_RDN structure.
dn	The DN value whose RDN will be set in rdn.

Description

This function sets an RDN value in a Slapi_RDN structure. The structure is freed from memory and freed of any previous content before setting the new RDN. The new RDN is taken from the DN value present in the dn parameter.

See Also

slapi_rdn_set_sdn()

slapi_rdn_set_rdn()

slapi_rdn_set_rdn()

Sets an RDN in a Slapi_RDN structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_set_rdn(Slapi_RDN *rdn,const Slapi_RDN *fromrdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
rdn	The target Slapi_RDN structure.
fromrdn	The Slapi_RDN structure from which to take the RDN.

Description

This function sets an RDN value in a `Slapi_RDN` structure. The structure is freed from memory and freed of any previous content before setting the new RDN.

See Also

`slapi_rdn_set_dn()`

`slapi_rdn_set_sdn()`

slapi_rdn_set_sdn()

Sets an RDN value in a `Slapi_RDN` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_set_sdn(Slapi_RDN *rdn,const Slapi_DN *sdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`rdn`	The target `Slapi_RDN` structure.
`sdn`	The `Slapi_DN` structure from which to take the RDN.

Description

This function sets an RDN value in a `Slapi_RDN` structure. The structure is freed from memory and freed of any previous content before setting the new RDN. The new RDN is taken from the DN value present inside of a `Slapi_DN` structure.

See Also

`slapi_rdn_set_dn()`

`slapi_rdn_set_rdn()`

slapi_register_object_extension()

Register an object extension.

Syntax

```
#include "slapi-plugin.h"
int slapi_register_object_extension(
    const char *pluginname,
    const char *objectname,
    slapi_extension_constructor_fnptr constructor,
    slapi_extension_destructor_fnptr destructor,
    int *objecttype,
    int *extensionhandle);
```

Parameters

This function takes the following parameters:

Parameter	Description
`pluginname`	String identifying the plug-in
`objectname`	Name of the object to extend such as `SLAPI_EXT_CONNECTION` to add private data to a connection or `SLAPI_EXT_OPERATION` to add private data to an operation
`constructor`	Constructor to allocate memory for the object extension and create the extension
`destructor`	Destructor to free memory used for the object extension
`objecttype`	Set by the server and used to retrieve the extension
`extensionhandle`	Set by the server and used to retrieve the extension

Description

This function registers an extension to an object such as a connection or an operation. This mechanism enables a plug-in to store private data with an operation that is passed from a preoperation to a postoperation plug-in for example, something not possible using parameter blocks.

Register object extensions as part of the plug-in initialization function.

Returns

This function returns `0` if successful. Otherwise, it returns `-1`.

See Also

`slapi_extension_constructor_fnptr`

`slapi_extension_destructor_fnptr`

`slapi_get_object_extension()`

`slapi_set_object_extension()`

slapi_register_plugin()

Register another plug-in.

Syntax

```
#include "slapi-plugin.h"
int slapi_register_plugin( const char *plugintype, int enabled,
    const char *initsymbol, slapi_plugin_init_fnptr initfunc,
    const char *name, char **argv, void *group_identity);
```

Parameters

This function takes the following parameters:

Parameter	Description
`plugintype`	String identifying the plug-in type as described under “Plug-In Registration,” on page 343
`enabled`	`1` to enable the plug-in on registration, `0` otherwise
`initsymbol`	String representation of the plug-in initialization function `initfunc` such as `"my_init_fcn"`
`initfunc`	Pointer to the plug-in initialization function
`name`	Common Name for the plug-in
`argv`	Arguments passed to the plug-in
`group_identity`	Plug-in group identifier, typically obtained from `SLAPI_PLUGIN_IDENTITY` for the plug-in calling this function

Description

This function registers another plug-in. Register plug-ins as part of the plug-in initialization function.

Returns

This function returns `0` if successful. Otherwise, it returns `-1`.

See Also

`slapi_plugin_init_fnptr`

slapi_register_role_get_scope()

Register a callback to determine the scope of a role.

Syntax

```
#include "slapi-plugin.h"
void slapi_register_role_get_scope(
    roles_get_scope_fn_type get_scope_fn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`get_scope_fn`	Callback to determine the scope of a role

Description

This function registers the callback that evaluates the scope of a role. Register the callback as part of the plug-in initialization function.

See Also

`roles_get_scope_fn_type`

`slapi_role_get_scope()`

slapi_register_supported_control()

Registers the specified control with the server. This function associates the control with an object identification (OID). When the server receives a request that specifies this OID, the server makes use of this information to determine if the control is supported by the server or its plug-ins.

Syntax

```
#include "slapi-plugin.h"
void slapi_register_supported_control( char *controloid,
    unsigned long controlops );
```

Parameters

This function takes the following parameters:

Parameter	Description

`controloid`	OID of the control you want to register.
`controlops`	Operation that the control is applicable to.

The `controlops` argument can have one or more of the following values:

ID	Description
`SLAPI_OPERATION_BIND`	The specified control applies to the LDAP bind operation.
`SLAPI_OPERATION_UNBIND`	The specified control applies to the LDAP unbind operation.
`SLAPI_OPERATION_SEARCH`	The specified control applies to the LDAP search operation.
`SLAPI_OPERATION_MODIFY`	The specified control applies to the LDAP modify operation.
`SLAPI_OPERATION_ADD`	The specified control applies to the LDAP add operation.
`SLAPI_OPERATION_DELETE`	The specified control applies to the LDAP delete operation.
`SLAPI_OPERATION_MODDN`	The specified control applies to the LDAP modify DN operation.
`SLAPI_OPERATION_MODRDN`	The specified control applies to the LDAPv3 modify RDN operation.
`SLAPI_OPERATION_COMPARE`	The specified control applies to the LDAP compare operation.
`SLAPI_OPERATION_ABANDON`	The specified control applies to the LDAP abandon operation.
`SLAPI_OPERATION_EXTENDED`	The specified control applies to the LDAPv3 extended operation.
`SLAPI_OPERATION_ANY`	The specified control applies to any LDAP operation.
`SLAPI_OPERATION_NONE`	The specified control applies to none of the LDAP operations.

You can specify a combination of values by bitwise ORing the values together (for example, `SLAPI_OPERATION_ADD | SLAPI_OPERATION_DELETE`).

See Also

`slapi_get_supported_controls_copy()`

`slapi_control_present()`

slapi_register_supported_saslmechanism()

Registers the specified Simple Authentication and Security Layer (SASL) mechanism with the server.

Syntax

```
#include "slapi-plugin.h"
void slapi_register_supported_saslmechanism( char *mechanism );
```

Parameters

This function takes the following parameter:

Parameter	Description
`mechanism`	String identifying the SASL mechanism to both the server and to clients requesting the mechanism during a SASL bind

Description

Use this function in the plug-in initialization function to register the name of a SASL mechanism supported by the plug-in. The preoperation function handling the SASL bind can then check the SASL bind mechanism name provided by the client to determine whether to attempt to handle the bind.

See Also

The sample *ServerRoot*`/plugins/slapd/slapi/examples/testsaslbind.c` plug-in demonstrates the use of this function.

slapi_rename_internal_set_pb()

Prepares a parameter block for an internal modify RDN operation.

Syntax

```
#include "slapi-plugin.h"
int slapi_rename_internal_set_pb(Slapi_PBlock *pb,
    const char *olddn, const char *newrdn,
    const char *newsuperior, int deloldrdn,
    LDAPControl **controls, const char *uniqueid,
    Slapi_ComponentId *plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block for the internal modify RDN operation
`olddn`	Distinguished Name of the entry to rename
`newrdn`	New Relative Distinguished name to apply
`newsuperior`	DN of parent after renaming
`deloldrdn`	`1` to delete the old RDN, *0* to retain the old RDN
`controls`	Array of controls to request for the modify RDN operation
`uniqueid`	Unique identifier for the entry if using this rather than DN
`plugin_identity`	Plug-in identifier obtained from `SLAPI_PLUGIN_IDENTITY` during plug-in initialization
`operation_flags`	Flags such as `SLAPI_OP_FLAG_NEVER_CHAIN`

Returns

This function returns `0` if successful. Otherwise, it returns an LDAP error code.

Description

This function prepares a parameter block for use with `slapi_modrdn_internal_pb()`.

Memory Concerns

Allocate the parameter block using `slapi_pblock_new()` before calling this function.

Free the parameter block after calling `slapi_modrdn_internal_pb()`.

See Also

`slapi_modrdn_internal_pb()`

`slapi_pblock_new()`

slapi_role_check()

Checks if the entry pointed to by `entry_to_check` contains the role indicated by `role_dn`.

Syntax

```
#include "slapi-plugin.h"
int slapi_role_check(Slapi_Entry *entry_to_check,
    Slapi_DN *role_dn,int *present);
```

Parameters

This function takes the following parameters:

Parameter	Description
`entry_to_check`	The entry in which the presence of a role is to be checked.
`role_dn`	The DN of the role for which to check.
`present`	Pointer to an integer where the result will be placed; present or not present.

Returns

This function returns one of the following values:

- `0` for success if `role_dn` is present in `entry_to_check` and `present` is set to non-zero. Otherwise it is `0`.
- non-zero (error condition) if the presence of the role is undetermined.

slapi_role_get_scope()

Determine the scope of a role.

Syntax

```
#include "slapi-plugin.h"
int slapi_role_get_scope(Slapi_Entry *role_entry,
    Slapi_DN ***scope_dn, int *nb_scope);
```

Parameters

This function takes the following parameters:

Parameter	Description
`role_entry`	Entry defining the role
`scope_dn`	Set by the callback to the Distinguished Name of the entry at the base of the scope
`nb_scope`	Set by the callback to a value such as `LDAP_SCOPE_BASE`, `LDAP_SCOPE_ONELEVEL`, or `LDAP_SCOPE_SUBTREE`

Description

This function triggers a callback to evaluate the scope of a role.

See Also

`roles_get_scope_fn_type`

`slapi_register_role_get_scope()`

slapi_sdn_add_rdn()

Adds the RDN contained in a `Slapi_RDN` structure to the DN contained in a `Slapi_DN` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_add_rdn(Slapi_DN *sdn, const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`sdn`	`Slapi_DN` structure containing the value to which a new RDN is to be added.
`rdn`	`Slapi_RDN` structure containing the RDN value that is to be added to the DN value.

Returns

This function returns the `Slapi_DN` structure with the new DN formed by adding the RDN value in `rdn` to the DN value in `dn`.

See Also

`slapi_sdn_set_rdn()`

slapi_sdn_compare()

Compares two DNs.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_compare( const Slapi_DN *sdn1, const Slapi_DN *sdn2 );
```

Parameters

This function takes the following parameters:

Parameter	Description
`sdn1`	DN to compare with the value in `sdn2`.
`sdn2`	DN to compare with the value in `sdn1`.

Returns

This function returns one of the following values:

- 0 if `sdn1` is equal to `sdn2`.
- -1 if `sdn1` is `NULL`.
- 1 if `sdn2` is `NULL` and `sdn1` is not `NULL`.

Description

This function compares two DNs, `sdn1` and `sdn2`. The comparison is case sensitive.

slapi_sdn_copy()

Makes a copy of a DN.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_copy(const Slapi_DN *from, Slapi_DN *to);
```

Parameters

This function takes the following parameters:

Parameter	Description
`from`	The original DN.
`to`	Destination of the copied DN, containing the copy of the DN in `from`.

Description

This function copies the DN in `from` to the structure pointed by `to`.

Memory Concerns

`to` must be allocated in advance of calling this funciton.

See Also

slapi_sdn_dup()

slapi_sdn_done()

Clears an instance of a `Slapi_DN` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_done(Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
`sdn`	Pointer to the structure to clear.

Description

This function clears the contents of a Slapi_DN structure. It frees both the DN and the normalized DN, if any, and sets those pointers to NULL.

See Also

slapi_sdn_free()

slapi_sdn_dup()

Duplicates a Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN * slapi_sdn_dup(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
sdn	Pointer to the Slapi_DN structure to duplicate.

Returns

This function returns a pointer to a duplicate of sdn.

See Also

slapi_sdn_copy()

slapi_sdn_new()

slapi_sdn_free()

Frees a Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_free(Slapi_DN **sdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
sdn	Pointer to the Slapi_DN structure to free.

Description

This function frees the Slapi_DN structure and its contents pointed to by the contents of sdn.

See Also

slapi_sdn_new()

slapi_sdn_done()

slapi_sdn_get_backend_parent()

Gets the DN of the parent of an entry within a specific backend.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_get_backend_parent(const Slapi_DN *sdn,
    Slapi_DN *sdn_parent,const Slapi_Backend *backend);
```

Parameters

This function takes the following parameters:

Parameter	Description
sdn	DN of the entry whose parent is searched.
sdn_parent	Parent DN of sdn.
backend	Backend of which the parent of sdn is to be searched.

Description

This function gets the parent DN of an entry within a given backend. The parent DN is returned is sdn_parent, unless sdn is empty or is a suffix of the backend itself. In this case, sdn_parent is empty.

Memory Concerns

A `Slapi_DN` structure for `sdn_parent` must be allocated before calling this function.

See Also

`slapi_sdn_get_parent()`

slapi_sdn_get_dn()

Gets the DN from a `Slapi_DN` structure.

Syntax

```
#include "slapi-plugin.h"
const char * slapi_sdn_get_dn(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
`sdn`	The `Slapi_DN` structure containing the DN value.

Returns

This function returns the DN value.

Description

This function retrieves the DN value of a `Slapi_DN` structure. The returned value can be the normalized DN (in a canonical format and in lower case) if no other value is present.

See Also

`slapi_sdn_get_rdn()`

`slapi_sdn_get_parent()`

`slapi_sdn_get_ndn()`

slapi_sdn_get_ndn()

Gets the normalized DN of a `Slapi_DN` structure.

Syntax

```
#include "slapi-plugin.h"
const char * slapi_sdn_get_ndn(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
sdn	The Slapi_DN structure containing the DN value.

Returns

This function returns the normalized DN value.

Description

This function retrieves the normalized DN (in a canonical format and lower case) from a Slapi_DN structure and normalizes sdn if it has not already been normalized.

See Also

slapi_sdn_get_dn()

slapi_sdn_get_ndn_len()

Gets the length of the normalized DN of a Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_get_ndn_len(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
sdn	The Slapi_DN structure containing the DN value.

Returns

This function returns the length of the normalized DN.

Description

This function contains the length of the normalized DN and normalizes sdn if it has not already been normalized.

slapi_sdn_get_parent()

Gets the parent DN of a given Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_get_parent(const Slapi_DN *sdn,Slapi_DN *sdn_parent);
```

Parameters

This function takes the following parameters:

Parameter	Description
sdn	Pointer to the initialized Slapi_DN structure containing the DN whose parent is searched.
sdn_parent	Pointer to the Slapi_DN structure where the parent DN is returned.

Description

This function returns a Slapi_DN structure containing the parent DN of the DN kept in the structure pointed to by sdn.

See Also

slapi_sdn_get_backend_parent()

slapi_sdn_get_rdn()

Gets the RDN from a DN.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_get_rdn(const Slapi_DN *sdn, Slapi_RDN *rdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
sdn	Pointer to the Slapi_DN structure containing the DN.
rdn	Pointer to the Slapi_RDN structure where the RDN is returned.

Description

This function takes the DN stored in the Slapi_DN structure pointed to by sdn and retrieves its returned RDN within the Slapi_RDN structure pointed to by rdn.

See Also

slapi_sdn_get_dn()

slapi_sdn_add_rdn()

slapi_sdn_isempty()

Checks whether there is a DN value stored in a Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_isempty( const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

Parameter	Description
sdn	Pointer to the Slapi_DN structure that is going to be checked.

Returns

This function returns one of the following values:

- 1 if there is no DN value (normalized or not) present in the Slapi_DN structure.
- 0 if sdn is not empty.

Description

This function checks whether a Slapi_DN structure contains a normalized or non-normalized value.

See Also

`slapi_sdn_done()`

slapi_sdn_isgrandparent()

Checks whether a DN is the parent of the parent of a given DN.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_isgrandparent( const Slapi_DN *parent,
    const Slapi_DN *child );
```

Parameters

This function takes the following parameters:

Parameter	Description
`parent`	Pointer to the `Slapi_DN` structure containing the DN that claims to be the grandparent DN of the DN in `child`.
`child`	Pointer to the `Slapi_DN` structure containing the DN of the supposed "grandchild" of the DN in the structure pointed to by `parent`.

Returns

This function returns one of the following values:

- `1` if the DN in `parent` is the grandparent of the DN in `child`.
- `0` if the DN in `parent` does not match the DN of the grandparent of the DN in `child`.

See Also

`slapi_sdn_isparent()`

`slapi_sdn_issuffix()`

`slapi_sdn_get_parent()`

slapi_sdn_isparent()

Checks whether a DN is the parent of a given DN.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_isparent(const Slapi_DN *parent,
    const Slapi_DN *child);
```

Parameters

This function takes the following parameters:

Parameter	Description
`parent`	Pointer to the `Slapi_DN` structure containing the DN that claims to be the parent of the DN in `child`.
`child`	Pointer to the `Slapi_DN` structure containing the DN of the supposed child of the DN in the structure pointed to by `parent`.

Returns

This function returns one of the following values:

- 1 if the DN in `parent` is the parent of the DN in `child`.
- 0 if the DN in `parent` does not match the DN of the parent of the DN in `child`.

See Also

`slapi_sdn_issuffix()`

`slapi_sdn_get_parent()`

slapi_sdn_issuffix()

Checks whether a `Slapi_DN` structure contains a suffix of another `Slapi_DN` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_issuffix(const Slapi_DN *sdn,
    const Slapi_DN *suffixsdn);
```

Parameters

This function takes the following parameters:

Parameter	Description

sdn	Pointer to the Slapi_DN structure to be checked.
suffixsdn	Pointer to the Slapi_DN structure of the suffix.

Returns

This function returns one of the following values:

- 1 if the DN is suffixsdn is the suffix of sdn.
- 0 if the DN in suffixsdn is not a suffix of sdn.

See Also

slapi_sdn_isparent()

slapi_sdn_new()

Allocates a new Slapi_DN structure and initializes it to NULL.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new();
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to the newly allocated, and still empty, Slapi_DN structure.

Description

This function creates a new Slapi_DN structure by allocating the necessary memory and initializing both DN and normalized DN values to NULL.

See Also

slapi_sdn_free()

slapi_sdn_copy()

slapi_sdn_done())

slapi_sdn_new_dn_byref()

Creates a new Slapi_DN structure and sets a DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_dn_byref(const char *dn);
```

Parameters

This function takes the following parameter:

Parameter	Description
dn	The DN value to be set in the new Slapi_DN structure.

Returns

This function returns a pointer to the new Slapi_DN structure with a DN value set to the content of dn.

Description

This function creates a new Slapi_DN structure and initializes its DN with the value of dn. The DN of the new structure will point to the same string pointed to by dn (the DN value is passed in to the parameter by reference). However, the FLAG_DN flag is not set and no counter is incremented.

Memory Concerns

The memory is allocated by the function itself.

See Also

slapi_sdn_new_dn_byval()

slapi_sdn_new_dn_passin()

slapi_sdn_new_dn_byval()

Creates a new Slapi_DN structure and sets a DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_dn_byval(const char *dn);
```

Parameters

This function takes the following parameter:

Parameter	Description
dn	The DN value to be set in the new Slapi_DN structure.

Returns

This function returns a pointer to the new Slapi_DN structure with a DN value set to the content of dn.

Description

This function creates a new Slapi_DN structure and initializes its DN with the value of dn. The DN of the new structure will point to a copy of the string pointed to by dn (the DN value is passed in to the parameter by value). The FLAG_DN flag is set and the internal counter is incremented.

Memory Concerns

The memory is allocated by the function itself.

See Also

slapi_sdn_new_dn_byref()

slapi_sdn_new_dn_passin()

slapi_sdn_new_dn_passin()

Creates a new Slapi_DN structure and sets a DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_dn_passin(const char *dn);
```

Parameters

This function takes the following parameter:

Parameter	Description
dn	The DN value to be set in the new Slapi_DN structure.

Returns

This function returns a pointer to the new `Slapi_DN` structure with DN value set to the content of `dn`.

Description

This function creates a new `Slapi_DN` structure and initializes its DN with the value of `dn`. The DN of the new structure will point to the string pointed to by `dn`. The `FLAG_DN` flag is set and the internal counter is incremented.

Memory Concerns

The memory is allocated by the function itself.

See Also

`slapi_sdn_new_dn_byval()`

`slapi_sdn_new_ndn_byref()`

slapi_sdn_new_ndn_byref()

Creates a new `Slapi_DN` structure and sets a normalized DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_ndn_byref(const char *ndn);
```

Parameters

This function takes the following parameter:

`ndn`	The normalized DN value to be set in the new `Slapi_DN` structure.

Returns

This function returns a pointer to the new `Slapi_DN` structure with a normalized DN value set to the content of `ndn`.

Description

This function creates a new `Slapi_DN` structure and initializes its normalized DN with the value of `ndn`. The normalized DN of the new structure will point to the same string pointed to by `ndn` (the normalized DN value is passed into the parameter by reference). However, the `FLAG_NDN` flag is not set and no counter is incremented.

Memory Concerns

The memory is allocated by the function itself.

See Also

`slapi_sdn_new_ndn_byval()`

slapi_sdn_new_ndn_byval()

Creates a new `Slapi_DN` structure and sets a normalized DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_ndn_byval(const char *ndn);
```

Parameters

This function takes the following parameter:

Parameter	Description
`ndn`	The normalized DN value to be set in the new `Slapi_DN` structure.

Returns

This function returns a pointer to the new `Slapi_DN` structure with a normalized DN value set to the content of `ndn`.

Description

This function creates a new `Slapi_DN` structure and initializes its normalized DN with the value of `ndn`. The normalized DN of the new structure will point to a copy of the string pointed to by `ndn` (the normalized DN value is passed into the parameter by value). The `FLAG_DND` flag is set and the internal counter is incremented.

Memory Concerns

The memory is allocated by the function itself.

See Also

`slapi_sdn_new_ndn_byref()`

slapi_sdn_scope_test()

Checks whether an entry, given its DN, is in the scope of a certain base DN.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_scope_test( const Slapi_DN *dn, const Slapi_DN *base,
    int scope );
```

Parameters

This function takes the following parameters:

Parameter	Description
dn	The DN of the entry subject of scope test.
base	The base DN to which dn is going to be tested against.
scope	The scope tested. This parameter can take one of the following levels • LDAP_SCOPE_BASE - where the entry DN should be the same as the base DN • LDAP_SCOPE_ONELEVEL - where the base DN should be the parent of the entry DN • LDAP_SCOPE_SUBTREE - where the base DN should at least be the suffix of the entry DN

Returns

This function returns non-zero if dn matches the scoping criteria given by base and scope.

Description

This function carries out a simple test to check whether the DN passed in the dn parameter is actually in scope of the base DN according to the values passed into the scope and base parameters.

See Also

slapi_sdn_compare()

slapi_sdn_isparent()

slapi_sdn_issuffix()

slapi_sdn_set_dn_byref()

Sets a DN value in a Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_dn_byref(Slapi_DN *sdn, const char *dn);
```

Parameters

This function takes the following parameters:

Parameter	Description
sdn	The target Slapi_DN structure.
dn	The DN value to be set in sdn.

Returns

This function returns a pointer to the Slapi_DN structure containing the new DN value.

Description

This function sets a DN value in a Slapi_DN structure. The DN of the new structure will point to the same string pointed to by dn (the DN value is passed into the parameter by value). However, the FLAG_DN flag is not set, and no internal counter is incremented.

See Also

slapi_sdn_set_dn_byval()

slapi_sdn_set_dn_passin()

slapi_sdn_set_dn_byval()

Sets a DN value in a Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_dn_byval(Slapi_DN *sdn, const char *dn);
```

Parameters

This function takes the following parameters:

Parameter	Description
sdn	The target Slapi_DN structure.
dn	The DN value to be set in sdn.

Returns

This function returns a pointer to the `Slapi_DN` structure containing the new DN value.

Description

This function sets a DN value in a `Slapi_DN` structure. The DN of the new structure will point to a copy of the string pointed to by `dn` (the DN value is passed into the parameter by value). The `FLAG_DN` flag is set, and the internal counters are incremented.

See Also

`slapi_sdn_set_ndn_byref()`

`slapi_sdn_set_dn_passin()`

slapi_sdn_set_dn_passin()

Sets a DN value in `Slapi_DN` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_dn_passin(Slapi_DN *sdn, const char *dn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`sdn`	The target `Slapi_DN` structure.
`dn`	The DN value to be set in `sdn`.

Returns

This function returns a pointer to the `Slapi_DN` structure containing the new DN value.

Description

This function sets a DN value in a `Slapi_DN` structure. The DN of the new structure will point to the same string pointed to by `dn`. The `FLAG_DN` flag is set, and the internal counters are incremented.

See Also

`slapi_sdn_set_dn_byval()`

`slapi_sdn_set_dn_byref()`

slapi_sdn_set_ndn_byref()

Sets a normalized DN in a Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_ndn_byref(Slapi_DN *sdn, const char *ndn);
```

Parameters

This function takes the following parameters:

Parameter	Description
sdn	The target Slapi_DN structure.
dn	The normalized DN value to be set in sdn.

Returns

This function returns a pointer to the Slapi_DN structure containing the new normalized DN value.

Description

This function sets a normalized DN value in a Slapi_DN structure. The normalized DN of the new structure will point to the same string pointed to by ndn (the normalized DN value is passed into the parameter by reference). However, the FLAG_DN flag is not set, and no internal counter is incremented.

See Also

slapi_sdn_set_ndn_byval()

slapi_sdn_set_dn_passin()

slapi_sdn_set_ndn_byval()

Sets a normalized DN value in Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_ndn_byval(Slapi_DN *sdn, const char *ndn);
```

Parameters

This function takes the following parameters:

Parameter	Description
sdn	The target Slapi_DN structure.
dn	The normalized DN value to be set in sdn.

Returns

This function returns a pointer to the Slapi_DN structure containing the new normalized DN value.

Description

This function sets a normalized DN value in a Slapi_DN structure. The normalized DN of the new structure will point to a copy of the string pointed to by ndn (the normalized DN value is passed into the parameter by value). The FLAG_DN flag is set, and the internal counters are incremented.

See Also

slapi_sdn_set_ndn_byref()

slapi_sdn_set_dn_passin()

slapi_sdn_set_parent()

Sets a new parent for a given entry.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_parent(Slapi_DN *sdn,
    const Slapi_DN *parentdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
sdn	The Slapi_DN structure containing the DN of the entry.
parentdn	The new parent DN.

Returns

The function returns a pointer to the `Slapi_DN` structure that contains the DN of the entry after the new parent DN has been set.

Description

This function sets a new parent for an entry. This is done by keeping the RDN of the original DN of the entry and by adding the DN of its new parent (the value of `parentdn`).

See Also

`slapi_sdn_isparent()`

`slapi_sdn_get_parent()`

slapi_sdn_set_rdn()

Sets a new RDN for given entry.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_rdn(Slapi_DN *sdn, const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`sdn`	The `Slapi_DN` structure containing the DN of the entry.
`rdn`	The new RDN.

Returns

This function returns a pointer to the `Slapi_DN` structure that keeps the DN of the entry after the new RDN has been set.

Description

This function sets a new RDN for an entry. This is done by retrieving the DN of the entry's parent of the origin DN of the entry and then adding it to the RDN (the value of `rdn`) to it.

See Also

`slapi_sdn_get_rdn()`

slapi_search_internal_callback_pb()

Performs an LDAP search operation based on a parameter block to search the directory. Unlike `slapi_search_internal_pb()`, this function allows you to specify callback functions that are invoked when the search operation finds matching entries or entries with referrals.

Syntax

```
#include "slapi-plugin.h"
int slapi_search_internal_callback_pb(Slapi_PBlock *pb,
    void *callback_data, plugin_result_callback prc,
    plugin_search_entry_callback psec,
    plugin_referral_entry_callback prec);
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	A parameter block that has been initialized using `slapi_search_internal_set_pb()`.
`callback_data`	A pointer to arbitrary plug-in or operation-specific data that you would like to pass to your callback functions.
`prc`	Callback function that the server calls to send result codes. The function must have the prototype specified by `plugin_result_callback`.
`psec`	Callback function that the server calls when finding a matching entry in the directory. The function must have the prototype specified by `plugin_search_entry_callback`.
`prec`	Callback function that the server calls when finding an entry that contains LDAP v3 referrals. The function must have the prototype specified by `plugin_referral_entry_callback`.

Returns

This function returns `-1` if the parameter passed is a `NULL` pointer. Otherwise, it returns `0`.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

Description

Like `slapi_search_internal_pb()`, this function allows you to search the directory from a plug-in function.

Unlike a search operation requested by a client, no result code, search entries, or referrals are sent to a client by `slapi_search_internal_callback_pb()`. However, you can write your own callback functions that are invoked when these events occur:

- You can write a callback function that is invoked when the search operation normally sends a result code to the client. This function must have the prototype specified by `plugin_result_callback`. You specify this function in the `prc` argument of `slapi_search_internal_callback_pb()`.
- You can write a callback function that is invoked when the search operation normally sends a search entry to the client. This function must have the prototype specified by `plugin_search_entry_callback`. You specify this function in the `psec` argument of `slapi_search_internal_callback_pb()`.
- You can write a callback function that is invoked when the search operation normally sends LDAP v3 search result references. This function must have the prototype specified by `plugin_referral_entry_callback`. You specify this function in the `prec` argument of `slapi_search_internal_callback_pb()`.

You can also pass arbitrary plug-in or operation-specific data to these callback functions. Specify the data that you want to pass as the `callback_data` argument of `slapi_search_internal_callback_pb()`.

Memory Concerns

The entries passed to the search entry callback function do not need to be freed. If you need to access an entry after returning from the callback function, call `slapi_entry_dup()` to make a copy.

The referral URLs passed to the referral entry callback function do not need to be freed. If you need to access a referral string after returning from the callback function, call `slapi_ch_strdup()` to make a copy.

You do not need to call `slapi_free_search_results_internal()` after calling `slapi_search_internal_callback_pb()`.

slapi_search_internal_get_entry()

Performs an internal search operation to read one entry (that is, it performs a base object search).

Syntax

```
#include "slapi-plugin.h"
int slapi_search_internal_get_entry( Slapi_DN *dn, char ** attrlist,
    Slapi_Entry **ret_entry, void *caller_identity);
```

Parameters

This function takes the following parameters:

Parameter	Description
`dn`	The DN of the entry to be read.
`attrlist`	A NULL terminated array of attribute types to return from entries that match filter. If you specify a NULL, all attributes will be returned.
`ret_entry`	The address of a `Slapi_Entry` pointer to receive the entry if it is found.
`caller_identity`	A plug-in or component identifier. This value can be obtained from the `SLAPI_PLUGIN_IDENTITY` field of the parameter block that is passed to your plug-in initialization function.

Returns

This function returns the LDAP result code for the search operation.

Description

This function performs an internal search operation to read one entry (that is, it preforms a base object search). If an entry named by `dn` is found, the `ret_entry` pointer will be set to point to a copy of the entry that contains the attribute values specified by the `attrlist` parameter.

Memory Concerns

The returned entry (`*ret_entry`) should be freed by calling `slapi_entry_free()`.

See Also

`slapi_search_internal_pb()`

`slapi_entry_free()`

slapi_search_internal_pb()

Performs an LDAP search operation based on a parameter block to search the directory.

Syntax

```
#include "slapi-plugin.h"
int slapi_search_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

Parameter	Description
pb	A parameter block that has been initialized using `slapi_search_internal_set_pb()`.

Returns

This function returns `-1` if the parameter passed is a `NULL` pointer. Otherwise, it returns `0`.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

Description

This function performs an internal search based on a parameter block. The parameter block should be initialized by calling the `slapi_search_internal_set_pb()` function.

Memory Concerns

`slapi_free_search_results_internal()` should be called to dispose of any entires and other items that were allocated by a call to `slapi_search_internal_pb()`.

slapi_search_internal_set_pb()

Sets a parameter block for an internal search.

Syntax

```
#include slapi-plugin.h

int slapi_search_internal_set_pb(Slapi_PBlock *pb,
    const char *base, int scope, const char *filter,
    char **attrs, int attrsonly,LDAPControl **controls,
    const char *uniqueid,Slapi_ComponentId *plugin_identity,
    int operation_flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block.
`base`	The base of the search.
`scope`	`LDAP_SCOPE_SUBTRREE`, `LDAP_SCOPE_ONELEVEL`, or `LDAP_SCOPE_BASE`
`filter`	The search filter.
`attrs`	Request attribute list.
`attrsonly`	Specifying `1` retrieves only the attribute types. Specifying `0` retrieves the attribute types and the attribute values.
`controls`	LDAP control that you wish to attach.
`uniqueid`	Unique ID must be set to `NULL`.
`plugin_identity`	Plug-in identifier obtained from `SLAPI_PLUGIN_IDENTITY` during plug-in initialization.
`operation_flags`	The only flag that is exposed in this release is `SLAPI_OP_FLAG_NEVER_CHAIN`. If this flag is set, then the search will not be conducted if the entry is a chained backend.

Returns

This function returns `0` if successful. Otherwise, it returns an LDAP error code.

Description

Use this function to set the pblock to perform an internal search. This function is needed in order to use the internal search operation function, `slapi_search_internal_pb()`.

You would typically create a Slapi_PBlock structure using slapi_pblock_new(), pass that parameter block to slapi_search_internal_pb(), and then pass the same pblock to actually do the search.

See Also

slapi_pblock_new()

slapi_search_internal_pb()

slapi_send_ldap_referral()

Processes an entry's LDAP v3 referrals (which are found in the entry's ref attribute). For LDAP v3 clients, this function sends the LDAP referrals back to the client. For LDAP v2 clients, this function copies the referrals to an array of berval structures that you can pass to slapi_send_ldap_result() function at a later time.

Syntax

```
#include "slapi-plugin.h"
int slapi_send_ldap_referral( Slapi_PBlock *pb, Slapi_Entry *e,
    struct berval **refs, struct berval ***urls );
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block.
e	Pointer to the Slapi_Entry structure representing the entry that you are working with.
refs	Pointer to the NULL-terminated array of berval structures containing the LDAP v3 referrals (search result references) found in the entry.
urls	Pointer to the array of berval structures used to collect LDAP referrals for LDAP v2 clients.

Returns

This function returns 0 if successful, or -1 if an error occurs.

Description

When you call this function, the server processes the LDAP referrals specified in the `refs` argument. The server processes referrals in different ways, depending on the version of the LDAP protocol supported by the client:

- In the LDAP v3 protocol, references to other LDAP servers (search result references) can be sent to clients as search results. (For example, a server can send a mixture of entries found by the search and references to other LDAP servers as the results of a search.)

  When you call the `slapi_send_ldap_referral()` function for LDAP v3 clients, the server sends the referrals specified in the `refs` argument back to the client as search result references. (The `urls` argument is not used in this case.)

- In the LDAP v2 protocol, servers can send the LDAP result code `LDAP_PARTIAL_RESULTS` to refer the client to other LDAP server.

  When you call the `slapi_send_ldap_referral()` function for LDAP v2 clients, the server collects the referrals specified in `refs` in the `urls` argument. No data is sent to the LDAP v2 client.

  To get the referrals to an LDAP v2 client, you need to pass the `urls` argument (along with an `LDAP_PARTIAL_RESULTS` result code) to the `slapi_send_ldap_result()` function. `slapi_send_ldap_result()` concatenates the referrals specified in the `urls` argument and sends the resulting string to the client as part of the error message.

If you want to define your own function for sending referrals, write a function that complies with the type definition `send_ldap_search_entry_fn_ptr_t` and set the `SLAPI_PLUGIN_DB_REFERRAL_FN` parameter in the parameter block to the name of your function.

See Also

`slapi_send_ldap_result()`

`slapi_send_ldap_search_entry()`.

slapi_send_ldap_result()

Sends an LDAP result code back to the client.

Syntax

```
#include "slapi-plugin.h"
void slapi_send_ldap_result( Slapi_PBlock *pb, int err,
    char *matched, char *text, int nentries, struct berval **urls );
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block.
err	LDAP result code that you want sent back to the client (for example, LDAP_SUCCESS).
matched	When sending back an LDAP_NO_SUCH_OBJECT result code, use this argument to specify the portion of the target DN that could be matched. (Pass NULL in other situations.)
text	Error message that you want sent back to the client. (Pass NULL if you do not want an error message sent back.)
nentries	When sending back the result code for an LDAP search operation, use this argument to specify the number of matching entries found.
urls	When sending back an LDAP_PARTIAL_RESULTS result code to an LDAP v2 client or an LDAP_REFERRAL result code to an LDAP v3 client, use this argument to specify the array of berval structures containing the referral URLs. (Pass NULL in other situations.)

Description

Call slapi_send_ldap_result() to send an LDAP result code (such as LDAP_SUCCESS) back to the client.

The following arguments are intended for use only in certain situations:

- matched

  When sending an LDAP_NO_SUCH_OBJECT result code back to a client, use matched to specify how much of the target DN could be found in the database. For example, if the client was attempting to find the DN:

  cn=Babs Jensen, ou=Product Division, o=Example, c=US

  and the database contains entries for c=US and o=Example, c=US, but no entry for ou=Product Division, o=Example, c=US, you should set the matched parameter to:

  o=Example, c=US

- urls

When sending an `LDAP_PARTIAL_RESULTS` result code back to an LDAP v2 client or an `LDAP_REFERRAL` result code back to an LDAP v3 client, use `urls` to specify the referral URLs.

For LDAP v3 referrals, you can call the `slapi_str2filter()` to send referrals to LDAP v3 clients and collect them for LDAP v2 clients. You can pass the array of collected referrals to the `urls` argument of `slapi_send_ldap_results()`. For example:

```
struct berval **urls;
...
slapi_send_ldap_referral(ld, e, &refs, &urls);
slapi_send_ldap_result(ld,LDAP_PARTIAL_RESULTS,NULL,NULL,0,urls);
```

If you want to define your own function for sending result codes, write a function that complies with the type definition `send_ldap_search_entry_fn_ptr_t` and set the `SLAPI_PLUGIN_DB_RESULT_FN` parameter in the parameter block to the name of your function.

See Also

`slapi_str2filter()`

`slapi_send_ldap_search_entry()`.

slapi_send_ldap_search_entry()

Sends an entry found by a search back to the client.

Syntax

```
#include "slapi-plugin.h"
int slapi_send_ldap_search_entry( Slapi_PBlock *pb, Slapi_Entry *e,
    LDAPControl **ectrls, char **attrs, int attrsonly );
```

Parameters

This function takes the following parameters:

Parameter	Description
`pb`	Parameter block.
`e`	Pointer to the `Slapi_Entry` structure representing the entry that you want to send back to the client.
`ectrls`	Pointer to the array of `LDAPControl` structures representing the controls associated with the search request.

`attrs`	Attribute types specified in the LDAP search request
`attrsonly`	Specifies whether or not the attribute values should be sent back with the result. • If 0, the values are included. • If 1, the values are not included.

Returns

This function returns 0 if successful, 1 if the entry is not sent (for example, if access control did not allow it to be sent), or -1 if an error occurs.

Description

Call `slapi_send_ldap_search_entry()` to send an entry found by a search back to the client.

`attrs` is the array of attribute types that you want to send from the entry. This value is equivalent to the `SLAPI_SEARCH_ATTRS` parameter in the parameter block.

`attrsonly` specifies whether you want to send only the attribute types or the attribute types and their values:

- Pass 0 for this parameter if you want to send both the attribute types and values to the client.
- Pass 1 for this parameter if you want to send only the attribute types (not the attribute values) to the client.

This value is equivalent to the `SLAPI_SEARCH_ATTRSONLY` parameter in the parameter block.

If you want to define your own function for sending entries, write a function complies with the type definition `send_ldap_search_entry_fn_ptr_t` and set the `SLAPI_PLUGIN_DB_ENTRY_FN` parameter in the parameter block to the name of your function.

See Also

`slapi_str2filter()`

`slapi_send_ldap_search_entry()`

slapi_set_object_extension()

Modify an object extension.

Syntax

```
#include "slapi-plugin.h"
void slapi_set_object_extension(int objecttype, void *object,
    int extensionhandle, void *extension);
```

Parameters

This function takes the following parameters:

Parameter	Description
`objecttype`	Set by the server and used to retrieve the extension
`object`	Extended object
`extensionhandle`	Set by the server and used to retrieve the extension
`extension`	New extension to attach to the object

Description

This function modifies an extension to an object.

See Also

`slapi_register_object_extension()`

slapi_str2entry()

Converts an LDIF description of a directory entry (a string value) into an entry of the `Slapi_Entry` type.

Syntax

```
#include "slapi-plugin.h"
Slapi_Entry *slapi_str2entry( char *s, int flags );
```

Parameters

This function takes the following parameters:

Parameter	Description
`s`	Description of an entry that you want to convert to `Slapi_Entry`.
`flags`	One or more flags specifying how the entry should be generated

The value of the `flags` argument can be one of the following values:

Argument	Description
`SLAPI_STR2ENTRY_REMOVEDUPVALS`	Removes any duplicate values in the attributes of the entry.
`SLAPI_STR2ENTRY_ADDRDNVALS`	Adds the relative distinguished name (RDN) components (for example, `uid=bjensen`) as attributes of the entry.

Returns

Pointer to the `Slapi_Entry` structure representing the entry, or `NULL` if the string cannot be converted (for example, if no DN is specified in the string).

Description

A directory entry can be described by a string in LDIF format.

Calling the `slapi_str2entry()` function converts a string description in this format to a `Slapi_Entry` structure, which you can pass to other API functions.

NOTE This function modifies the string argument `s`. If you must use the string value again, make a copy of this string before calling `slapi_str2entry()`.

If an error occurred during the conversion process, the function returns `NULL` instead of the entry.

When you are finished working with the entry, you should call the `slapi_entry_free()` function.

To convert an entry to a string description, call the `slapi_filter_free()` function.

Memory Concerns

Do not use `free()` or `slapi_ch_free()` to free the entry. Always use `slapi_entry_free()` instead.

See Also

`slapi_filter_free()`.

`slapi_entry_free()`

slapi_str2filter()

Converts a string description of a search filter into a filter of the `Slapi_Filter` type.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_str2filter( char *str );
```

Parameters

This function takes the following parameter:

Parameter	Description
`str`	String description of a search filter.

Memory Concerns

Do not pass a static string to this function. Instead, create a duplicate using `slapi_ch_strdup()`. When you are finished working with this filter, you should free the `Slapi_Filter` structure by calling `slapi_filter_free()`.

Returns

Pointer to the `Slapi_Filter` structure representing the search filter, or NULL if the string cannot be converted (for example, if an empty string is specified or if the filter syntax is incorrect).

slapi_lock_mutex()

Try to release a mutex.

Syntax

```
#include "slapi-plugin.h"
int slapi_unlock_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

Parameter	Description
`mutex`	Mutex for thread synchronization

Description

This function allows thread synchronization. Once a thread has locked a mutex using `slapi_lock_mutex()`, other threads attempting to acquire the lock are blocked until the thread holding the mutex calls this function.

Returns

This function returns `1` if the mutex could be unlocked. Otherwise, it returns `0`.

See Also

`slapi_destroy_mutex()`

`slapi_lock_mutex()`

`slapi_new_mutex()`

slapi_UTF8CASECMP()

Compares two UTF8 strings.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF8CASECMP(char *s0, char *s1);
```

Parameters

This function takes the following parameters:

Parameter	Description
`s0`	A `NULL`-terminated UTF8 string.
`s1`	A `NULL`-terminated UTF8 string.

Returns

This function returns one of the following values:

- positive number if `s0` is after `s1`.
- `0` if the two string are identical, ignoring case.
- negative number if `s1` is after `s0`.

Description

This function has the following rules:

- If both UTF8 strings are `NULL` or 0-length, `0` is returned.
- If one of the strings is `NULL` or 0-length, the `NULL` or 0-length string is smaller.
- If one or both of the strings are not UTF8, system provided `strcasecmp` is used.
- If one of the two strings contains no 8-bit characters, `strcasecmp` is used.
- The strings are compared after they are converted to lower-case UTF8.
- Each character is compared from the beginning.

Evaluation occurs in this order:

- If the length of one character is shorter then the other, the difference of the two lengths is returned.
- If the length of the corresponding characters is the same, each byte in the characters is compared.
- If there is a difference between two bytes, the difference is returned.
- If one string is shorter then the other, the difference is returned.

slapi_UTF8NCASECMP()

Compares a specified number of UTF8 characters.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF8NCASECMP(char *s0, char *s1, int n);
```

Parameters

This function takes the following parameters:

Parameter	Description
`s0`	A `NULL`-terminated UTF8 string.
`s1`	A `NULL`-terminated UTF8 string.
`n`	The number of UTF8 characters (not bytes) from `s0` and `s1` to compare.

Returns

This function returns one of the following values:

- positive number if `s0` is after `s1`.
- `0` if the two string are identical, ignoring case.
- negative number if `s1` is after `s0`.

Description

This function has the following rules:

- If both UTF8 strings are `NULL` or 0-length, `0` is returned.
- If one of the strings is `NULL` or 0-length, the `NULL` or 0-length string is smaller.
- If one or both of the strings are not UTF8, system provided `strcasecmp` is used.
- If one of the two strings contains no 8-bit characters, `strcasecmp` is used.
- The strings are compared after they are converted to lower-case UTF8.
- Each character is compared from the beginning.

Evaluation occurs in this order:

- If the length of one character is shorter then the other, the difference of the two lengths is returned.
- If the length of the corresponding characters is the same, each byte in the characters is compared.
- If there is a difference between two bytes, the difference is returned.
- If one string is shorter then the other, the difference is returned.

slapi_UTF8ISLOWER()

Verifies if a UTF8 character is lower case.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF8ISLOWER(char *s);
```

Parameters

This function takes the following parameter:

Parameter	Description

s	Pointer to a single UTF8 character (could be multiple bytes).

Returns

This function returns `1` if the character is a lower case letter, or `0` if it is not.

slapi_UTF8ISUPPER()

Verifies if a single UTF8 character is upper case.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF8ISUPPER(char *s);
```

Parameters

This function takes the following parameter:

Parameter	Description
s	Pointer to a single UTF8 character (could be multiple bytes).

Returns

This function returns `1` if the character is an upper case letter, or `0` if it is not.

slapi_unlock_mutex()

Unlocks the specified mutex.

Syntax

```
#include "slapi-plugin.h"
int slapi_unlock_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

Parameter	Description

`mutex`	Pointer to an `Slapi_Mutex` structure representing the mutex that you want to unlock.

Returns

One of the following values:

- A non-zero value if the mutex was successfully unlocked.
- 0 if the mutex was `NULL` or was not locked by the calling thread.

Description

This function unlocks the mutex specified by the `Slapi_Mutex` structure.

slapi_UTF8STRTOLOWER()

Converts a UTF8 string to lower case.

Syntax

```
#include "slapi-plugin.h"
unsigned char *slapi_UTF8STRTOLOWER(char *s);
```

Parameters

This function takes the following parameter:

Parameter	Description
`s`	A `NULL`-terminated UTF8 string to be converted to lower case.

Returns

This function returns a pointer to a null-terminated UTF8 string whose characters are converted to lower case. Characters which are not upper case are copied as is. If the string is not found to be a UTF8 string, this function returns `NULL`.

Description

This function converts a string of multiple UTF8 characters, and not a single character, as in `slapi_UTF8TOLOWER()`.

Memory Concerns

The output string is allocated, and needs to be released when it is no longer needed.

See Also

`slapi_UTF8TOLOWER()`

slapi_UTF8STRTOUPPER()

Converts a string made up of UTF8 characters and converts it to upper case.

Syntax

```
#include "slapi-plugin.h"
unsigned char *slapi_UTF8STRTOUPPER(char *s);
```

Parameters

This function takes the following parameter:

Parameter	Description
s	A `NULL`-terminated UTF8 string to be converted to upper case.

Returns

This function returns a null-terminated UTF8 string whose characters are converted to upper case. Character which are not lower case are copied as is. If the string is not considered to be a UTF8 string, this function returns `NULL`.

Memory Concerns

The output string is allocated in this function, and needs to be released when it is no longer used.

slapi_UTF8TOLOWER()

Converts a UTF8 character to lower case.

Syntax

```
#include "slapi-plugin.h"
void slapi_UTF8TOLOWER(char *s, char *d, int *ssz, int *dsz);
```

Parameters

This function takes the following parameters:

Parameter	Description
s	A single UTF8 character (could be multiple bytes).
d	Pointer to the lower case form of s. The memory for this must be allocated by the caller before calling the function.
ssz	Returns the length in bytes of the input character.
dsz	Returns the length in bytes of the output character.

slapi_UTF8TOUPPER()

Converts a lower case UTF8 character to an upper case character.

Syntax

```
#include "slapi-plugin.h"
void slapi_UTF8TOUPPER(char *s, char *d, int *ssz, int *dsz);
```

Parameters

This function takes the following parameters:

Parameter	Description
s	Pointer to a single UTF8 character (could be multiple bytes).
d	Pointer to the upper case version of s. The memory for this must be allocated by the caller before calling the function.
ssz	Returns the length in bytes of the input character.
dsz	Returns the length in bytes of the output character.

slapi_value_compare()

Compares two values for a given attribute to determine if they are equals.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_compare(const Slapi_Attr *a, const Slapi_Value *v1,
    const Slapi_Value *v2);
```

Parameters

This function takes the following parameters:

Parameter	Description
a	A pointer to an attribute used to determine how the two values will be compared.
v1	Pointer to the Slapi_Value structure containing the first value to compare.
v2	Pointer to the Slapi_Value structure containing the second value to compare.

Returns

This function returns one of the following values:

- 0 if the two values are equal.
- -1 if v1 is smaller than v2.
- 1 if v1 is greater than v2.

Description

This function compares two Slapi_Values using the matching rule associated to the attribute a.

This function replaces the deprecated slapi_attr_value_cmp() function used in previous releases, and uses the Slapi_Value attribute values instead of the berval attribute values.

slapi_value_dup()

Duplicates a value.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_dup(const Slapi_Value *v);
```

Parameters

This function takes the following parameter:

Parameter	Description

v	Pointer to the `Slapi_Value` structure you wish to duplicate.

Returns

This function returns a pointer to a newly allocated `Slapi_Value`.

Memory Concerns

The new `Slapi_Value` is allocated and needs to be freed by the caller, using `slapi_value_free()`.

See Also

`slapi_value_free()`

slapi_value_free()

Frees the specified `Slapi_Value` structure and its members from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_value_free(Slapi_Value **value);
```

Parameters

This function takes the following parameter:

Parameter	Description
`value`	Address of the pointer to the `Slapi_Value` you wish to free.

Description

This function frees the `Slapi_Value` structure and its members (if it is not `NULL`), and sets the pointer to `NULL`.

Memory Concerns

Call this function when you are finished working with the structure.

slapi_valuearray_free()

Frees the specified array of `Slapi_Value` structures and their members from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_valuearray_free(Slapi_Value ***value);
```

Parameters

This function takes the following parameter:

Parameter	Description
`value`	Array of `Slapi_Value` structures to free.

Description

This function frees each `Slapi_Value` structure and its members, and sets the pointer to `NULL`.

Memory Concerns

Call this function when you are finished working with the array of values.

slapi_value_get_berval()

Gets the `berval` structure of the value.

Syntax

```
#include "slapi-plugin.h"
const struct berval * slapi_value_get_berval(
    const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

Parameter	Description
`value`	Pointer to the `Slapi_Value` of which you wish to get the `berval`.

Returns

Returns a pointer to the `berval` structure contained in the `Slapi_Value`. This function returns a pointer to the actual `berval` structure, and not a copy of it.

Memory Concerns

You should not free the `berval` structure unless you plan to replace it by calling `slapi_value_set_berval()`.

See Also

`slapi_value_set_berval()`

slapi_value_get_int()

Converts the value to an integer.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_get_int(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

Parameter	Description
`value`	Pointer to the `Slapi_Value` of which you wish to get as an integer.

Returns

This function returns an integer that corresponds to the value stored in the `Slapi_Value` structure, or `0` if there is no value.

Description

Converts the value in the `Slapi_Value` to an integer.

See Also

`slapi_value_get_long()`

`slapi_value_get_uint()`

slapi_value_get_length()

Gets the actual length of the value.

Syntax

```
#include "slapi-plugin.h"
size_t slapi_value_get_length(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

Parameter	Description
`value`	Pointer to the `Slapi_Value` of which you wish to get the length.

Returns

This function returns the length of the value contained in `Slapi_Value`, or `0` if there is no value.

Description

This function returns the actual length of a value contained in the `Slapi_Value` structure.

slapi_value_get_long()

Converts the value into a long integer.

Syntax

```
#include "slapi-plugin.h"
long slapi_value_get_long(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

Parameter	Description
`value`	Pointer to the `Slapi_Value` of which you wish to get as a long integer.

Returns

This function returns a long integer that corresponds to the value stored in the `Slapi_Value` structure, or `0` if there is no value.

Description

This function converts the value contained in the `Slapi_Value` structure into a long integer.

See Also

`slapi_value_get_int()`

`slapi_value_get_ulong()`

`slapi_value_get_uint()`

slapi_value_get_string()

Returns the value as a string.

Syntax

```
#include "slapi-plugin.h"
const char * slapi_value_get_string(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

Parameter	Description
`value`	Pointer to the `Slapi_Value` of which you wish to get as a string.

Returns

This function returns a string containing the value, or `NULL` if there is no value.

This function returns a pointer to the actual string value in `Slapi_Value`, not a copy of it.

Memory Concerns

You should not free the string unless to plan to replace it by calling `slapi_value_set_string()`.

See Also

`slapi_value_set_string()`

slapi_value_get_uint()

Converts the value to an unsigned integer.

Syntax

```
#include "slapi-plugin.h"
unsigned int slapi_value_get_uint(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

Parameter	Description
`value`	Pointer to the `Slapi_Value` of which you wish to get as an unsigned integer.

Returns

This function returns an unsigned integer that corresponds to the value stored in the `Slapi_Value` structure, or `0` if there is no value.

Description

Converts the value contained in `Slapi_Value` into an unsigned integer.

See Also

`slapi_value_get_int()`

`slapi_value_get_long()`

`slapi_value_get_ulong()`

slapi_value_get_ulong()

Converts the value into an unsigned long.

Syntax

```
#include "slapi-plugin.h"
unsigned long slapi_value_get_ulong(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

Parameter	Description
value	Pointer to the Slapi_Value of which you wish to get as an unsigned long integer.

Returns

This function returns an unsigned long integer that corresponds to the value stored in the Slapi_Value structure, or 0 if there is no value.

Description

Converts the value contained in the Slapi_Value structure into an unsigned long integer.

See Also

slapi_value_get_long()

slapi_value_get_int()

slapi_value_get_uint()

slapi_value_init()

Initializes a Slapi_Value structure with no value.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_init(Slapi_Value *v);
```

Parameters

This function takes the following parameter:

Parameter	Description
v	Pointer to the value to be initialized. The pointer must not be NULL.

Returns

This function returns a pointer to the initialized Slapi_Value structure (itself).

Description

This function initializes the `Slapi_Value` structure, resetting all of its fields to zero. The value passed as the parameter must be a valid `Slapi_Value`.

slapi_value_init_berval()

Initializes a `Slapi_Value` structure from the `berval` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_init_berval(Slapi_Value *v,
    struct berval *bval);
```

Parameters

This function takes the following parameters:

Parameter	Description
v	Pointer to the value to initialize. The pointer must not be NULL.
bval	Pointer to the `berval` structure to be used to initialize the value.

Returns

This function returns a pointer to the initialized `Slapi_Value` structure (itself).

Description

This function initializes the `Slapi_Value` structure with the value contained in the `berval` structure. The content of the `berval` structure is duplicated.

slapi_value_init_string()

Initializes a `Slapi_Value` structure from a string.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_init_string(Slapi_Value *v,const char *s);
```

Parameters

This function takes the following parameters:

Parameter	Description
v	Pointer to the value to be initialized. The pointer must not be NULL.
s	NULL-terminated string used to initialize the value.

Returns

This function returns a pointer to the initialized Slapi_Value structure (itself).

Description

This function initializes the Slapi_Value structure with the value contained in the string. The string is duplicated.

slapi_value_init_string_passin()

Initializes a Slapi_Value structure with value contained in the string.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_init_string_passin (Slapi_value *v,
    char *s);
```

Parameters

This function takes the following parameters:

Parameter	Description
v	Pointer to the value to be initialized. The pointer must not be NULL.
s	NULL-terminated string used to initialize the value.

Returns

This function returns a pointer to the initialized Slapi_Value structure (itself).

Description

This function initializes a Slapi_Value structure with the value contained in the string. The string is not duplicated and must be freed.

Memory Concerns

The string will be freed when the `Slapi_Value` structure is freed from memory by calling `slapi_value_free()`.

See Also

`slapi_value_free()`

`slapi_value_new_string_passin()`

`slapi_value_set_string_passin()`

slapi_value_new()

Allocates a new `Slapi_Value` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new();
```

Parameters

This function does not take any parameters.

Returns

This function returns a pointer to the newly allocated `Slapi_Value` structure. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Description

This function returns an empty `Slapi_Value` structure. You can call other functions of the API to set the value.

Memory Concerns

When you are no longer using the value, free it from memory by calling `slapi_value_free()`.

See Also

`slapi_value_dup()`

`slapi_value_free()`

`slapi_value_new_berval()`

slapi_value_new_berval()

Allocates a new Slapi_Value structure and initializes it from a berval structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new_berval(const struct berval *bval);
```

Parameters

This function takes the following parameter:

Parameter	Description
bval	Pointer to the berval structure used to initialize the newly allocated Slapi_Value.

Returns

This function returns a pointer to the newly allocated Slapi_Value. If space cannot be allocated (for example, if no more virtual memory exists), the slapd program will terminate.

Description

This function returns a Slapi_Value structure containing a value duplicated from the berval structure passed as the parameter.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling slapi_value_free().

See Also

slapi_value_new()

slapi_value_dup()

slapi_value_free()

slapi_value_new_string()

slapi_value_new_string()

Allocates a new Slapi_Value structure and initializes it from a string.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new_string(const char *s);
```

Parameters

This function takes the following parameter:

Parameter	Description
s	NULL-terminated string used to initialize the newly allocated Slapi_Value.

Returns

This function returns a pointer to the newly allocated Slapi_Value. If space cannot be allocated (for example, if no more virtual memory exists), the slapd program will terminate.

Description

This function returns a Slapi_Value structure containing a value duplicated from the string passed as the parameter.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling slapi_value_free().

See Also

slapi_value_new()

slapi_value_new_berval()

slapi_value_free()

slapi_value_dup()

slapi_value_new_string_passin()

Allocates a new Slapi_Value structure and initializes it from a string.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new_string_passin ( char *s );
```

Parameters

This function takes the following parameter:

Parameter	Description
s	NULL-terminated string used to initialize the newly allocated Slapi_Value structure.

Returns

This function returns a pointer to a newly allocated Slapi_Value structure. If space cannot be allocated (for example, if no virtual memory exists), the slapd program terminates.

Description

This function returns a Slapi_Value structure containing the string passed as the parameter. The string passed in must not be freed from memory.

Memory Concerns

The value should be freed by the caller, using slapi_value_free().

See Also

slapi_value_free()

slapi_value_dup()

slapi_value_new()

slapi_value_new_value()

Allocates a new Slapi_Value structure and initializes it from another Slapi_Value structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new_value(const Slapi_Value *v);
```

Parameters

This function takes the following parameter:

Parameter	Description

v	Pointer to the `Slapi_Value` structure used to initialize the newly allocated `Slapi_Value`.

Returns

This function returns a pointer to the newly allocated `Slapi_Value`. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program will terminated.

Description

This function returns a `Slapi_Value` structure containing a value duplicated from the `Slapi_Value` structure passed as the parameter. This function is identical to `slapi_value_dup()`.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling the `slapi_value_free()` function/

See Also

`slapi_value_dup()`

`slapi_value_free()`

slapi_value_set()

Sets the value in a `Slapi_Value` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_set(Slapi_Value *value, void *val,
    unsigned long len);
```

Parameters

This function takes the following parameters:

Parameter	Description
`value`	Pointer to the `Slapi_Value` in which to set the value.
`val`	Pointer to the value.
`len`	Length of the value.

Returns

This function returns a pointer to the `Slapi_Value` with the value set.

Description

This function sets the value in the `Slapi_Value` structure. The value is a duplicate of the data pointed to by `val` and of length `len`.

Memory Concerns

If the pointer to the `Slapi_Value` structure is `NULL`, then nothing is done and the function returns `NULL`. If the `Slapi_Value` structure already contains a value, it is freed from memory before the new one is set.

When you are no longer using the `Slapi_Value` structure, you should free it from memory by calling `slapi_value_free()`.

See Also

`slapi_value_free()`

slapi_value_set_berval()

Copies the value from a `berval` structure into a `Slapi_Value` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_set_berval(Slapi_Value *value,
    const struct berval *bval );
```

Parameters

This function takes the following parameters:

Parameter	Description
`value`	Pointer to the `Slapi_Value` structure in which to set the value.
`bval`	Pointer to the `berval` value to be copied.

Returns

This function returns the pointer to the `Slapi_Value` structure passed as the parameter, or `NULL` if it was `NULL`.

Description

This function sets the value of `Slapi_Value` structure. The value is duplicated from the `berval` structure `bval`.

Memory Concerns

If the pointer to the `Slapi_Value` structure is `NULL`, nothing is done and the function returns `NULL`. If the `Slapi_Value` already contains a value, it is freed from memory before the new one is set.

When you are no longer using the `Slapi_Value` structure, you should free it from memory by calling `slapi_value_free()`.

See Also

`slapi_value_free()`

slapi_value_set_int()

Sets the integer value of a `Slapi_Value` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_set_int(Slapi_Value *value, int intVal);
```

Parameters

This function takes the following parameters:

Parameter	Description
`value`	Pointer to the `Slapi_Value` structure in which to set the integer value.
`intVal`	The integer containing the value to set.

Returns

This function returns one of the following values:

- `0` if the value is set.
- `-1` if the pointer to the `Slapi_Value` is `NULL`.

Description

This function sets the value of the `Slapi_Value` structure from the integer `intVal`.

Memory Concerns

If the pointer to the Slapi_Value structure is NULL, nothing is done and the function returns -1. If the Slapi_Value already contains a value, it is freed from memory before the new one is set.

When you are no longer using the Slapi_Value structure, you should free it from memory by calling slapi_value_free().

See Also

slapi_value_free()

slapi_value_set_string()

Copies a string in the value of a Slapi_Value structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_set_string(Slapi_Value *value, const char *strVal);
```

Parameters

This function takes the following parameters:

Parameter	Description
value	Pointer to the Slapi_Value structure in which to set the value.
strVal	The string containing the value to set.

Returns

This function returns one of the following:

- 0 if value is set.
- -1 if the pointer to the Slapi_Value is NULL.

Description

This function sets the value of the Slapi_Value structure by duplicating the string strVal.

Memory Concerns

If the pointer to the `Slapi_Value` is `NULL`, nothing is done and the function returns `-1`. If the `Slapi_Value` already contains a value, it is freed from memory before the new one is set.

When you are no longer using the `Slapi_Value` structure, you should free it from memory by calling `slapi_value_free()`.

See Also

`slapi_value_free()`

slapi_value_set_string_passin()

Sets the value of a `Slapi_Value` structure from a string.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_set_string_passin(Slapi_Value *value, char *strVal);
```

Parameters

This function takes the following parameters:

Parameter	Description
`value`	Pointer to the `Slapi_Value` structure in which to set the value.
`strVal`	The string containing the value to set.

Returns

This function returns `0` if the value is set, or `-1` if the pointer to the `Slapi_Value` structure is `NULL`.

Description

This function sets the value of `Slapi_Value` structure with the string `strVal`. If the `Slapi_Value` structure already contains a value, it is freed from memory before the new one is set. The string `strVal` must not be freed from memory.

Memory Concerns

Use `slapi_value_free()` when you are finished working with the structure to free it from memory.

slapi_value_set_value()

Copies the value of a Slapi_Value structure into a Slapi_Value structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_set_value(Slapi_Value *value,
    const Slapi_Value *vfrom);
```

Parameters

This function takes the following parameters:

Parameter	Description
value	Pointer to the Slapi_Value in which to set the value.
vfrom	Pointer to the Slapi_Value from which to get the value.

Returns

This function returns the pointer to the Slapi_Value structure passed as the parameter, or NULL if it was NULL.

Description

This function sets the value of the Slapi_Value structure. This value is duplicated from the Slapi_Value structure vfrom. vfrom must not be NULL.

Memory Concerns

If the pointer to the Slapi_Value is NULL, nothing is done and the function returns NULL. If the Slapi_Value already contains a value, it is freed from before the new one is set.

When you are no longer using the Slapi_Value structure, you should free it from memory by calling slapi_value_free().

See Also

slapi_value_free()

slapi_valueset_add_value()

Adds a Slapi_Value in the Slapi_ValueSet structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_add_value(Slapi_ValueSet *vs,
    const Slapi_Value *addval);
```

Parameters

This function takes the following parameters:

Parameter	Description
vs	Pointer to the Slapi_ValueSet structure to which to add the value.
addval	Pointer to the Slapi_Value to add to the Slapi_ValueSet.

Description

This function adds a value in the form of a Slapi_Value structure in a Slapi_ValueSet structure.

Memory Concerns

The value is duplicated from the Slapi_Value structure, which can be freed from memory after using it without altering the Slapi_ValueSet structure.

This function does not verify if the value is already present in the Slapi_ValueSet structure. You can manually check this using slapi_valueset_first_value() and slapi_valueset_next_value().

See Also

slapi_valueset_first_value()

slapi_valueset_next_value()

slapi_valueset_count()

Returns the number of values contained in a Slapi_ValueSet structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_valueset_count(const Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameter:

Parameter	Description
vs	Pointer to the Slap_ValueSet structure of which you wish to get the count.

Returns

This function returns the number of values contained in the Slapi_ValueSet structure.

slapi_valueset_done()

Frees the values contained in the Slapi_ValueSet structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_done(Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameter:

Parameter	Description
vs	Pointer to the Slapi_ValueSet structure from which you wish to free its values.

Memory Concerns

Use this function when you are no longer using the values, but you want to reuse the Slapi_ValueSet structure for a new set of values.

slapi_valueset_find()

Finds the value in a value set using the syntax of an attribute.

```
#include "slapi-plugin.h"
Slapi_Value *slapi_valueset_find(const Slapi_Attr *a,
    const Slapi_ValueSet *vs, const Slapi_Value *v);
```

Parameters

This function takes the following parameters:

Parameter	Description
a	Pointer to the attribute. This is used to determine the syntax of the values and how to match them.
vs	Pointer to the `Slapi_ValueSet` structure from which you wish to get the value.
v	Address of the pointer to the `Slapi_Value` structure for the returned value.

Returns

This function returns a pointer to the value in the value set if the value was found. Otherwise, it returns `NULL`.

Description

Use this function to check for duplicate values in an attribute.

slapi_valueset_first_value()

Gets the first value of a `Slapi_ValueSet` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_valueset_first_value(Slapi_ValueSet *vs, Slapi_Value **v);
```

Parameters

This function takes the following parameters:

Parameter	Description
vs	Pointer to the `Slapi_ValueSet` structure from which you wish to get the value.
v	Address of the pointer to the `Slapi_Value` structure for the returned value.

Returns

This function returns the index of the value in the `Slapi_ValueSet`, or `-1` if there was no value.

Description

Call this function when you wish to get the first value of a `Slapi_ValueSet`, or you wish to iterate through all of the values. The returned value is the index of the value in the `Slapi_ValueSet` structure and must be passed to call `slapi_valueset_next_value()` to get the next value.

Memory Concerns

This function gives a pointer to the actual value within the `Slapi_ValueSet`. You should not free it from memory.

See Also

`slapi_valueset_next_value()`

slapi_valueset_free()

Frees the specified `Slapi_ValueSet` structure and its members from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_free(Slapi_ValueSet *vs)
```

Parameters

This function takes the following parameter:

Parameter	Description
`vs`	Pointer to the `Slapi_ValueSet` to free.

Description

This function frees the `Slapi_ValueSet` structure and its members if it is not `NULL`. Call this function when you are finished working with the structure.

See Also

`slapi_valueset_done()`

slapi_valueset_init()

Resets a `Slapi_ValueSet` structure to no values.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_init(Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameter:

Parameter	Description
`vs`	Pointer to the `Slapi_ValueSet` to reset.

Description

This function initializes the values contained in the `Slapi_ValueSet` structure (sets them to 0). This does not free the values contained in the structure. To free the values, use `slapi_valueset_done()`.

Memory Concerns

When you are no longer using the `Slapi_ValueSet` structure, you should free it from memory by using `slapi_valueset_free()`.

See Also

`slapi_valueset_done()`

`slapi_valueset_free()`

slapi_valueset_new()

Allocates a new `Slapi_ValueSet` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_ValueSet *slapi_valueset_new( void );
```

Parameters

This function takes no parameters.

Returns

This function returns a a pointer to the newly allocated `Slapi_ValueSet` structure. If no space could be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Description

This function returns an empty `Slapi_ValueSet` structure. You can call other slapi_valuset functions of the API to set the values in the `Slapi_ValueSet` structure.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling `slapi_valueset_free()`.

See Also

`slapi_valueset_free()`

slapi_valueset_next_value()

Gets the next value from a `Slapi_ValueSet` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_valueset_next_value( Slapi_ValueSet *vs, int index,
    Slapi_Value **v);
```

Parameters

This function takes the following parameters:

Parameter	Description
`vs`	Pointer to the `Slapi_ValueSet` structure from which you wish to get the value.
`index`	Value returned by the previous call to slapi_valueset_next_value or `slapi_value_first_value()`.
`v`	Address to the pointer to the `Slapi_Value` structure for the returned value.

Returns

This function returns the index of the value in the `Slapi_ValueSet`, or `-1` if there was no more value or the input index is incorrect.

Description

Call this function when you wish to get the next value of a `Slapi_ValueSet`, after having first called `slapi_valueset_first_value()`. The returned value is the index of the value in the `Slapi_ValueSet` structure and must be passed to `slapi_valueset_next_value()`.

Memory Concerns

This function gives a pointer to the actual value within the `Slapi_ValueSet` and you should not free it from memory.

See Also

`slapi_valueset_first_value()`

slapi_valueset_set_from_smod()

Copies the values of `Slapi_Mod` structure into a `Slapi_ValueSet` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_set_from_smod(Slapi_ValueSet *vs,
    Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

Parameter	Description
`vs`	Pointer to the `Slapi_ValueSet` structure into which you wish to copy the values.
`smod`	Pointer to the `Slapi_Mod` structure from which you wish to copy the values.

Description

This function copies all of the values contained in a `Slapi_Mod` structure into a `Slapi_ValueSet` structure.

Memory Concerns

This function does not verify that the `Slapi_ValueSet` structure already contains values, so it is your responsibility to verify that there are no values prior to calling this function. If you do not verify this, the allocated memory space will leak. You can free existing values by calling `slapi_valueset_done()`.

See Also

`slapi_valueset_done()`

slapi_valueset_set_valueset()

Initializes a `Slapi_ValueSet` structure from another `Slapi_ValueSet` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_set_valueset(Slapi_ValueSet *vs1,
    const Slapi_ValueSet *vs2);
```

Parameters

This function takes the following parameters:

Parameter	Description
`vs1`	Pointer to the `Slapi_ValueSet` structure to which you wish to set the values.
`vs2`	Pointer to the `Slapi_ValueSet` structure from which you wish to copy the values.

Description

This function initializes a `Slapi_ValueSet` structure by copying the values contained in another `Slapi_ValueSet` structure.

Memory Concerns

The function does not verify that the `Slapi_ValueSet` structure contains values, so it is your responsibility to verify that there are no values prior to calling this function. If you do not verify this, the allocated memory space will leak. You can free existing values by calling `slapi_valueset_done()`.

See Also

`slapi_valueset_done()`

slapi_vattr_attr_free()

Free a virtual attribute.

Syntax

```
#include "slapi-plugin.h"
void slapi_vattr_attr_free(Slapi_Attr **a, int buffer_flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
a	Attribute to free
buffer_flags	Bitmask of SLAPI_VIRTUALATTRS_RETURNED_POINTERS, SLAPI_VIRTUALATTRS_RETURNED_COPIES, SLAPI_VIRTUALATTRS_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_RETURNED_TYPENAME_ONLY

Description

Use this function to frees a virtual attribute when finished with it.

See Also

slapi_vattr_attrs_free()

slapi_vattr_attrs_free()

Free a list of virtual attributes.

Syntax

```
#include "slapi-plugin.h"
void slapi_vattr_attrs_free(vattr_type_thang **types, int flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
types	List of attributes to free

flags	Bitmask of SLAPI_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_ONLY, SLAPI_VIRTUALATTRS_REQUEST_POINTERS, SLAPI_VIRTUALATTRS_LIST_OPERATIONAL_ATTRS passed as flags to slapi_vattr_list_attrs()

Description

Use this function to frees a list of virtual attributes obtained using slapi_vattr_list_attrs().

See Also

slapi_vattr_attr_free()

slapi_vattr_list_attrs()

slapi_vattr_filter_test()

Test a filter against an entry that may contain virtual attributes.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_filter_test( Slapi_PBlock *pb, Slapi_Entry *e,
    struct slapi_filter *f, int verify_access);
```

Parameters

This function takes the following parameters:

Parameter	Description
pb	Parameter block containing the search request
e	Candidate entry
f	Filter to check
verify_access	1 to verify access to the entry before checking, 0 otherwise

Description

This function checks whether the candidate entry e matches the filter f. It does not support LDAP v3 extensible match filters.

Returns

This functions returns `0` if the filter matches, or if the filter is `NULL`. Otherwise, it returns `-1`.

slapi_vattr_is_registered()

Check whether an attribute may be virtual.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_is_registered(const char *attrtype,
    const char *scopendn);
```

Parameters

This function takes the following parameters:

Parameter	Description
`attrtype`	Attribute type to check
`scopendn`	Base of the scope to check

Description

This function checks whether a virtual attribute service is registered for the attribute type in the scope specified.

The fact that a virtual attribute service is registered for an attribute type does not guarantee that the service can currently provide a value.

Returns

This functions returns `1` if the attribute may be virtual in the scope specified. Otherwise, it returns `0`.

slapi_vattr_list_attrs()

Get a list of the real and virtual attributes for an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_list_attrs(Slapi_Entry *e,
    vattr_type_thang **types, int flags, int *buffer_flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
e	Get attributes for this entry
types	List of attributes set by the server
flags	Bitmask of SLAPI_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_ONLY, SLAPI_VIRTUALATTRS_REQUEST_POINTERS, SLAPI_VIRTUALATTRS_LIST_OPERATIONAL_ATTRS determining what to set in the list
buffer_flags	Bitmask of SLAPI_VIRTUALATTRS_RETURNED_POINTERS, SLAPI_VIRTUALATTRS_RETURNED_COPIES, SLAPI_VIRTUALATTRS_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_RETURNED_TYPENAME_ONLY determining how the virtual attributes should be handled

Description

This function sets types to point to a full list of attributes for the entry e depending on the flags parameter. Use the buffer_flags parameter when freeing the list.

Use slapi_vattr_values_type_thang_get() to access the attributes.

Returns

This functions returns 1 if the attribute may be virtual in the scope specified. Otherwise, it returns 0.

Memory Considerations

When finished with the types list, free it using slapi_vattr_attrs_free().

See Also

vattr_type_thang

slapi_vattr_attrs_free()

slapi_vattr_values_type_thang_get()

slapi_vattr_value_compare()

Compares attribute type and name in a given entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_value_compare( Slapi_Entry *e, char *type,
    Slapi_Value *test_this, int *result, int flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry to be compared.
`type`	Attribute type name.
`test_this`	Value to be tested.
`result`	0 if the compare is true, 1 if the compare is false.
`flags`	Not used. You should pass 0 for this parameter.

Returns

This function returns 0 for success, in which case `result` contains the result of the comparison.

Otherwise, this function returns the following:

- `SLAPI_VIRTUALATTRS_LOOP_DETECTED` (failed to evaluate a vattr).
- `SLAPI_VIRTUAL_NOT_FOUND` (type not recognized by any vattr and not a real attr in entry).
- `ENOMEM` (memory error).

Description

There is no need to call `slapi_vattr_values_free()` after calling this function.

slapi_vattr_values_free()

Frees the valueset and type names.

Syntax

```
#include "slapi-plugin.h"
void slapi_vattr_values_free(Slapi_ValueSet **value,
    char **actual_type_name, int flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
`value`	Valueset to be freed.
`actual_type_name`	List of type names.
`flags`	The buffer flags returned from `slapi_vattr_values_get_ex()`. This contains information that this function needs to determine which objects need to be freed.

Description

This function should be used to free the valueset and type names returned from `slapi_vattr_values_get_ex()`.

See Also

`slapi_vattr_values_get_ex()`

slapi_vattr_values_get_ex()

Returns the values for an attribute type from an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_values_get_ex(Slapi_Entry *e, char *type,
    Slapi_ValueSet*** results, int **type_name_disposition,
    char ***actual_type_name, int flags, int *buffer_flags,
    int *subtype_count);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry from which to get the values.
`type`	Attribute type name.
`results`	Pointer to result set.
`type_name_disposition`	Matching result.

actual_type_name	Type name as found.
flags	Bit mask of options. Valid values are as follows: • SLAPI_REALATTRS_ONLY • SLAPI_VIRTURALATTRS_ONLY • SLAPI_VIRTUALATTRS_REQUEST_POINTERS • SLAPI_VIRTUALATTRS_LIST_OPERATIONAL_ATTRS
buffer_flags	bit mask to be used as input flags for slapi_values_free().
subtype_count	Number of subtypes matched.

Returns

This function returns 0 for success, in which case:

- results contains the current values for type all of the subtypes in e.
- type_name_disposition contains information on how each type was matched. Valid values are
  - SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS.
  - SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_SUBTYPE.
- actual_type_name contains the type name as found.
- buffer_flags contains the bit mask to be used as input flags for slapi_values_free().
- subtype_count contains the number of subtypes matched.

Otherwise, this function returns the following

- SLAPI_VIRTUALATTRS_LOOP_DETECTED (failed to evaluate a vattr).
- SLAPI_VIRTUAL_NOT_FOUND (type not recognized by any vattr and not a real attr in entry).
- ENOMEM (memory error).

Description

This function returns the values for an attribute type from an entry, including the values for any subtypes of the specified attribute type. The routine will return the values of virtual attributes in that entry if requested to do so.

Memory Concerns

`slapi_vattr_values_free()` should be used to free the returned result set and type names, passing the `buffer_flags` value returned from this routine.

See Also

`slapi_vattr_values_free()`

slapi_vattr_values_type_thang_get()

Get values from a list of the real and virtual attributes for an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_values_type_thang_get(Slapi_Entry *e,
    vattr_type_thang *type_thang, Slapi_ValueSet** results,
    int *type_name_disposition, char **actual_type_name,
    int flags, int *buffer_flags);
```

Parameters

This function takes the following parameters:

Parameter	Description
`e`	Entry the attributes belong to
`type_thang`	Real or virtual attribute type
`results`	Values for the attribute, set by the server
`type_name_disposition`	Set by the server to reflect how type name matched; one of `SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS`, `SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_SUBTYPE`, `SLAPI_VIRTUALATTRS_NOT_FOUND` (type matched no real or virtual attribute on the entry), or `SLAPI_VIRTUALATTRS_LOOP_DETECTED` (could not evaluate the virtual attribute)
`actual_type_name`	Set by the server to the actual type name found
`flags`	Bitmask of `SLAPI_REALATTRS_ONLY`, `SLAPI_VIRTUALATTRS_ONLY`, `SLAPI_VIRTUALATTRS_REQUEST_POINTERS`, `SLAPI_VIRTUALATTRS_LIST_OPERATIONAL_ATTRS` applied when obtaining the list using `slapi_vattr_list_attrs()`

buffer_flags	Set by the server to a bitmask of SLAPI_VIRTUALATTRS_RETURNED_POINTERS, SLAPI_VIRTUALATTRS_RETURNED_COPIES, SLAPI_VIRTUALATTRS_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_RETURNED_TYPENAME_ONLY, useful for freeing the list

Description

This function offers optimized access to values of attributes in a list set by slapi_vattr_list_attrs().

Returns

This function returns 0 for success, in which case:

- results contains the current values for type all of the subtypes in e.
- type_name_disposition contains information on how each type was matched. Valid values are
  - SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS.
  - SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_SUBTYPE.
- actual_type_name contains the type name as found.
- buffer_flags contains the bit mask to be used as input flags for slapi_values_free().
- subtype_count contains the number of subtypes matched.

Otherwise, this function returns the following

- SLAPI_VIRTUALATTRS_LOOP_DETECTED (failed to evaluate a vattr).
- SLAPI_VIRTUAL_NOT_FOUND (type not recognized by any vattr and not a real attr in entry).
- ENOMEM (memory error).

See Also

vattr_type_thang

slapi_vattr_list_attrs()

slapi_wait_condvar()

Wait for a change in a conditional variable.

Syntax

```
#include "slapi-plugin.h"
int slapi_wait_condvar(Slapi_CondVar *cvar,
    struct timeval *timeout);
```

Parameters

This function takes the following parameter:

Parameter	Description
`cvar`	Conditional variable on which to wait
`timeout`	`NULL` means block until notified. Otherwise, block until the time is up, then try again to acquire the lock.

Description

This function enables thread synchronization using a wait/notify mechanism.

Returns

This function returns `1` if successful. Otherwise, it returns `NULL`.

Memory Considerations

Call `slapi_notify_condvar()` to notify other threads of a change to the conditional variable.

See Also

`slapi_destroy_condvar()`

`slapi_new_condvar()`

`slapi_notify_condvar()`

Chapter 3

Parameter Block Reference

This chapter describes the parameters available in the parameter block (`Slapi_PBlock`) structure. This chapter lists data types associated with each parameter.

TIP To read parameter values, use `slapi_pblock_get()`. To write parameter values, use `slapi_pblock_set()`. Use of other functions may crash the server.

Refer to the *Sun ONE Directory Server Plug-In API Programming Guide* and the example plug-ins under *ServerRoot*`/plugins/slapd/slapi/examples/` to better grasp how to use these parameters.

This chapter categorizes parameters as follows.

Table 3-1 `Slapi_PBlock` Parameter Categories

Category	Accessible to...[1]
“Access Log,” on page 336	All plug-in functions
“Add,” on page 336	Pre- and post-operation add functions
“Backend Information,” on page 337	All plug-in functions
“Bind,” on page 337	Pre- and post-operation bind functions
“Compare,” on page 338	Pre- and post-operation compare functions
“Connection Information,” on page 338	All plug-in functions
“Delete,” on page 340	Pre- and post-operation delete functions

Table 3-1 `Slapi_PBlock` Parameter Categories

Category	Accessible to...[1]
"Directory Configuration Information," on page 340	All plug-in functions
"Extended Operations," on page 340	Extended operation functions
"Internal Operations," on page 341	All plug-in functions
"Modify," on page 341	Pre- and post-operation modify functions
"Operation Information," on page 342	All plug-in functions
"Plug-In Registration," on page 343	Plug-in initialization functions and specific types as mentioned in this section
"Rename (Modify RDN)," on page 350	Pre- and post-operation modify RDN functions
"Results," on page 351	All plug-in functions
"Search," on page 352	Pre- and post-operation search functions

1. Object plug-ins may register any type of plug-in function and thus access different parameters depending on the operation.

Access Log

The following parameter allows you to configure access log output.

Table 3-2 Access Log Parameters

Parameter ID	Data Type	Description
`SLAPI_OPERATION_NOTES`	`unsigned int`	Flag specifying that unindexed searches be indicated in the access log. Setting this parameter to `SLAPI_OP_NOTE_UNINDEXED` causes the string `Notes=U` to be appended to access log entries reflecting unindexed searches.

Add

The following parameters allow you to access an entry to add to the directory through the parameter block.

Table 3-3 Add Function Parameters

Parameter ID	Data Type	Description
SLAPI_ADD_ENTRY	Slapi_Entry *	Entry to add.
SLAPI_ADD_TARGET	char *	DN of the entry to add.

Backend Information

The following parameters allow you to access information about directory backends through the parameter block.

Table 3-4 Backend Information Parameters

Parameter ID	Data Type	Description
SLAPI_BACKEND	Slapi_Backend *	Backend serving the current operation. May be NULL if no backend is associated with the current operation.
SLAPI_BE_LASTMOD	int	Whether the backend tracks modification time and who made the modification: • 0 if modifications are not tracked • 1 if modifications are tracked
SLAPI_BE_READONLY	int	Value of nsslapd-readonly in the server configuration file: • 0 if the backend is writeable. • 1 if the backend is read-only.
SLAPI_BE_TYPE	char *	Value of nsslapd-database in the server configuration file.
SLAPI_DBSIZE	unsigned int	Size of the backend database in KB.

Bind

The following parameters allow you to access information about the bind operation through the parameter block.

Table 3-5 Bind Function Parameters

Parameter ID	Data Type	Description
SLAPI_BIND_CREDENTIALS	struct berval *	Bind request credentials such as a password or SASL mechanism credentials, depending on the bind method.
SLAPI_BIND_METHOD	int	Authentication method used. • LDAP_AUTH_NONE (anonymous) • LDAP_AUTH_SASL (SASL) • LDAP_AUTH_SIMPLE (password) • LDAP_AUTH_SSL (certificate)
SLAPI_BIND_RET_SASLCREDS	struct berval *	SASL server credentials to send to the client.
SLAPI_BIND_SASLMECHANISM	char *	SASL mechanism used for bind.
SLAPI_BIND_TARGET	char *	DN used to bind.

Compare

The following parameters allow you to access an entry or attribute to use in a comparison through the parameter block.

Table 3-6 Compare Function Parameters

Parameter ID	Data Type	Description
SLAPI_COMPARE_TARGET	char *	DN of the entry to use in the comparison.
SLAPI_COMPARE_TYPE	char *	Attribute type to use in the comparison.
SLAPI_COMPARE_VALUE	struct berval *	Attribute value to use in the comparison.

Connection Information

The following parameters allow you to access information about the client connection through the parameter block.

Table 3-7 Connection Information Parameters

Parameter ID	Data Type	Description
SLAPI_CLIENT_DNS	struct berval *	Fully qualified domain name of the client.
SLAPI_CONN_AUTHMETHOD	char *	Authentication method used. • SLAPD_AUTH_NONE (anonymous) • SLAPD_AUTH_SASL (extensible SASL) • SLAPD_AUTH_SIMPLE (password) • SLAPD_AUTH_SSL (certificate)
SLAPI_CONN_CLIENTNETADDR	PRNetAddr *	IP address of client.[1]
SLAPI_CONN_DN	char *	DN of the user authenticated for the current connection.
SLAPI_CONNECTION	Slapi_Connection *	The current connection.
SLAPI_CONN_ID	int	Identifier for the current connection.
SLAPI_CONN_IS_REPLICATION_SESSION	int	Whether the connection is for replication. • 0 false. • 1 true.
SLAPI_CONN_IS_SSL_SESSION	int	Whether the connection is over SSL. • 0 false. • 1 true.
SLAPI_CONN_SERVERNETADDR	PRNetAddr *	IP address client is connected to.

1. PRNetAddr is an NSPR 4.x data structure.

Delete

The following parameters allow you to access an entry to delete through the parameter block.

Table 3-8 Delete Function Parameters

Parameter ID	Data Type	Description
SLAPI_DELETE_TARGET	char *	DN of the entry to delete.
SLAPI_ORIGINAL_TARGET	char *	Non-normalized DN of the entry to delete.

Directory Configuration Information

The following parameters allow you to access information about configuration of the directory instance through the parameter block.

Table 3-9 Directory Configuration Information Parameters

Parameter ID	Data Type	Description
SLAPI_ARGC	int	Number of command-line arguments passed to the server at startup.
SLAPI_ARGV	char **	Array of command-line arguments passed to the server at startup.
SLAPI_CONFIG_DIRECTORY	char *	File system directory containing configuration files for the instance.
SLAPI_CONFIG_FILENAME	char *	Configuration file used, such as dse.ldif.

Extended Operations

The following parameters allow you to access information about an extended operation through the parameter block.

Table 3-10 Extended Operation Parameters

Parameter ID	Data Type	Description
SLAPI_EXT_OP_REQ_OID	char *	Object identifier (OID) of the extended operation specified in the request.
SLAPI_EXT_OP_REQ_VALUE	struct berval *	Value specified in the request.

Table 3-10 Extended Operation Parameters

Parameter ID	Data Type	Description
SLAPI_EXT_OP_RET_OID	char *	Object identifier (OID) to return to the client.
SLAPI_EXT_OP_RET_VALUE	struct berval *	Value to send to the client.

Internal Operations

The following parameters allow you to access information about internal operations through the parameter block.

Table 3-11 Internal Operation Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTOP_RESULT	int	Result code of internal operation.
SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES	Slapi_Entry **	Array of entries found by internal search.
SLAPI_PLUGIN_INTOP_SEARCH_REFERRALS	char **	Array of referrals found by internal search in LDAP URL format.

Modify

The following parameters allow you to access an entry to modify through the parameter block.

Table 3-12 Modify Function Parameters

Parameter ID	Data Type	Description
SLAPI_MODIFY_MODS	LDAPMod **	NULL-terminated array of LDAPMod structures containing modifications to perform on the target entry.
SLAPI_MODIFY_TARGET	char *	DN of the entry to modify.
SLAPI_ORIGINAL_TARGET	char *	Non-normalized DN of the entry to modify.

Operation Information

The following parameters allow you to access information about the current operation through the parameter block.

Table 3-13 Operation Information Parameters

Parameter ID	Data Type	Description
SLAPI_CONTROLS_ARG	LDAPControl **	Array of controls passed before the operation is created.
SLAPI_IS_INTERNAL_OPERATION	int	Whether the operation originated internally, or as the result of a client request. • 0 client request. • 1 internal operation.
SLAPI_IS_REPLICATED_OPERATION	int	Whether the operation is part of replication with another server. • 0 false. • 1 true.
SLAPI_OPERATION	Slapi_Operation *	Operation currently in progress.
SLAPI_OPERATION_ID	int	Identifier for the operation.
SLAPI_OPERATION_MSGID	long	Message identifier for the operation.
SLAPI_OPINITIATED_TIME	time_t	Time when the server began processing the operation.
SLAPI_REQCONTROLS	LDAPControl **	Array of controls specified in the request.
SLAPI_REQUESTOR_DN	char *	DN of the user requesting the operation.
SLAPI_REQUESTOR_IS_ROOT	int	Whether the bind DN of the user requesting the operation corresponds to the root DN, the value of nsslapd-rootdn on cn=config for the instance. • 0 false. • 1 true.
SLAPI_TARGET_DN	char *	DN to which the operation applies.

Plug-In Registration

The following parameters are for use when registering plug-ins and their functions with the server and when accessing information about plug-in type and identity.

The following table lists information accessible to all types of plug-ins.

Table 3-14 Plug-In Information Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN	void *	Internal server representation of the plug-in.
SLAPI_PLUGIN_ARGC	int	Number of arguments in the configuration entry.
SLAPI_PLUGIN_ARGV	char *	Array of arguments in the configuration entry.
SLAPI_PLUGIN_IDENTITY	void *	Plug-in identifier set and then required by the server when handling internal operations. You may cast this to Slapi_ComponentId.
SLAPI_PLUGIN_PRIVATE	void *	Private data you pass to your plug-in functions. You define this data structure.

Table 3-14 Plug-In Information Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_TYPE	int	Type of plug-in function, corresponding to the value of nsslapd-pluginType in the configuration entry for the plug-in. (Use the plug-in type values given here between parentheses in configuration entries.) • SLAPI_PLUGIN_EXTENDEDOP (extendedop) • SLAPI_PLUGIN_INTERNAL_POSTOPERATION (internalpostoperation) • SLAPI_PLUGIN_INTERNAL_PREOPERATION (internalpreoperation) • SLAPI_PLUGIN_LDBM_ENTRY_FETCH_STORE (ldbmentryfetchstore) • SLAPI_PLUGIN_MATCHINGRULE (matchingrule) • SLAPI_PLUGIN_TYPE_OBJECT (object)[1] • SLAPI_PLUGIN_POSTOPERATION (postoperation) • SLAPI_PLUGIN_PREOPERATION (preoperation) • SLAPI_PLUGIN_PWD_STORAGE_SCHEME (pwdstoragescheme) • SLAPI_PLUGIN_REVER_PWD_STORAGE_SCHEME (reverpwdstoragescheme)
SLAPI_PLUGIN_VERSION	char *	Plug-in API version supported by the plug-in. • SLAPI_PLUGIN_CURRENT_VERSION (presently SLAPI_PLUGIN_VERSION_03) • SLAPI_PLUGIN_VERSION_01 (3.x and later servers) • SLAPI_PLUGIN_VERSION_02 (4.x and later servers) • SLAPI_PLUGIN_VERSION_03 (5.x servers)

1. Object plug-ins may register any type of plug-in function.

Startup and Shutdown

Parameters for registering generic plug-in functions follow. These function types may be registered by any plug-in.

Table 3-15 Generic Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_CLOSE_FN	void *	Function called at server shutdown.
SLAPI_PLUGIN_START_FN	void *	Function called at server startup. May be called more than once.

Extended Operations

Parameters for registering an extendedop plug-in function and object identifier list follow.

Table 3-16 Extended Operation Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_EXT_OP_FN	void *	Function called upon request for an LDAP v3 extended operation.
SLAPI_PLUGIN_EXT_OP_OIDLIST	char **	NULL-terminated list of object identifiers (OIDs) handled by the plug-in.

Internal Postoperation

Parameters for registering an internalpostoperation plug-in functions follow.

Table 3-17 Internal Postoperation Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTERNAL_POST_ADD_FN	void *	Function called after an internal add operation completes.
SLAPI_PLUGIN_INTERNAL_POST_DELETE_FN	void *	Function called after an internal delete operation completes.
SLAPI_PLUGIN_INTERNAL_POST_MODIFY_FN	void *	Function called after an internal modify operation completes.

Table 3-17 Internal Postoperation Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTERNAL_POST_MODRDN_FN	void *	Function called after an internal rename (modify RDN) operation completes.

Internal Preoperation

Parameters for registering an `internalpreoperation` plug-in functions follow.

Table 3-18 Internal Preoperation Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTERNAL_PRE_ADD_FN	void *	Function called before an internal add operation is performed.
SLAPI_PLUGIN_INTERNAL_PRE_DELETE_FN	void *	Function called before an internal delete operation is performed.
SLAPI_PLUGIN_INTERNAL_PRE_MODIFY_FN	void *	Function called before an internal modify operation is performed.
SLAPI_PLUGIN_INTERNAL_PRE_MODRDN_FN	void *	Function called before an internal rename (modify RDN) operation is performed.

Entry Storage and Retrieval

Parameters for registering `ldbmentryfetchstore` plug-in functions follow.

Table 3-19 Entry Store / Fetch Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_ENTRY_FETCH_FUNC	void *	Function called upon retrieval of an entry from the backend.
SLAPI_PLUGIN_ENTRY_STORE_FUNC	void *	Function called before storing an entry through the backend.

Matching Rules

Parameters for registering `matchingrule` plug-in functions and arguments follow.

Table 3-20 Matching Rule Function and Argument Registration Parameters

Parameter ID	Data Type	Description
SLAPI_MATCHINGRULE_DESC	-	Used to signify registration of the matching rule description with `slapi_matchingrule_set()`.
SLAPI_MATCHINGRULE_NAME	-	Used to signify registration of the matching rule name with `slapi_matchingrule_set()`.
SLAPI_MATCHINGRULE_OBSOLETE	-	Used to signify the matching rule is obsolete when registering with `slapi_matchingrule_set()`.
SLAPI_MATCHINGRULE_OID	-	Used to signify registration of the matching rule object identifier with `slapi_matchingrule_set()`.
SLAPI_MATCHINGRULE_SYNTAX	-	Used to signify registration of the matching rule syntax with `slapi_matchingrule_set()`.
SLAPI_PLUGIN_DESTROY_FN	`void *`	Function called to free memory allocated to filter object.
SLAPI_PLUGIN_MR_FILTER_CREATE_FN	`void *`	Filter factory function.
SLAPI_PLUGIN_MR_FILTER_INDEX_FN	`void *`	Function called to set the indexer function.
SLAPI_PLUGIN_MR_FILTER_MATCH_FN	`void *`	Function called to check for a match.
SLAPI_PLUGIN_MR_FILTER_RESET_FN	`void *`	Function called to reset the match filter.
SLAPI_PLUGIN_MR_FILTER_REUSABLE	`unsigned int`	Whether the filter is reusable. • `0` false. • `1` true.
SLAPI_PLUGIN_MR_INDEXER_CREATE_FN	`void *`	Index factory function.
SLAPI_PLUGIN_MR_INDEX_FN	`void *`	Function called to index a single entry.
SLAPI_PLUGIN_MR_KEYS	`struct berval **`	Array of index keys corresponding to the attribute values.

Table 3-20 Matching Rule Function and Argument Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_MR_OID	char *	Object identifier (OID) corresponding to the extensible match rule.
SLAPI_PLUGIN_MR_QUERY_OPERATOR	int	Type of operator used to check for matches.
		SLAPI_OP_EQUAL
		SLAPI_OP_GREATER
		SLAPI_OP_GREATER_OR_EQUAL
		SLAPI_OP_LESS
		SLAPI_OP_LESS_OR_EQUAL
		SLAPI_OP_SUBSTRING
SLAPI_PLUGIN_MR_TYPE	char *	Matching rule filter type.
SLAPI_PLUGIN_MR_USAGE	unsigned int	Whether to use the rule to index or to sort.
		SLAPI_PLUGIN_MR_USAGE_INDEX
		SLAPI_PLUGIN_MR_USAGE_SORT
SLAPI_PLUGIN_MR_VALUE	struct berval *	Attribute value to match.
SLAPI_PLUGIN_MR_VALUES	struct berval **	Array of attribute values to match.
SLAPI_PLUGIN_OBJECT	void *	Filter object for extensible match.
		You define this data structure to use in a matching rule plug-in.

Postoperation

Parameters for registering postoperation plug-in functions follow.

Table 3-21 Postoperation Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_POST_ABANDON_FN	void *	Function called after an operation is abandoned.
SLAPI_PLUGIN_POST_ADD_FN	void *	Function called after an add operation completes.
SLAPI_PLUGIN_POST_BIND_FN	void *	Function called after a bind operation completes.

Table 3-21 Postoperation Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_POST_COMPARE_FN	void *	Function called after a compare operation completes.
SLAPI_PLUGIN_POST_DELETE_FN	void *	Function called after a delete operation completes.
SLAPI_PLUGIN_POST_ENTRY_FN	void *	Function called after an entry is sent to the client.
SLAPI_PLUGIN_POST_MODIFY_FN	void *	Function called after a modify operation completes.
SLAPI_PLUGIN_POST_MODRDN_FN	void *	Function called after an rename (modify RDN) operation completes.
SLAPI_PLUGIN_POST_REFERRAL_FN	void *	Function called after a referral is sent to the client.
SLAPI_PLUGIN_POST_RESULT_FN	void *	Function called after a result is sent to the client.
SLAPI_PLUGIN_POST_SEARCH_FN	void *	Function called after a search operation completes. For persistent search operations, this function is called after the client interrupts the search.
SLAPI_PLUGIN_POST_UNBIND_FN	void *	Function called after an unbind operation completes.

Preoperation

Parameters for registering `preoperation` plug-in functions follow.

Table 3-22 Preoperation Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_PRE_ABANDON_FN	void *	Function called before an operation is abandoned.
SLAPI_PLUGIN_PRE_ADD_FN	void *	Function called before an add operation is performed.
SLAPI_PLUGIN_PRE_BIND_FN	void *	Function called before a bind operation is performed.
SLAPI_PLUGIN_PRE_COMPARE_FN	void *	Function called before a compare operation is performed.
SLAPI_PLUGIN_PRE_DELETE_FN	void *	Function called before an delete operation is performed.
SLAPI_PLUGIN_PRE_ENTRY_FN	void *	Function called before an entry is sent to the client.
SLAPI_PLUGIN_PRE_MODIFY_FN	void *	Function called before a modify operation is performed.

Table 3-22 Preoperation Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_PRE_MODRDN_FN	void *	Function called before an rename (modify RDN) operation is performed.
SLAPI_PLUGIN_PRE_REFERRAL_FN	void *	Function called before a referral is sent to the client.
SLAPI_PLUGIN_PRE_RESULT_FN	void *	Function called before a result is sent to the client.
SLAPI_PLUGIN_PRE_SEARCH_FN	void *	Function called before a search operation is performed.
SLAPI_PLUGIN_PRE_UNBIND_FN	void *	Function called before an unbind operation is performed.

One-Way and Reversible Password Storage

Parameters for registering `pwdstorage` and `reverpwdstorage` plug-in functions follow.

Table 3-23 Password Storage Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_CMP_FN	void *	Function called to encode a password for comparison with a stored, encoded password.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DB_PWD	char *	Stored, encoded user password.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DEC_FN	void *	(`reverpwdstorage` plug-ins only) Function called to decode an encrypted password.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_ENC_FN	void *	Function called to encode a password for storage.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_NAME	char *	Short password storage scheme name used by the server to identify the encoding scheme.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_USER_PWD	char *	User password in clear text.

Rename (Modify RDN)

The following parameters allow you to access an entry to rename through the parameter block.

Table 3-24 Rename (Modify RDN) Function Parameters

Parameter ID	Data Type	Description
SLAPI_MODRDN_DELOLDRDN	int	Whether to delete the old Relative Distinguished Name (RDN). • 0 false. • 1 true.
SLAPI_MODRDN_NEWRDN	char *	New RDN to assign to the entry.
SLAPI_MODRDN_NEWSUPERIOR	char *	DN of the new parent of the entry being renamed.
SLAPI_MODRDN_TARGET	char *	DN of the entry to rename.
SLAPI_ORIGINAL_TARGET	char *	Non-normalized DN of the entry to rename.

Results

The following parameters allow you to access results through the parameter block.

Table 3-25 Result Parameters

Parameter ID	Data Type	Description
SLAPI_ADD_RESCONTROL	LDAPControl *	Lets you add a control to the set of controls to send to the client. Use slapi_pblock_set() to add a control with this parameter.
SLAPI_RES_CONTROLS	LDAPControl **	Array of controls to send to client. If you use this with slapi_pblock_set() to change the set of controls to send to the client, you must retrieve and free the existing set of controls pointed to by SLAPI_RES_CONTROLS.
SLAPI_RESULT_CODE	int	Result code to send to client.
SLAPI_RESULT_MATCHED	char *	Portion of target DN that matched when sending LDAP_NO_SUCH_OBJECT to the client.
SLAPI_RESULT_TEXT	char *	Message to send to client.

Search

The following parameters allow you to access search parameters through the parameter block.

Table 3-26 Search Function Parameters

Parameter ID	Data Type	Description
SLAPI_NENTRIES	int	Number of entries returned by the search.
SLAPI_SEARCH_ATTRS	char **	Array of attribute types to return in the search results. The asterisk, *, can be used to mean all real (non-virtual) attributes.
SLAPI_SEARCH_ATTRSONLY	int	Whether to return both attribute types and attribute values. • 0 return both. • 1 return only types.
SLAPI_SEARCH_DEREF	int	Method for handling aliases. • LDAP_DEREF_ALWAYS • LDAP_DEREF_FINDING • LDAP_DEREF_NEVER • LDAP_DEREF_SEARCHING
SLAPI_SEARCH_FILTER	Slapi_Filter *	Filter to be used for the search.
SLAPI_SEARCH_REFERRALS	struct berval **	Array of URLs to other LDAP servers to which the client is referred.
SLAPI_SEARCH_RESULT_ENTRY	void *	Entry returned while iterating through the result set. You may cast this to Slapi_Entry.
SLAPI_SEARCH_SCOPE	int	Scope of the search. • LDAP_SCOPE_BASE • LDAP_SCOPE_ONELEVEL • LDAP_SCOPE_SUBTREE
SLAPI_SEARCH_SIZELIMIT	int	Maximum number of entries to return in the search results.
SLAPI_SEARCH_STRFILTER	char *	String representation of the filter to be used for the search.

Table 3-26 Search Function Parameters

Parameter ID	Data Type	Description
SLAPI_SEARCH_TARGET	char *	DN of base entry for the search.
SLAPI_SEARCH_TIMELIMIT	int	Maximum number of seconds to allow for the search.

Index

R

S

V