z/OS

IBM Tivoli Directory Server Plug-in Reference for z/OS

SA76-0148-00

z/OS

IBM Tivoli Directory Server Plug-in Reference for z/OS

SA76-0148-00

Note
Before using this information and the product it supports, be sure to read the general information under Notices.

First Edition, September 2008

This edition applies to Version 1 Release 10 of z/OS (5694-A01). This edition, SA76-0148-00, applies to Version 1 Release 10 of z/OS (5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/systems/z/os/zos/webqs.html

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:
- Title and order number of this document
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Tables v

About this document vii
Who should use this document vii
Conventions used in this document vii
Where to find more information vii
Softcopy publications vii
z/OS online library vii

Chapter 1. Introduction to server plug-ins 1

Chapter 2. Building an LDAP server plug-in 3
Steps for writing an IBM TDS for z/OS plug-in 3

Chapter 3. Operation plug-ins 5
Pre-operation plug-ins 5
Post-operation plug-ins 5
Client-operation plug-ins 5

Chapter 4. Plug-in application service routines 9
slapi_add_internal() 10
slapi_attr_get_normalized_values() 11
slapi_attr_get_numvalues() 12
slapi_attr_get_type() 13
slapi_attr_get_values() 14
slapi_attr_value_cmp() 15
slapi_ch_calloc() 16
slapi_ch_free() 17
slapi_ch_free_values() 18
slapi_ch_malloc() 19
slapi_ch_realloc() 20
slapi_ch_strdup() 21
slapi_compare_internal() 22
slapi_control_present() 23
slapi_delete_internal() 24
slapi_dn_ignore_case_v3() 25
slapi_dn_isparent() 27
slapi_dn_normalize_v3() 28
slapi_dn_normalize_case_v3() 29
slapi_entry_add_value() 31
slapi_entry_add_values() 32
slapi_entry_alloc() 33
slapi_entry_attr_delete() 34
slapi_entry_attr_find() 35
slapi_entry_delete_value() 36
slapi_entry_delete_values() 37
slapi_entry_dup() 38
slapi_entry_first_attr() 39
slapi_entry_free() 40
slapi_entry_get_dn() 41
slapi_entry_merge_value() 42
slapi_entry_merge_values() 43
slapi_entry_next_attr() 44

slapi_entry_replace_value() . . . 45
slapi_entry_replace_values() . . . 46
slapi_entry_schema_check() . . . 47
slapi_entry_set_dn() . . . 48
slapi_filter_get_attribute_type() . . . 49
slapi_filter_get_ava() . . . 50
slapi_filter_get_choice() . . . 51
slapi_filter_get_subfilt() . . . 52
slapi_filter_list_first() . . . 53
slapi_filter_list_next() . . . 54
slapi_isSDBM_authenticated() . . . 55
slapi_log_error() . . . 56
slapi_modify_internal() . . . 58
slapi_modrdn_internal() . . . 59
slapi_op_abandoned() . . . 60
slapi_pblock_destroy() . . . 61
slapi_pblock_get() . . . 62
slapi_pblock_set() . . . 69
slapi_search_internal() . . . 72
slapi_send_ldap_referral() . . . 74
slapi_send_ldap_result() . . . 76
slapi_send_ldap_search_entry() . . . 77
slapi_trace() . . . 78

Appendix A. Plug-in sample . . . 81
Steps for building and running a sample plug-in . . . 81

Appendix B. Accessibility . . . 83
Using assistive technologies . . . 83
Keyboard navigation of the user interface . . . 83
z/OS information . . . 83

Notices . . . 85
Programming interface information . . . 86
Policy for unsupported hardware . . . 86
Trademarks . . . 86

Bibliography . . . 87

Index . . . 89

Tables

1. **slapi_filter_get_choice()** search filters . . . 51
2. printf()-style substitution codes . . . 56
3. Operational parameters . . . 62
4. General request parameters . . . 63
5. ABANDON request parameters . . . 64
6. ADD request parameters . . . 64
7. BIND request parameters . . . 64
8. COMPARE request parameters . . . 65
9. DELETE request parameters . . . 65
10. EXTENDED OPERATION request parameters . . . 65
11. MODIFY request parameters . . . 65
12. MODIFY DN request parameters . . . 65
13. SEARCH request parameters . . . 66
14. Callback parameters . . . 66
15. General result parameters . . . 67
16. Internal request result parameters . . . 67
17. Registration parameters . . . 69
18. Operational parameters . . . 70
19. Callback parameters . . . 70
20. General result parameters . . . 71
21. EXTENDED OPERATION result parameters . . . 71
22. printf()-style substitution codes . . . 79

About this document

The IBM® Tivoli® Directory Server for z/OS® is the IBM implementation of the Lightweight Directory Access Protocol (LDAP) for the z/OS operating system.

This document contains reference information about using and writing plug-ins, which extend the capabilities of the IBM Tivoli Directory Server for z/OS (5694-A01).

Who should use this document

This document is intended for application programmers. Application programmers should be experienced and have previous knowledge of directory services.

Conventions used in this document

This document uses the following typographic conventions:

Bold	**Bold** words or characters represent API names, functions, routines, utility names, and system elements that you must enter into the system literally, such as commands and options.
Italic	*Italic* words or characters represent variables for which you must supply values.
`Example font`	Path names, attributes, environment variables, parameter values, examples, and information displayed by the system appear in `constant width type style`.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices.
...	Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.
\	A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character \ as the last non-blank character on the line to be continued, and continue the command on the next line.

Where to find more information

Where necessary, this document references information in other documents. For complete titles and order numbers for all elements of z/OS, see *z/OS Information Roadmap*, SA22-7500.

Softcopy publications

The IBM Tivoli Directory Server and LDAP libraries are available on a CD-ROM collection, *z/OS Collection*. The CD-ROM online library collections include Softcopy Reader, which is a program that enables you to view the softcopy documents.

z/OS online library

The softcopy z/OS publications are also available for Web browsing and for viewing or printing PDFs using the following URL:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv`

You can also provide comments about this document and any other z/OS documentation by visiting that URL. Your feedback is important in helping to provide the most accurate and high-quality information.

Chapter 1. Introduction to server plug-ins

This document explains how to create an IBM Tivoli Directory Server for z/OS plug-in. In general, a plug-in is a software module that adds function to an existing program or application. In this case, configured plug-ins extend the capabilities of your directory server.

Plug-ins are dynamically loaded into the LDAP server's address space when the server is started. When the plug-in is loaded, a plug-in initialization routine is called to register plug-in functions. The server calls plug-in functions from the dynamically loaded library by using registered function pointers.

When the LDAP server receives a client request, the server attempts to call a configured database backend function to process the request. If a database backend is found that accepts the client request, that backend processes the request. LDAP server backends typically process client requests by reading or writing data to a database containing directory entries. In addition to these types of database operations, LDAP server backends may also provide functions that support replication and dynamic schema updates.

If a client request is not accepted by a database backend, then the LDAP server attempts to call a configured plug-in to process the request. If a plug-in is found, which accepts the request, that plug-in processes the request.

Once the request is processed by a configured database backend or plug-in, that backend or plug-in must return a message to the client. If the client request is not processed, the LDAP server returns an error message to the client. Only one message is returned to the client.

The following types of plug-ins are supported by the IBM Tivoli Directory Server for z/OS: (See Chapter 3, "Operation plug-ins," on page 5 for more information.)

pre-operation
: a plug-in that is executed before a client request is processed. For example, a plug-in that checks for a new entry, before the new entry is added to a directory

post-operation
: a plug-in that is executed after a client request is processed. For example, a plug-in that audits clients after they bind to the server

client-operation
: a plug-in that is called to process a client request

Chapter 2. Building an LDAP server plug-in

Each plug-in is a separate dynamic link library (DLL) that is loaded by the LDAP server. The **slapi-plugin.h** include file defines the various structures and service routine prototypes that are available to the plug-in.

LDAP server SLAPI export definitions are contained in one of two DLL library load modules:

- The GLDSLP31.x side file contains the export definitions that a 31-bit plug-in DLL imports.
- The GLDSLP64.x side file contains the export definitions that a 64-bit plug-in DLL imports.

The plug-in must be stored as a member of a PDS or PDSE (a 64-bit plug-in DLL must be stored in a PDSE). The plug-in data set must be in the load list for the LDAP server, either through a STEPLIB statement or the system LNKLST.

The LDAP server **plugin** configuration option is used to define a plug-in, and must be added to the LDAP server configuration file. This option is described in *IBM Tivoli Directory Server Administration and Use for z/OS*, *Chapter 8, Customizing the LDAP server configuration*. It has three required parameters and one optional parameter:

1. the plug-in type - preOperation, clientOperation or postOperation
2. the plug-in DLL name
3. the name of the plug-in initialization routine, which will be called during LDAP server initialization
4. optional parameters which the plug-in can retrieve

For example:

```
plugin postOperation PLUGSAMP plugin_init "auditFile"
```

Steps for writing an IBM TDS for z/OS plug-in

How to build an IBM TDS for z/OS plug-in:

- Start by designing and writing the plug-in initialization routine and SLAPI service functions

  The plug-in initialization routine must register the following that are supported by the plug-in:
  service functions
  message types
  distinguished name suffixes
  extended operation object identifiers

  Return code 0 must be returned when successful and non-zero when not successful. The plug-in initialization routine receives as input, the plug-in parameter block (**Slapi_PBlock**) and returns an integer as the return value. An example of an initialization routine prototype:

  ```
  int plugin_init ( Slapi_PBlock * pb );
  ```

  Note: For this example, the name `plugin_init` would be the initialization routine name used with the **plugin** configuration option.
- When writing the SLAPI service functions that implement the plug-in design, see Chapter 4, “Plug-in application service routines,” on page 9 for application service routines to use and for defined prototypes. You can also see **slapi-plugin.h** for defined prototypes.
- Decide on any input parameters for the plug-in

  Plug-in input parameters can be retrieved using the SLAPI_PLUGIN_ARGC or SLAPI_PLUGIN_ARGV parameters with the **slapi_pblock_get()** service routine.
- Include **slapi-plugin.h**, which contains defined SLAPI data structures and prototypes
- Export the plug-in initialization routine

- Compile the plug-in code into object files
- Link the plug-in object files with one of the LDAP server SLAPI side files listed above
- Ensure the plug-in DLL module is in the load list of the LDAP server and is a member of either a PDS or PDSE
- APF authorize the data set that contains the plug-in DLL
- Edit and add the **plugin** configuration option to the LDAP server configuration file. See *IBM Tivoli Directory Server Administration and Use for z/OS* for more information about the configuration option.
- Restart the LDAP server

You may want to program trace statements to follow processing flow in the plug-in. The trace macro, **SLAPI_TRACE()**, is provided in **slapi-plugin.h** to assist in tracing. This macro uses the **slapi_trace()** service routine, described in Chapter 4, “Plug-in application service routines,” on page 9. For example:

```
SLAPI_TRACE((LDAP_DEBUG_PLUGIN, "PLUGSAMP", "Entered."));
```

A sample plug-in showing several examples of using SLAPI service routines and a makefile are provided in Appendix A, “Plug-in sample,” on page 81:

- **/usr/lpp/ldap/examples/plug-insample.c** is the sample plug-in
- **/usr/lpp/ldap/examples/makefile.plugin** is its makefile

Chapter 3. Operation plug-ins

The IBM Tivoli Directory Server for z/OS supports the following operational plug-ins:

- Pre-operation
- Post-operation
- Client-operation

Pre-operation plug-ins

A pre-operation plug-in is executed before a client request is processed.

The plug-in initialization function is responsible for registering the message types supported by the plug-in by calling the **slapi_pblock_set()** routine. The plug-in will not be called for a message type that it has not registered.

The pre-operation message function receives the plug-in parameter block, (**Slapi_PBlock**), as an input parameter and returns an integer as the function return value:

```
int plug-in_message_function (
  Slapi_PBlock * pb);
```

The return value is zero if request processing continues and nonero if request processing terminates. If a non-zero value is returned, the pre-operation plug-in must return a result message to the client by calling the **slapi_send_ldap_result()** routine. If a zero value is returned, the pre-operation plug-in must not return a result to the client. A result message is not returned for ABANDON and UNBIND requests and the plug-in return value is ignored for these message types.

Note: Post-operation plug-ins are called even if a nonzero value is returned by the pre-operation plug-in.

Post-operation plug-ins

A post-operation plug-in is executed after a client request is processed.

The plug-in initialization function is responsible for registering the message types supported by the plug-in by calling the **slapi_pblock_set()** routine. The plug-in will not be called for a message type that it has not registered.

A post-operation message function receives the plug-in parameter block, (**Slapi_PBlock**), as an input parameter. There is no function return value.

```
void plug-in_message_function (
  Slapi_PBlock * pb);
```

The plug-in must not return a result message to the client since this has already been done before the post-operation plug-in is called. The **slapi_pblock_get()** routine is called to obtain the result code returned to the client for the request.

Client-operation plug-ins

A client-operation plug-in is executed after a client request is processed. For ADD, BIND, COMPARE, DELETE, MODIFY, MODIFY DN and SEARCH requests, the plug-in is called if it registered a suffix that matches the target DN for the request. For EXTENDED OPERATION requests, the plug-in is called if it registered an object identifier that matches the object identifier in the request. All client-operation plug-ins are called for ABANDON and UNBIND requests.

The client-operation plug-in initialization function is responsible for registering the message types, distinguished name suffixes and extended operations supported by the plug-in by calling the **slapi_pblock_set()** routine. The plug-in is only called for message types or extended operations that it has registered for.

The client-operation message function receives the plug-in parameter block (**Slapi_PBlock**) as an input parameter. There is no function return value.

```
void plug-in_message_function (
  Slapi_PBlock * pb);
```

The client operation plug-in must return a result message to the client for all message types except ABANDON and UNBIND (these message types do not return a response to the client). The **slapi_send_ldap_result()** routine is used to send the result message to the client. For a SEARCH request, the **slapi_send_ldap_search_entry()** and **slapi_send_ldap_referral()** routines are used to send the search results to the client before sending the result message.

Additional server controls are registered with the LDAP server by specifying SLAPI_PLUGIN_CTLLIST when calling the **slapi_pblock_set()** routine. Server control registration is only permitted during plug-in initialization. At any time, a plug-in can retrieve the list of server controls registered by specifying SLAPI_PLUGIN_CTLLIST when calling the **slapi_pblock_get()** routine.

The plug-in can access the server controls supplied with a client request by specifying SLAPI_REQCONTROLS when calling the **slapi_pblock_get()** routine. The plug-in can also set a list of server controls to be returned in the client result message by specifying SLAPI_RETCONTROLS when calling the **slapi_pblock_set()** routine.

In addition to client requests, the client-operation plug-in can also register a callback routine. The callback routine is called by the LDAP server when the server needs additional information. The plug-in calls the **slapi_pblock_get()** routine for the SLAPI_CALLBACK_TYPE parameter to get the callback type. Some examples of callbacks are:

Get the user password
Get the group list
Get the alternate names

ABANDON
: Each client-operation plug-in is called for an ABANDON request if the plug-in has registered a **SLAPI_PLUGIN_ABANDON_FN** routine. The plug-in must not return a response to the client since there is no client response for an ABANDON request. The plug-in stops processing a request that is abandoned by the client.

  Instead of registering a **SLAPI_PLUGIN_ABANDON_FN** routine, the plug-in can periodically call the **slapi_op_abandoned()** routine to see if an active request is abandoned by the client.

ADD The client-operation plug-in is called for an ADD request if the entry DN matches a suffix registered by the plug-in and the plug-in registered a **SLAPI_PLUGIN_ADD_FN** routine. The plug-in is responsible for processing the request and returning the result message to the client.

BIND The client-operation plug-in is called for a simple BIND if the authentication DN matches a suffix registered by the plug-in and the plug-in registered a **SLAPI_PLUGIN_BIND_FN** routine. A SASL BIND is not passed to the plug-in. The plug-in is responsible for authenticating the DN and returning the result message to the client. Extended group gathering is performed for an authentication DN located in a plug-in database but plug-in databases are not included in the group gathering process.

COMPARE
: The client-operation plug-in is called for a COMPARE request if the entry DN matches a suffix

registered by the plug-in and the plug-in registered a **SLAPI_PLUGIN_COMPARE_FN** routine. The plug-in is responsible for processing the request and returning the result message to the client.

DELETE

The client-operation plug-in is called for a DELETE request if the entry DN matches a suffix registered by the plug-in and the plug-in registered a **SLAPI_PLUGIN_DELETE_FN** routine. The plug-in is responsible for processing the request and returning the result message to the client.

EXTENDED OPERATION

The client-operation plug-in is called for an EXTENDED OPERATION request if the request object identifier matches an object identifier registered by the plug-in and the plug-in registered a **SLAPI_PLUGIN_EXT_OP_FN** routine. The plug-in is responsible for processing the extended operation request and returning the result to the client. The **slapi_pblock_set()** routine is used to set the extended operation result object identifier (SLAPI_EXT_OP_RET_OID) and value (SLAPI_EXT_OP_RET_VALUE) in the result message. The **slapi_send_ldap_result()** routine is then used to return the result to the client.

MODIFY

The client-operation plug-in is called for a MODIFY request if the entry DN matches a suffix registered by the plug-in and the plug-in registered a **SLAPI_PLUGIN_MODIFY_FN** routine. The plug-in is responsible for processing the request and returning the result message to the client.

MODIFY DN

The client-operation plug-in is called for a MODIFY DN request if the entry DN matches a suffix registered by the plug-in and the plug-in registered a **SLAPI_PLUGIN_MODRDN_FN** routine. The plug-in is responsible for processing the request and returning the result message to the client.

SEARCH

The client-operation plug-in is called for a SEARCH request if the base DN matches a suffix registered by the plug-in and the plug-in registered a **SLAPI_PLUGIN_SEARCH_FN** routine. The plug-in is responsible for processing the request and returning the result message to the client. Search entries are returned by calling the **slapi_send_ldap_search_entry()** routine, search referrals are returned by calling the **slapi_send_ldap_referral()** routine, and the search result is returned by calling the **slapi_send_ldap_result()** routine.

UNBIND

Each client-operation plug-in is called for an UNBIND request if the plug-in registered a **SLAPI_PLUGIN_UNBIND_FN** routine. The plug-in must not return a response to the client since there is no client response for an UNBIND request. The plug-in does not release any resources that are allocated for the connection.

Chapter 4. Plug-in application service routines

This topic describes the plug-in application service routines. The **slapi-plugin.h** include file defines the data structures and function prototypes. The **GLDSLP31.x** and **GLDSLP64.x** side files provide the DLL import definitions for 31-bit and 64-bit load modules.

Text data is represented in UTF-8 format. The application is responsible for any necessary code page conversions.

The service routines assume that the directory objects (entries, attributes and filters) are used by a single thread. The application is responsible for providing concurrency control if it is sharing directory objects among multiple threads.

slapi_add_internal()

Purpose

Issue an ADD entry request.

Format

```
#include <slapi-plugin.h>

Slapi_PBlock * slapi_add_internal (
  const char *            dn,
  LDAPMod **              mods,
  LDAPControl **          controls,
  int                     l)
```

Parameters

Input

dn The distinguished name of the new entry.

mods The *mod_op* field is ignored other than checking the LDAP_MOD_BVALUES flag. The attribute value is specified as a BerVal structure if the LDAP_MOD_BVALUES flag is set and is specified as a character string if it is not set.

controls A NULL-terminated array of server controls for the ADD request. Specify NULL if there are no server controls.

l This parameter is not used and should be set to 0. It is included for compatibility with other LDAP implementations.

Usage

The **slapi_add_internal()** routine issues an ADD request and returns the results to the plug-in for processing. The LDAP Version 3 protocol and the current client authentication is used for the ADD request. The request is unauthenticated if a client request is not being processed. You should call the **slapi_pblock_get()** routine to obtain the results from the returned parameter block. The following values can be retrieved from the parameter block:

- SLAPI_PLUGIN_INTOP_RESULT - The result code from the result message
- SLAPI_PLUGIN_INTOP_ERRMSG - The error message from the result message
- SLAPI_PLUGIN_INTOP_MATCHED_DN - The matched DN from the result message
- SLAPI_PLUGIN_INTOP_REFERRALS - The referrals from the result message

Related topics

The function return value is the address of a plug-in parameter block or NULL if the ADD request is not issued. The **slapi_pblock_destroy()** routine is to release the plug-in parameter block when it is no longer needed. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
EIO	Unable to process the ADD request
ENOMEM	Insufficient storage is available

slapi_attr_get_normalized_values()

Purpose

Obtain the normalized attribute values.

Format

```
#include <slapi-plugin.h>

int slapi_attr_get_normalized_values (
 Slapi_Attr *       attr,
 BerVal ***         vals)
```

Parameters

Input

attr The directory entry attribute.

Output

vals This variable sets the address of the normalized attribute value array or the NULL if there are no attribute values. The end of the array is indicated by a NULL value address. The application must not modify or release the normalized attribute values.

Usage

The **slapi_attr_get_normalized_values()** routine returns the address of the array of normalized attribute values. The attribute values are normalized using the equality matching rule for the attribute type as defined in the LDAP schema. The unnormalized attribute values are returned if the attribute type does not have an equality matching rule.

Related topics

The function return value is 0 if the normalized attribute values are returned and -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EINVAL A parameter is not valid

slapi_attr_get_numvalues()

Purpose

Obtain the number of attribute values.

Format

```
#include <slapi-plugin.h>

int slapi_attr_get_numvalues (
 Slapi_Attr *          attr,
 int *                 numValues)
```

Parameters

Input

attr The directory entry attribute.

Output

numValues This variable is set to the number of attribute values.

Usage

The **slapi_attr_get_numvalues()** routine returns the number of values for the supplied attribute.

Related topics

The function return value is 0 if the number of attribute values is returned or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EINVAL A parameter is not valid

slapi_attr_get_type()

Purpose

Obtain the attribute type.

Format

```
#include <slapi-plugin.h>

int slapi_attr_get_type (
  Slapi_Attr *       attr,
  char **            type)
```

Parameters

Input

attr The directory entry attribute.

Output

type This variable is set to the address of the attribute type. The application must not modify or release the attribute type.

Usage

The **slapi_attr_get_type()** routine returns the name of a directory attribute. The returned value is the lowercased primary attribute name as defined in the LDAP schema.

Related topics

The function return value is 0 if the attribute type is returned or -1 if an error occurred. The errno variable is set to one of the following values when the function return value is -1:

EINVAL	A parameter is not valid
ENOENT	Attribute type is not set

slapi_attr_get_values()

Purpose

Obtain the attribute values.

Format

```
#include <slapi-plugin.h>

int slapi_attr_get_values (
    Slapi_Attr *         attr,
    BerVal ***           vals)
```

Parameters

Input

attr The directory entry attribute.

Output

vals This variable is set to the address of the attribute value array or to NULL if there are no attribute values. The end of the array is indicated by a NULL `BerVal` address. The application must not modify or release the attribute values.

Usage

The **slapi_attr_get_values()** routine returns the address of the array of attribute values.

Related topics

The function return value is 0 if the attribute values are returned and -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EINVAL A parameter is not valid

slapi_attr_value_cmp()

Purpose

Compare two attribute values.

Format

```
#include <slapi-plugin.h>

int slapi_attr_value_cmp (
     Slapi_Attr *       attr,
     BerVal *           value1,
     BerVal *           value2)
```

Parameters

Input

attr The directory entry attribute.

values1 The first attribute value.

values2 The second attribute value.

Usage

The **slapi_attr_value_cmp()** routine compares two values using the equality matching rule for the attribute type as defined in the LDAP schema. The unnormalized attribute values are compared if there is no equality matching rule for the attribute type.

Related topics

The function return value is 0 if the attribute values are equal, 1 if the attribute values are not equal and -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EILSEQ	Unable to normalize attribute value
EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available
ESRCH	Attribute type is not defined in LDAP schema

slapi_ch_calloc()

Purpose

Allocate storage for an array.

Format

```
#include <slapi-plugin.h>

void * slapi_ch_calloc (
   unsigned long     elemCount,
   unsigned long     elemSize)
```

Parameters

Input

elemCount The number of elements in the array.

elemSize The size of each element in the array.

Usage

The **slapi_ch_calloc()** routine allocates storage for an array. Call the **slapi_ch_free()** routine to release the storage when it is no longer needed.

Related topics

The function return value is the address of the allocated storage or NULL if the storage is not allocated. The *errno* variable is set to ENOMEM if the storage is not allocated.

slapi_ch_free()

Purpose

Release allocated storage.

Format

```
#include <slapi-plugin.h>

void slapi_ch_free (
    void *          ptr)
```

Parameters

Input

ptr The address of the storage is released.

Usage

The **slapi_ch_free()** routine releases allocated storage.

Related topics

There is no function return value.

slapi_ch_free_values()

Purpose

Release an array of values.

Format

```
#include <slapi-plugin.h>

void slapi_ch_free_values (
    BerVal **          values)
```

Parameters

Input

values The array of values. The end of the array is indicated by a NULL `BerVal` address.

Usage

The **slapi_ch_free_values()** routine releases an array of `BerVal` structures. Each value is released and then the array is released.

Related topics

There is no function return value.

slapi_ch_malloc()

Purpose

Allocate storage.

Format

```
#include <slapi-plugin.h>

void * slapi_ch_malloc (
  unsigned long       size)
```

Parameters

Input

size The number of bytes is allocated.

Usage

The **slapi_ch_malloc()** routine allocates storage for use by the plug-in. Call the **slapi_ch_free()** routine to release the storage when it is no longer needed.

Related topics

The function return value is the address of the allocated storage or NULL if the storage is not allocated. The *errno* variable is set to ENOMEM if the storage is not allocated.

slapi_ch_realloc()

Purpose

Reallocate storage.

Format

```
#include <slapi-plugin.h>

void * slapi_ch_realloc (
   void *               block,
   unsigned long        newSize)
```

Parameters

Input

block The address of block is reallocated.

newSize The new size for the block.

Usage

The **slapi_ch_realloc()** routine reallocates a block of storage. The size of the original block is changed or a new block of storage is allocated. The contents of the original block of storage are copied to the new block and the original block is released if a new block of storage is allocated. Call the **slapi_ch_free()** routine to release the storage when it is no longer needed.

Related topics

The function return value is the address of the reallocated storage or NULL if the storage is not reallocated. The *errno* variable is set to ENOMEM if the storage is not reallocated. The original storage block is still allocated if the reallocate request is not successful.

slapi_ch_strdup()

Purpose

Duplicate a character string.

Format

```
#include <slapi-plugin.h>

char * slapi_ch_strdup (
   const char *       string)
```

Parameters

Input

string The string is duplicated.

Usage

The **slapi_ch_strdup()** routine duplicates a character string by allocating storage for the new string and then copying the original string to the allocated storage. Call the **slapi_ch_free()** routine to release the copied string when it is no longer needed.

Related topics

The function return value is the address of the duplicated string or NULL if the storage is not allocated. The *errno* variable sets to ENOMEM if the storage is not allocated.

slapi_compare_internal()

Purpose

Issue a COMPARE request.

Format

```
#include <slapi-plugin.h>

Slapi_PBlock * slapi_compare_internal (
   const char *          dn,
   const char *          type,
   const BerVal *        value,
   LDAPControl **        controls)
```

Parameters

Input

dn The distinguished name of the entry.

type The attribute name.

value The attribute value.

controls A NULL-terminated array of server controls for the COMPARE request. Specify NULL if there are no server controls.

Usage

The **slapi_compare_internal()** routine issues a COMPARE request and returns the results to the plug-in for processing. The LDAP Version 3 protocol and the current client authentication are used for the COMPARE request. The request is unauthenticated if a client request is not being processed. The **slapi_pblock_get()** routine is called to obtain the results from the returned parameter block. The following values are retrieved from the parameter block:

- `SLAPI_PLUGIN_INTOP_RESULT` - The result code from the result message.
- `SLAPI_PLUGIN_INTOP_ERRMSG` - The error message from the result message.
- `SLAPI_PLUGIN_INTOP_MATCHED_DN` - The matched DN from the result message.
- `SLAPI_PLUGIN_INTOP_REFERRALS` - The referrals from the result message.

Related topics

The function return value is the address of the plug-in parameter block or NULL if the COMPARE request is not issued. Call the **slapi_pblock_destroy()** routine to release the plug-in parameter block when it is no longer needed. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
EIO	Unable to process the COMPARE request
ENOMEM	Insufficient storage is available

slapi_control_present()

Purpose

Determine if a server control is present.

Format

```
#include <slapi-plugin.h>

int slapi_control_present (
   LDAPControl **           controls,
   const char *             oid,
   BerVal **                value,
   int *                    isCritical)
```

Parameters

Input

controls The array of server controls. The end of the array is indicated by a NULL control address.

oid The object identifier of the desired control.

Output

value This variable is set to the address of the control value if the control is found. The application must not modify or release the control value.

isCritical The returned value is 1 if the control is critical and 0 otherwise.

Usage

The **slapi_control_present()** routine searchs an array of server controls for a control with the specified object identifier. If the control is found, a pointer to the control value is returned along with an indication of whether or not the control is marked as critical.

Related topics

The function return value is 1 if the control is found, 0 if the control is not found and -1 if an error occurred. The *errno* variable ise set to one of the following values when the function return value is -1:

EINVAL A parameter is not valid

ENOMEM Insufficient storage is available

slapi_delete_internal()

Purpose

Issue a DELETE request.

Format

```
#include <slapi-plugin.h>

Slapi_PBlock * slapi_delete_internal (
  const char *          dn,
  LDAPControl **        controls,
  int                   l)
```

Parameters

Input

dn The distinguished name of the entry.

controls A NULL-terminated array of server controls for the DELETE request. Specify NULL if there are no server controls.

l This parameter is not used and should be set to 0. It is included for compatibility with other LDAP implementations.

Usage

The **slapi_delete_internal()** routine issues a DELETE request and returns the results to the plug-in for processing. The LDAP Version 3 protocol and the current client authentication is used for the DELETE request. The request is unauthenticated if a client request is not being processed. Call the **slapi_pblock_get()** routine to obtain the results from the returned parameter block. The following values can be retrieved from the parameter block:

- `SLAPI_PLUGIN_INTOP_RESULT` - The result code from the result message
- `SLAPI_PLUGIN_INTOP_ERRMSG` - The error message from the result message
- `SLAPI_PLUGIN_INTOP_MATCHED_DN` - The matched DN from the result message
- `SLAPI_PLUGIN_INTOP_REFERRALS` - The referrals from the result message

Related topics

The function return value is the address of a plug-in parameter block or NULL if the DELETE request is not issued. Call the **slapi_pblock_destroy()** routine to release the plug-in parameter block when it is no longer needed. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
EIO	Unable to process the DELETE request
ENOMEM	Insufficient storage is available

slapi_dn_ignore_case_v3()

Purpose

Normalize a distinguished name and convert to lowercase.

Format

```
#include <slapi-plugin.h>

char * slapi_dn_ignore_case_v3 (
   const char *        dn)
```

Parameters

Input

dn The distinguished name to be normalized.

Usage

The **slapi_dn_ignore_case_v3()** routine converts a distinguished name (DN) by removing leading and trailing spaces, spaces between name components and spaces around the equals signs. The API normalizes the attribute type name to the lowercased primary attribute type name in the LDAP schema definition. Any semicolons used to separate relative distinguished names (RDN) are converted to commas. The entire name is then converted to lowercase. A compound RDN is sorted alphabetically by the primary attribute type names. Special characters within a DN are represented using the backslash (\) escape character. For example,

```
cn="a + b", o=ibm, c=us
```

is converted to

```
cn=a\+b,o=ibm,c=us
```

Escaped hexadecimal attribute values are converted to the character representation. For example,

```
cn=\4a\6f\68\6e Doe,ou=Engineering,o=Darius
```

is converted to

```
cn=john doe,ou=engineering,o=darius
```

BER-encoded attribute values are converted to UTF-8 values. For example,

```
cn=#04084a6f686e20446f65,ou=Engineering,o=Darious
```

is converted to

```
cn=john doe,ou=engineering,o=darius
```

If an attribute type is not defined in the LDAP schema, the primary attribute type name is the lowercased attribute type.

Related topics

The function return value is the normalized name or NULL if an error occurred. Call the **slapi_ch_free()** routine to release the normalized name when it is no longer needed. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available

NULL will be returned if a NULL DN is passed in and EINVAL will be the return value.d EINVAL will be the return value.

slapi_dn_isparent()

Purpose

Determines whether or not a particular DN is the parent of another specified DN. Before calling this function, call **slapi_dn_ignore_case_v3** to normalize the DNs, which also converts all characters to lowercase.

Format

```
#include <slapi-plugin.h>

int slapi_dn_isparent(
  const char *       parentdn,
  const char *       childdn )
```

Parameters

Input

parentdn Determine if this DN is the parent of childdn.

childdn Determine if this DN is the child of parentdn.

Usage

The **slapi_dn_isparent()** routine takes two normalized, lowercase DNs as input and compares them, determining if the first DN is the parent of the second DN. Input string formats are expected to be UTF-8 characters.

Related topics

A nonzero positive value is returned if *parentdn* is the parent of *childdn*, 0 if the *parentdn* is not the parent of *childdn* and -1 if an error is detected.

slapi_dn_normalize_v3()

Purpose

Normalize a distinguished name and preserve the case of attribute values.

Format

```
#include <slapi-plugin.h>

char * slapi_dn_normalize_v3 (
   const char *         dn)
```

Parameters

Input

dn The distinguished name to be normalized.

Usage

The **slapi_dn_normalize_v3()** routine converts a distinguished name (DN) by removing leading and trailing spaces, spaces between name components and spaces around the equals signs. The API normalizes the attribute type name to the primary attribute type name in the LDAP schema definition. Any semicolons used to separate relative distinguished names (RDN) are converted to commas. A compound RDN is sorted alphabetically by the primary attribute type names. Special characters within a DN are represented using the backslash (\) escape character. For example,

```
cn="a + b", o=ibm, c=us
```

is converted to

```
cn=a\+b,o=ibm,c=us
```

Escaped hexadecimal attribute values are converted to the character representation. For example,

```
cn=\4a\6f\68\6e Doe,ou=Engineering,o=Darius
```

is converted to

```
cn=John Doe,ou=Engineering,o=Darius
```

BER-encoded attribute values are converted to UTF-8 values. For example,

```
cn=#04084a6f686e20446f65,ou=Engineering,o=Darious
```

is converted to

```
cn=John Doe,ou=Engineering,o=Darius
```

If an attribute type is not defined in the LDAP schema, the primary attribute type name is the lowercased attribute type.

Related topics

The function return value is the normalized name or NULL if an error occurred. Call the **slapi_ch_free()** routine to release the normalized name when it is no longer needed. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available

NULL will be returned if a NULL DN is passed in and EINVAL will be the return value.

slapi_dn_normalize_case_v3()

Purpose

Normalize a distinguished name and convert case-insensitive attribute values to uppercase.

Format

```
#include <slapi-plugin.h>

char * slapi_dn_normalize_case_v3 (
   const char *         dn)
```

Parameters

Input

dn The distinguished name to be converted.

Usage

The **slapi_dn_normalize_case_v3()** routine:

- Converts a distinguished name (DN) to a canonical form by removing leading and trailing spaces, spaces between name components and spaces around the equals signs
- Normalizes the attribute type name to the uppercased primary attribute type name in the LDAP schema definition
- Any semicolons used to separate relative distinguished names (RDN) are converted to commas
- A compound RDN is sorted alphabetically by the primary attribute type names
- An attribute value is converted to uppercase if the associated matching rule is case insensitive, otherwise the case of the attribute value is preserved
- Special characters within a DN are represented using the backslash (\) escape character

For example,

```
cn="a + b", o=ibm, c=us
```

is converted to

```
CN=A\+B,O=IBM,C=US
```

Escaped hexadecimal attribute values are converted to the character representation. For example,

```
cn=\4a\6f\68\6e Doe,ou=Engineering,o=Darius
```

is converted to

```
CN=JOHN DOE,OU=ENGINEERING,O=DARIUS
```

BER-encoded attribute values are converted to UTF-8 values. For example,

```
cn=#04084a6f686e20446f65,ou=Engineering,o=Darious
```

is converted to

```
CN=JOHN DOE,OU=ENGINEERING,O=DARIUS
```

If an attribute type is not defined in the LDAP schema, the primary attribute type name is the uppercased attribute type and the attribute matching rule is caseIgnoreMatch.

Related topics

The function return value is the normalized name or NULL if an error occurred. Call the **slapi_ch_free()** routine to release the normalized name when it is no longer needed. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available

NULL will be returned if a NULL DN is passed in and EINVAL will be the return value.

slapi_entry_add_value()

Purpose

Add an attribute value to a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_add_value (
    Slapi_Entry *          entry,
    const char *           type,
    BerVal *               value)
```

Parameters

Input

entry The directory entry.

type The attribute name. This can be the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

value The attribute value to be added. Specify NULL to add the attribute without a value (an error is returned if the attribute already exists).

Usage

The **slapi_entry_add_value()** routine adds an attribute value to a directory entry that was allocated by the **slapi_entry_alloc()** routine. A case-insensitive compare is used when searching for the attribute type. The attribute type is created if it does not already exist for the entry. An error is returned if the entry already contains the attribute value. Use the **slapi_entry_merge_value()** routine if you want to ignore a duplicate attribute value. Use the **slapi_entry_replace_value()** routine to replace the existing attribute values with the new value.

The **slapi_entry_add_value()** routine makes a copy of the supplied attribute value. An error is returned if the attribute value is not normalized using the equality matching rule defined for the attribute type.

Related topics

The function return value is 0 if the attribute value is added to the entry or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EEXIST	The attribute value already exists
EILSEQ	Unable to normalize attribute value
EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available
ESRCH	Attribute type is not defined in LDAP schema

slapi_entry_add_values()

Purpose

Add an attribute value to a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_add_values (
    Slapi_Entry *          entry,
    const char *           type,
    BerVal *               values)
```

Parameters

Input

entry The directory entry.

type The attribute name. This can be the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

value A NULL-terminated array of values to be added.

Usage

The **slapi_entry_add_values()** routine adds multiple attribute values to a directory entry that was allocated by the **slapi_entry_alloc()** routine. A case-insensitive compare is used when searching for the attribute type. The attribute type is created if it does not already exist for the entry. An error is returned if the entry already contains one of the supplied attribute values and none of the attribute values are added to the entry. Use the **slapi_entry_merge_values()** routine to add non-matching attribute values when the entry contains one or more matching attribute values. Use the **slapi_entry_replace_values()** routine to replace the existing attribute values with the new values.

The **slapi_entry_add_values()** routine makes copies of the supplied attribute values. An error is returned if the attribute values is not normalized using the equality matching rule defined for the attribute type.

Related topics

The function return value is 0 if the attribute value is added to the entry or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EEXIST	The attribute value already exists
EILSEQ	Unable to normalize attribute value
EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available
ESRCH	Attribute type is not defined in LDAP schema

slapi_entry_alloc()

Purpose

Allocate a new directory entry.

Format

```
#include <slapi-plugin.h>

Slapi_Entry * slapi_entry_alloc ( void )
```

Parameters

None.

Usage

The **slapi_entry_alloc()** routine allocates a new directory entry. After the entry is allocated, the **slapi_entry_set_dn()** routine is called to set the entry distinguished name and the **slapi_entry_add_values()** routine is called to add the entry attributes. The **slapi_entry_free()** routine is called to release the directory entry when it is no longer needed.

Related topics

The function return value is the address of the new entry or NULL if an error occurred. The *errno* variable is set to one of the following values when the function return value is NULL:

ENOMEM	Insufficient storage is available

slapi_entry_attr_delete()

Purpose

Delete a directory entry attribute.

Format

```
#include <slapi-plugin.h>

int slapi_entry_attr_delete (
  Slapi_Entry *        entry,
  const char *         type)
```

Parameters

Input

entry The directory entry.

type The attribute name. This is the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

Usage

The **slapi_entry_attr_delete()** routine deletes an attribute from a directory entry. A case-insensitive compare is used when searching for the attribute type.

Related topics

The function return value is 0 if the attribute was deleted, 1 if the entry does not contain the attribute, and -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available

slapi_entry_attr_find()

Purpose

Find a directory entry attribute.

Format

```
#include <slapi-plugin.h>

int slapi_entry_attr_find (
  Slapi_Entry *          entry,
  const char *           type,
  Slapi_Attr **          attr)
```

Parameters

Input

entry The directory entry.

type The attribute name. This is the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

Output

attr This variable is set to the address of the attribute if the attribute is found in the directory entry. The application must not modify or release the attribute.

Usage

The **slapi_entry_attr_find()** routine searches the directory entry for the specified attribute and returns the address of the attribute if it is found. A case-insensitive compare is used when searching for the attribute type. The attribute name in the returned attribute is the lowercased primary attribute name as defined in the LDAP schema.

Related topics

The function return value is 0 if the attribute is found and -1 otherwise. The *errno* variable sets to one of the following values when the function return value is -1:

EINVAL	A parameter is not valid
ENOENT	Attribute not found
ESRCH	Attribute type is not defined in LDAP schema

slapi_entry_delete_value()

Purpose

Remove an attribute value from a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_delete_value (
  Slapi_Entry *           entry,
  const char *            type,
  BerVal *                value)
```

Parameters

Input

entry The directory entry.

type The attribute name. This is the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

value The attribute value to be deleted.

Usage

The **slapi_entry_delete_value()** routine removes an attribute value from a directory entry. The attribute is deleted if there are no attribute values left after deleting the requested value. A case-insensitive compare is used when searching for the attribute type. An error is returned if the entry does not contain the requested attribute value. Use the **slapi_entry_attr_delete()** routine to delete an attribute and all of its values.

An error is returned if the attribute value is not normalized using the equality matching rule defined for the attribute type.

Related topics

The function return value is 0 if the requested attribute value is deleted or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EILSEQ	Unable to normalize attribute value
EINVAL	A parameter is not valid
ENOENT	The attribute value was not found
ESRCH	Attribute type is not defined in LDAP schema

slapi_entry_delete_values()

Purpose

Remove multiple attribute values from a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_delete_values (
  Slapi_Entry *         entry,
  const char *          type,
  BerVal *              values)
```

Parameters

Input

entry The directory entry.

type The attribute name. This is the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

values A NULL-terminated array of attribute values to be deleted.

Usage

The **slapi_entry_delete_values()** routine removes multiple attribute values from a directory entry. The attribute is deleted if there are no attribute values left after deleting the requested values. A case-insensitive compare is used when searching for the attribute type. An error is returned if the entry does not contain the requested attribute values. Use the **slapi_entry_attr_delete()** routine to delete an attribute and all of its values.

An error is returned if the attribute value is not normalized using the equality matching rule defined for the attribute type.

Related topics

The function return value is 0 if the requested attribute values are deleted or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EILSEQ	Unable to normalize attribute value
EINVAL	A parameter is not valid
ENOENT	The attribute value was not found
ESRCH	Attribute type is not defined in LDAP schema

slapi_entry_dup()

Purpose

Duplicate a directory entry.

Format

```
#include <slapi-plugin.h>

Slapi_Entry * slapi_entry_dup (
   Slapi_Entry *        entry)
```

Parameters

Input

entry The directory entry to be duplicated.

Usage

The **slapi_entry_dup()** routine creates a copy of a directory entry. Call the **slapi_entry_free()** routine to release the copied directory entry when it is no longer needed.

Related topics

The function return value is the address of the copied directory entry or NULL if an error occurred. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available

slapi_entry_first_attr()

Purpose

Obtain the first attribute in a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_first_attr (
  Slapi_Entry *        entry,
  Slapi_Attr **        attr)
```

Parameters

Input

entry The directory entry.

Output

attr This variable is set to the address of the first attribute. The application must not modify or release the attribute.

Usage

The **slapi_entry_first_attr()** routine returns the first attribute in a directory entry. The attribute type in the returned attribute is the primary attribute name. The application cycles through all of the entry attributes by calling **slapi_entry_first_attr()** to obtain the first attribute and then repeatedly calling **slapi_entry_next_attr()** to obtain the remaining attributes.

Related topics

The function return value is 0 if the attribute is found and -1 otherwise. The *errno* variable sets to one of the following values when the function return value is -1:

EINVAL	A parameter is not valid
ENOENT	The entry has no attributes

slapi_entry_free()

Purpose

Free a directory entry.

Format

```
#include <slapi-plugin.h>

void slapi_entry_free (
   Slapi_Entry *       entry)
```

Parameters

Input

entry The directory entry to be freed.

Usage

The **slapi_entry_free()** routine frees a directory entry that was allocated by the **slapi_entry_alloc()** or **slapi_entry_dup()** routine. The entry name and any entry attributes are freed.

Related topics

There is no function return value.

slapi_entry_get_dn()

Purpose

Obtain the directory entry name.

Format

```
#include <slapi-plugin.h>

char * slapi_entry_get_dn (
  Slapi_Entry *       entry)
```

Parameters

Input

entry The directory entry.

Usage

The **slapi_entry_get_dn()** routine returns the distinguished name of a directory entry. This name must not be modified or released by the application.

Related topics

The function return value is the address of the entry name or NULL if an error occurred. The *errno* variable sets to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
ENOENT	Attribute type is not set

slapi_entry_merge_value()

Purpose

Add an attribute value to a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_merge_value (
  Slapi_Entry *         entry,
  const char *          type,
  BerVal *              value)
```

Parameters

Input

entry The directory entry.

type The attribute name. This is the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

value The attribute value to be added. Specify NULL to add the attribute without a value (the attribute is created if it does not exist).

Usage

The **slapi_entry_merge_value()** routine adds an attribute value to a directory entry that was allocated by the **slapi_entry_alloc()** routine. A case-insensitive compare is used when searching for the attribute type. The attribute type is created if it does not already exist for the entry. No error is returned if the entry already contains the supplied attribute value. Use the **slapi_entry_add_value()** routine to add the attribute value if you want to be notified when a duplicate attribute value exists. Use the **slapi_entry_replace_value()** routine to replace the existing attribute values with a new value.

The **slapi_entry_merge_value()** routine makes a copy of the supplied attribute value. An error is returned if the attribute value is not normalized using the equality matching rule defined for the attribute type.

Related topics

The function return value is 0 if the attribute value is added to the entry or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EILSEQ	Unable to normalize attribute value
EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available
ESRCH	Attribute type is not defined in LDAP schema

slapi_entry_merge_values()

Purpose

Add multiple attribute values to a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_merge_values (
  Slapi_Entry *         entry,
  const char *          type,
  BerVal *              values)
```

Parameters

Input

entry The directory entry.

type The attribute name. This is the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

values A NULL-terminated array of values to be added.

Usage

The **slapi_entry_merge_values()** routine adds multiple attribute values to a directory entry that was allocated by the **slapi_entry_alloc()** routine. A case-insensitive compare is used when searching for the attribute type. The attribute type is created if it does not already exist for the entry. No error is returned if the entry already contains the supplied attribute values. Use the **slapi_entry_add_values()** routine to add the attribute value if you want to be notified when the entry contains one or more matching attribute values. Use the **slapi_entry_replace_values()** routine to replace the existing attribute values with the new values.

The **slapi_entry_merge_values()** routine makes copies of the supplied attribute values. An error is returned if the attribute values is not normalized using the equality matching rule defined for the attribute type.

Related topics

The function return value is 0 if the attribute values are added to the entry or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EILSEQ	Unable to normalize attribute value
EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available
ESRCH	Attribute type is not defined in LDAP schema

slapi_entry_next_attr()

Purpose

obtain the next attribute in a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_next_attr (
  Slapi_Entry *         entry,
  Slapi_Attr *          prevAttr,
  Slapi_Attr **         attr)
```

Parameters

Input

entry The directory entry.

prevAttr The previous attribute returned by **slapi_entry_first_attr()** or **slapi_entry_next_attr()**.

Output

attr This variable is set to the address of the attribute following the attribute specified by the *prevAttr* parameter. The application must not modify or release the attribute.

Usage

The **slapi_entry_next_attr()** routine returns the next attribute in a directory entry. The attribute type in the returned attribute is the primary attribute name. The application cycles through all of the entry attributes by calling **slapi_entry_first_attr()** to obtain the first attribute and then repeatedly calling **slapi_entry_next_attr()** to obtain the remaining attributes.

Related topics

The function return value is 0 if the attribute is found and -1 otherwise. The *errno* variable sets to one of the following values when the function return value is -1:

EINVAL	A parameter is not valid
ENOENT	There are no more attributes

slapi_entry_replace_value()

Purpose

Replace the attribute values in a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_replace_value (
  Slapi_Entry *         entry,
  const char *          type,
  BerVal *              value)
```

Parameters

Input

entry The directory entry.

type The attribute name. This is the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

value The replacement attribute value. Specify NULL to remove all of the values for the attribute (the attribute is created if it does not exist).

Usage

The **slapi_entry_replace_value()** routine replaces all of the attribute values in a directory entry that was allocated by the **slapi_entry_alloc()** routine. A case-insensitive compare is used when searching for the attribute type. The attribute type is created if it does not already exist for the entry. Use the **slapi_entry_add_value()** or **slapi_entry_merge_value()** routine to add an attribute value to the existing values.

The **slapi_entry_replace_value()** routine makes a copy of the supplied attribute value. An error is returned if the attribute value is not normalized using the equality matching rule defined for the attribute type.

Related topics

The function return value is 0 if the attribute values are replaced or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EILSEQ	Unable to normalize attribute value
EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available
ESRCH	Attribute type is not defined in LDAP schema

slapi_entry_replace_values()

Purpose

Replace the attribute values in a directory entry.

Format

```
#include <slapi-plugin.h>

int slapi_entry_replace_values (
  Slapi_Entry *         entry,
  const char *          type,
  BerVal *              values)
```

Parameters

Input

entry The directory entry.

type The attribute name. This is the attribute object identifier, the primary attribute name or an alternate attribute name as defined in the LDAP schema.

value A NULL-terminated array of replacement values.

Usage

The **slapi_entry_replace_values()** routine replaces all of the attribute values in a directory entry that was allocated by the **slapi_entry_alloc()** routine. A case-insensitive compare is used when searching for the attribute type. The attribute type is created if it does not already exist for the entry. Use the **slapi_entry_add_values()** or **slapi_entry_merge_values()** routine to add attribute values to the existing values.

The **slapi_entry_replace_values()** routine makes a copy of the supplied attribute value. An error is returned if the attribute value is not normalized using the equality matching rule defined for the attribute type.

Related topics

The function return value is 0 if the attribute values are replaced or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EEXIST	Duplicate value in replacement values
EILSEQ	Unable to normalize attribute value
ENOMEM	Insufficient storage is available
ESRCH	Attribute type is not defined in LDAP schema

slapi_entry_schema_check()

Purpose

Check a directory entry against the LDAP schema.

Format

```
#include <slapi-plugin.h>

int slapi_entry_schema_check (
  Slapi_Entry *         entry)
```

Parameters

Input

entry The directory entry.

Usage

The **slapi_entry_schema_check()** routine validates a directory entry using the LDAP schema.

An error is returned if any of the following conditions are true:

- The entry contains an undefined attribute type or object class.
- The entry contains an obsolete attribute type or object class.
- The entry contains an attribute type that can not be modified by the user.
- The entry contains an attribute type that is not allowed by the entry object classes.
- A single-valued attribute type contains multiple attribute values.
- A required attribute type is not found and the extensibleObject object class is not specified.
- There is not one and only one base structural object class.
- An auxiliary object class is a base object class.

Related topics

The function return value is 0 if the directory entry is valid or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EDOM	Obsolete attribute type or object class
EEXIST	Single-valued attribute has multiple values
EILSEQ	An auxiliary object class is a base object class or there is not one and only one structural object class
EINVAL	A parameter is not valid
ENOENT	Required attribute not found
ENOMEM	Insufficient storage is available
EPERM	Attribute cannot be modified by user
ERANGE	Attribute not allowed by object class
ESRCH	Undefined attribute type or object class

slapi_entry_set_dn()

Purpose

Set the directory entry name.

Format

```
#include <slapi-plugin.h>

int slapi_entry_set_dn (
  Slapi_Entry *       entry,
  const char *        dn)
```

Parameters

Input

entry The directory entry.

dn The distinguished name for the entry.

Usage

The **slapi_entry_set_dn()** routine sets or changes the entry name for a directory entry allocated by the **slapi_entry_alloc()** routine. The **slapi_entry_set_dn()** routine must not be used to change the name in a directory entry returned by the **slapi_pblock_get()** routine. The **slapi_entry_set_dn()** routine stores a copy of the supplied name in the directory entry. The storage for the previous DN is released.

Related topics

The function return value is 0 if the entry name is set and -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available

slapi_filter_get_attribute_type()

Purpose

Obtain the search filter attribute type.

Format

```
#include <slapi-plugin.h>

int slapi_filter_get_attribute_type (
   Slapi_Filter *    filter,
   char **           type)
```

Parameters

Input

filter The search filter.

Output

type This variable is set to the address of the attribute type for the search filter. The application must not modify or release the attribute type.

Usage

The **slapi_filter_get_attribute_type()** routine returns the attribute type for the following search filters:

```
LDAP_FILTER_APPROX
LDAP_FILTER_EQUALITY
LDAP_FILTER_GE
LDAP_FILTER_LE
LDAP_FILTER_PRESENT
LDAP_FILTER_SUBSTRINGS
```

An error is returned if the search filter is not one of these types. The attribute type is the lowercased primary attribute name as defined in the LDAP schema.

Related topics

The function return value is 0 if the attribute type is returned or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EINVAL	A parameter is not valid
EPERM	The filter does not have an attribute type
ESRCH	Attribute type is not defined in LDAP schema

slapi_filter_get_ava()

Purpose

Obtain the search filter assertion value.

Format

```
#include <slapi-plugin.h>

int slapi_filter_get_ava (
  Slapi_Filter *       filter,
  char **              type,
  BerVal **            value)
```

Parameters

Input

filter The search filter.

Output

type This variable is set to the address of the attribute type for the search filter. The application must not modify or release the attribute type.

value This variable is set to the address of the assertion value for the search filter. The application must not modify or release the assertion value.

Usage

The **slapi_filter_get_ava()** routine returns the assertion value for the following search filters:

```
LDAP_FILTER_APPROX
LDAP_FILTER_EQUALITY
LDAP_FILTER_GE
LDAP_FILTER_LE
LDAP_FILTER_PRESENT
```

An error is returned if the search filter is not one of these types. The attribute type is the lowercased primary attribute name as defined in the LDAP schema. The assertion value is normalized using the equality matching rule for the attribute type. An error is returned if the assertion value is not normalized.

Related topics

The function return value is 0 if the assertion value is returned or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EILSEQ	Assertion value is not normalized
EINVAL	A parameter is not valid
EPERM	The filter does not have an attribute type
ESRCH	Attribute type is not defined in LDAP schema

slapi_filter_get_choice()

Purpose

Obtain the search filter type.

Format

```
#include <slapi-plugin.h>

int slapi_filter_get_choice (
  Slapi_Filter *        filter)
```

Parameters

Input

filter The search filter.

Usage

The **slapi_filter_get_choice()** routine returns the search filter type. The top-level search filter is obtained by calling the **slapi_pblock_get()** routine with the SLAPI_SEARCH_FILTER parameter. Lower-level search filters are obtained by calling the **slapi_filter_list_first()** and **slapi_filter_list_next()** routines.

The following search filter types are defined:

Table 1. **slapi_filter_get_choice()** *search filters*

Header	Header
LDAP_FILTER_AND	AND filter: (&(cn=John)(sn=Doe))
LDAP_FILTER_OR	OR filter: (|(cn=John)(cn=Jane))
LDAP_FILTER_NOT	NOT filter: (!(cn=John))
LDAP_FILTER_EQUALITY	EQ filter: (cn=John)
LDAP_FILTER_GE	GE filter: (cn>=John)
LDAP_FILTER_LE	LE filter: (cn<=John)
LDAP_FILTER_PRESENT	Presence filter: (cn=*)
LDAP_FILTER_APPROX	Approximate filter: (cn~=John)
LDAP_FILTER_SUBSTRINGS	Substrings filter: (cn=J*Doe)

Related topics

The function return value is one of the above search filter types or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EINVAL A parameter is not valid

slapi_filter_get_subfilt()

Purpose

Obtain the search filter substrings.

Format

```
#include <slapi-plugin.h>

int slapi_filter_get_subfilt (
  Slapi_Filter *        filter,
  char **               type,
  char **               initial,
  char ***              any,
  char **               final)
```

Parameters

Input

filter The search filter.

Output

type This variable is set to the address of the attribute type for the search filter. The application must not modify or release the attribute type.

initial This variable is set to the address of the 'initial' substring or NULL if there is no 'initial' substring. The application must not modify or release the substring.

any This variable is set to the address of the array of 'any' substrings or NULL if there are no 'any' substrings. The end of the array is indicated by a NULL string address. The application must not modify or release the substrings.

final This variable is set to the address of the 'final' substring or NULL if there is no 'final' substring. The application must not modify or release the substring.

Usage

The **slapi_filter_get_subfilt()** routine returns the substrings for an LDAP_FILTER_SUBSTRINGS search filter. An error is returned if this is not a substrings filter. The attribute type is the lowercased primary attribute name as defined in the LDAP schema. The substrings are normalized using the equality matching rule for the attribute type. An error is returned if the substrings are not normalized.

For example, if the `filter` is *(cn=John*Q*Public)*, the `initial` substring is `John`, the `final` substring is `Public`, and the `any` substrings array contains the single substring `Q`.

Related topics

The function return value is 0 if the substrings are returned or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EILSEQ	Unable to normalize attribute value
EINVAL	A parameter is not valid
EPERM	The filter does not have substrings
ESRCH	Attribute type is not defined in LDAP schema

slapi_filter_list_first()

Purpose

Obtain the first subfilter.

Format

```
#include <slapi-plugin.h>

Slapi_Filter * slapi_filter_list_first (
  Slapi_Filter *      filter)
```

Parameters

Input

filter The search filter.

Usage

The **slapi_filter_list_first()** routine returns the first subfilter in an AND, OR, or NOT filter. For example, if the search `filter` is `(&(cn=John)(sn=Doe))`, the first `subfilter` is `(cn=John)`. The top-level search filter is obtained by calling the **slapi_pblock_get()** routine with the SLAPI_SEARCH_FILTER parameter. Lower-level search filters are obtained by calling the **slapi_filter_list_first()** and **slapi_filter_list_next()** routines.

Related topics

The function return value is the first subfilter or NULL if an error occurred. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
ENOENT	There are no subfilters
EPERM	The filter is not an AND, OR, or NOT filter

slapi_filter_list_next()

Purpose

Obtain the next subfilter.

Format

```
#include <slapi-plugin.h>

Slapi_Filter * slapi_filter_list_next (
  Slapi_Filter *         filter,
  Slapi_Filter *         subfilter)
```

Parameters

Input

filter The search filter.

subfilter The current subfilter.

Usage

The **slapi_filter_list_next()** routine returns the next subfilter in an AND or OR filter. For example, if the search `filter` is `(&(cn=John)(sn=Doe))` and the current `subfilter` is `(cn=John)`, then the next `subfilter` is `(sn=Doe)`. The return value is NULL and *errno* is set to **ENOENT** when all of the subfilters have been processed.

Related topics

The function return value is the next *subfilter* or NULL if an error occurred. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
ENOENT	There are no subfilters
EPERM	The filter is not an AND, OR, or NOT filter

slapi_isSDBM_authenticated()

Purpose

Determines whether or not the client BIND DN is contained in the SDBM backend.

Format

```
#include <slapi_plugin.h>

int   slapi_isSDBM_authenticated (
  Slapi_PBlock *  pb )
```

Parameters

Input

pb The plug-in parameter block.

Usage

The **slapi_isSDBM_authenticated()** routine retrieves the BIND DN associated with the connection from the plug-in parameter block and checks whether the DN belongs to the SDBM backend, meaning the BIND DN is authenticated by the RACF® security server.

Related topics

A nonzero positive value is returned if the BIND DN was authenticated by the security server, 0 if it was not authenticated, and -1 if an error is detected.

slapi_log_error()

Purpose

Write a message to the LDAP server job log.

Format

```
#include <slapi-plugin.h>

void slapi_log_error (
              int         msg_severity,
              char *      subsystem,
              char *      fmt, ...)
```

Parameters

Input

msg_severity Level of severity of the message. Level of severity is one of the following:
- LDAP_MSG_LOW
- LDAP_MSG_MED
- LDAP_MSG_HIGH

To force the message to the console logically on LDAP_OP_CONSOLE with the *msg_severity*, see Usage for when messages are written to the log.

subsystem Name of the plug-in subsystem in which this function is called.

fmt Message you want written. This message is in printf()-style format. Only the following printf()-style substitution codes are supported:

Table 2. printf()-style substitution codes

%d	signed integer
%ld	signed long integer
%u	unsigned integer
%lu	unsigned long integer
%x	lowercase hexadecimal unsigned integer (specify %08x or %8.8x for an 8-character value with zero-fill)
%lx	lowercase hexadecimal unsigned long integer
%X	uppercase hexadecimal unsigned integer (specify %08X or %8.8X for an 8-character value with zero-fill)
%lX	uppercase hexadecimal unsigned long integer
%p	pointer
%c	EBCDIC character
%s	EBCDIC string

The format specifications uses either the XPG4 ″%n$f″ form or the ″%f″ form, but the two forms cannot be intermixed in the same message.

Usage

1. The **slapi_log_error()** routine formats a message and writes it to the job log.
2. The message is written to the operator console depending on the setting of the environment variable LDAP_CONSOLE_LEVEL unless LDAP_OP_CONSOLE is logically ORed with the *msg_severity*. In this case, it will always be written to the operator console. The **slapi_log_error()** severity level equates to the LDAP_CONSOLE_LEVEL severity level as follows:

LDAP_MSG_LOW is an Information (I) severity level
LDAP_MSG_MED is an Attention (W) severity level
LDAP_MSG_HIGH is an Error (E) severity level

See *IBM Tivoli Directory Server Administration and Use for z/OS* for more information on the use of LDAP_CONSOLE_LEVEL, activity logging and LDAP server configuration.

3. Operator console messages must include a message identifier. The subsystem input field is used for the message identifier.
4. The message is written to the LDAP server activity log when activity logging is enabled.
5. Examples (<*Italics*> are filled in with the appropriate system and LDAP server information):
   - ```
     slapi_log_error(LDAP_MSG_MED,
      "GLD1004I","LDAP server is ready for requests.\n" );
     ```

     Writes the following message to the job log:

     ```
     <date time>  GLD1004I LDAP server is ready for requests.
     ```
   - ```
     slapi_log_error ( LDAP_MSG_HIGH,
      "GLD1059I", "Listening for requests on %s port %d.\n", ip,port );
     ```

     where

     ```
     LDAP_CONSOLE_LEVEL=E
     ip is the string "127.0.0.1"
     port = 386
     ```

     Writes the following message to the job log:

     ```
     <date time> GLD1059I Listening for requests on 127.0.0.1 port 386.
     ```

     The same message is written to the operator console, depending on how your console is configured. The date and time is excluded.
   - ```
     slapi_log_error ( LDAP_MSG_LOW | LDAP_OP_CONSOLE,
      "GLD1005I", "LDAP server start command processed.\n" );
     ```

     Writes the following message to the job log:

     ```
     <date time> GLD1005I LDAP server start command processed.
     ```

     The same message is written to the operator console, depending on how your console is configured. The date and time is excluded.

Related topics

None.
     ```

slapi_modify_internal()

Purpose

Issue a modify request.

Format

```
#include <slapi-plugin.h>

Slapi_PBlock * slapi_modify_internal (
  const char *          dn,
  LDAPMod **            mods,
  LDAPControl **        controls,
  int                   l)
```

Parameters

Input

dn The distinguished name of the entry.

mods A NULL-terminated array of modifications. The attribute value is specified as a `BerVal` structure if the LDAP_MOD_BVALUES flag is set and is specified as a character string if it is not set.

controls A NULL-terminated array of server controls for the MODIFY request. Specify NULL if there are no server controls.

l This parameter is not used and is set to 0. It is included for compatibility with other LDAP implementations.

Usage

The **slapi_modify_internal()** routine issues a MODIFY request and returns the results to the plug-in for processing. The LDAP Version 3 protocol and the current client authentication is used for the MODIFY request. The request is unauthenticated if a client request is not being processed. Call the **slapi_pblock_get()** routine to obtain the results from the returned parameter block. The following values can be retrieved from the parameter block:

- `SLAPI_PLUGIN_INTOP_RESULT` - The result code from the result message.
- `SLAPI_PLUGIN_INTOP_ERRMSG` - The error message from the result message.
- `SLAPI_PLUGIN_INTOP_MATCHED_DN` - The matched DN from the result message.
- `SLAPI_PLUGIN_INTOP_REFERRALS` - The referrals from the result message.

Related topics

The function return value is the address of a plug-in parameter block or NULL if the MODIFY request is not issued. Call the **slapi_pblock_destroy()** routine to release the plug-in parameter block when it is no longer needed. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
EIO	Unable to process the MODIFY request
ENOMEM	Insufficient storage is available

slapi_modrdn_internal()

Purpose

Issue a MODIFY-DN request.

Format

```
#include <slapi-plugin.h>

Slapi_PBlock * slapi_modrdn_internal (
  const char *          dn,
  const char *          newrdn,
  int                   deloldrdn,
  LDAPControl **        controls,
  int                   l)
```

Parameters

Input

dn The distinguished name of the entry.

newrdn The new RDN for the entry.

deloldrdn Specify 1 if the old RDN is to be deleted and 0 if the old RDN is not to be deleted.

controls A NULL-terminated array of server controls for the MODIFY-DN request. Specify NULL if there are no server controls.

l The parameter is not used and should be set to 0. It is included for compatibility with other LDAP implementations.

Usage

The **slapi_modrdn_internal()** routine issues a MODIFY-DN request and returns the results to the plug-in for processing. The LDAP Version 3 protocol and the current client authentication is used for the MODIFY-DN request. The request is unauthenticated if a client request is not being processed. Call the **slapi_pblock_get()** routine to obtain the results from the returned parameter block. The following values can be retrieved from the parameter block:

- `SLAPI_PLUGIN_INTOP_RESULT` - The result code from the result message.
- `SLAPI_PLUGIN_INTOP_ERRMSG` - The error message from the result message.
- `SLAPI_PLUGIN_INTOP_MATCHED_DN` - The matched DN from the result message.
- `SLAPI_PLUGIN_INTOP_REFERRALS` - The referrals from the result message.

Related topics

The function return value is the address of a plug-in parameter block or NULL if the MODIFY-DN request is not issued. Call the **slapi_pblock_destroy()** routine to release the plug-in parameter block when it is no longer needed. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
EIO	Unable to process the MODIFY-DN request
ENOMEM	Insufficient storage is available

slapi_op_abandoned()

Purpose

Check if the current request has been abandoned.

Format

```
#include <slapi-plugin.h>

int slapi_op_abandoned (
  Slapi_PBlock *        pb)
```

Parameters

Input

pb The plug-in parameter block.

Usage

The **slapi_op_abandoned()** routine checks if the client has abandoned the current request.

Related topics

The function return value is 1 if the request is abandoned, 0 if the request is not abandoned, and -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1

EINVAL	A parameter is not valid
EPERM	There is no client request

slapi_pblock_destroy()

Purpose

Release a plug-in parameter block returned for an internal request.

Format

```
#include <slapi-plugin.h>

void slapi_pblock_destroy (
  Slapi_PBlock *         pb)
```

Parameters

Input

pb The plug-in parameter block.

Usage

The **slapi_pblock_destroy()** routine releases a plug-in parameter block returned by an internal request routine, such as **slapi_add_internal()**. This routine must not be used to release a plug-in parameter block supplied as input to a plug-in callback routine.

Related topics

There is no function return value.

slapi_pblock_get()

Purpose

Retrieve a value from the plug-in parameter block.

Format

```
#include <slapi-plugin.h>

  int slapi_pblock_get (
  Slapi_PBlock *         pb,
  int                    arg,
  void *                 value)
```

Parameters

Input

pb The plug-in parameter block.

arg The parameter value to be retrieved.

Output

value The address of a variable that will be set to the parameter value.

Usage

The specified parameter value is retrieved from the plug-in parameter block. The plug-in must not modify or release any of the values returned by the **slapi_pblock_get()** routine. For SLAPI_PLUGIN_PRIVATE and SLAPI_CONN_PRIVATE, the parameter value is an address that is saved in the plug-in parameter block and can be freed. EINVAL is returned if the parameter type or value is not valid while EPERM is returned if the parameter type is not allowed for the current plug-in invocation.

These parameter types are valid only for a parameter block returned by an internal request routine:

```
SLAPI_PLUGIN_INTOP_REFERRALS
SLAPI_PLUGIN_INTOP_RESULT
SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES
SLAPI_PLUGIN_INTOP_SEARCH_REFERRALS
```

The other parameter types are not valid for an internal request parameter block.

Table 3. Operational parameters

Name	Format	Usage
SLAPI_PLUGIN_ARGC	int	The number of arguments specified on the **plugin** configuration statement.
SLAPI_PLUGIN_ARGV	char **	A NULL-terminated array of arguments specified on the **plugin** configuration statement. See SLAPI_PLUGIN_ARGC for the number of arguments.
SLAPI_PLUGIN_CTLLIST	char **	An array of server control object identifiers registered by the current plug-in. The value is NULL if there are no server controls.
SLAPI_PLUGIN_DB_SUFFIXES	char **	A NULL-terminated array of database suffixes registered for the current plug-in. The value is NULL if there are no database suffixes registered. The database suffixes are normalized as determined by the LDAP server schema.

Table 3. Operational parameters (continued)

Name	Format	Usage
SLAPI_PLUGIN_EXT_OP_OIDLIST	char **	A NULL-terminated array of extended operation object identifiers registeredfor the current plug-in. The value is NULL if there are no object identifiers registered.
SLAPI_PLUGIN_PRIVATE	void *	Private value set by the **slapi_pblock_set() routine**. Each plug-in can have its own private value and must be freed on termination.
SLAPI_PLUGIN_TYPE	int	Current plug-in type: SLAPI_PLUGIN_PREOPERATION SLAPI_PLUGIN_CLIENTOPERATION SLAPI_PLUGIN_POSTOPERATION

Table 4. General request parameters

Name	Format	Usage
SLAPI_CONN_ID	unsigned long	Client connection identifier. Connection identifiers are reused when a connection is closed. The plug-in registers a SLAPI_PLUGIN_CLOSE_FN if it needs to be informed when a client connection is closed.
SLAPI_CONN_PRIVATE	void *	Private value for the current connection. Each plug-in can have its own set of private connection values and must be freed on termination. The value is NULL if the plug-in has not set a private value for the connection.
SLAPI_CONN_VERSION	int	The LDAP protocol version for the connection. This is the previous protocol version while processing a BIND request (use the SLAPI_BIND_VERSION parameter to obtain the protocol version specified in the BIND request)
SLAPI_REQCONTROLS	LDAPControl **	A NULL-terminated array of server controls specified in the request. The value is NULL if there are no controls.
SLAPI_REQUEST_ID	unsigned int	Message identifier for the current client request.
SLAPI_REQUESTOR_ALT_NAMES	char **	A NULL-terminated array of normalized alternate names for the authentication DN. The value is NULL if there are no alternate names.
SLAPI_REQUESTOR_DN	char *	Authenticated DN of the client requesting the operation. A zero-length string is returned if the client is not authenticated.
SLAPI_REQUESTOR_GROUPS	char **	A NULL-terminated array of normalized group names for the authentication DN. The value is NULL if the authentication DN is not a member of any groups or if group gathering was not enabled for the BIND request.

Table 4. General request parameters (continued)

Name	Format	Usage
SLAPI_REQUESTOR_IS_ADMIN	int	The value is 1 if the requestor is the LDAP administrator. Otherwise, the value is 0.
SLAPI_REQUESTOR_NORM_DN	char *	Normalized authenticated DN of the client requesting the operation. A zero-length string is returned if the client is not authenticated.
SLAPI_REQUESTOR_SAFID	char *	SAF user ID of the bound client. A zero-length string is returned when no SAF user ID is associated with the client. The value is uppercased and in local code page.
SLAPI_REQUESTOR_SECURITY_LABEL	char *	The security label associated with the client requesting the operation. The security label is returned as a local code page string. A zero-length string is returned when the client is not authenticated or when LDAP server security label processing is not configured for client operations.
SLAPI_TARGET_DN	char *	Target DN specified in the current request. The value is NULL if the request does not have a target DN.

Table 5. ABANDON request parameters

Name	Format	Usage
SLAPI_ABANDON_MSGID	unsigned int	Message identifier of the message is abandoned.

Table 6. ADD request parameters

Name	Format	Usage
SLAPI_ADD_ENTRY	Slapi_Entry *	Entry to be added. The server will create SLAPI_ENTRY whenever an add is requested. This function returns the address of the entry. This entry will be freed at the end of the request.
SLAPI_ADD_TARGET	char *	DN of the entry to be added. This is the same value returned by SLAPI_TARGET_DN.

Table 7. BIND request parameters

Name	Format	Usage
SLAPI_BIND_CREDENTIALS	BerVal *	Credentials from the BIND request.
SLAPI_BIND_METHOD	int	Bind method: LDAP_AUTH_SIMPLE LDAP_AUTH_SASL
SLAPI_BIND_TARGET	char *	Authentication DN from the BIND request. This is the same value returned by SLAPI_TARGET_DN.

Table 7. BIND request parameters (continued)

Name	Format	Usage
SLAPI_BIND_VERSION	int	The LDAP protocol version from the BIND request.

Table 8. COMPARE request parameters

Name	Format	Usage
SLAPI_COMPARE_TARGET	char *	DN of the entry to be used for the comparison. This is the same value returned by SLAPI_TARGET_DN.
SLAPI_COMPARE_TYPE	char *	Attribute type to be used for the comparison. The lowercased primary attribute name is returned as defined in the LDAP schema.
SLAPI_COMPARE_VALUE	BerVal *	Attribute value to be used for the comparison. The normalized attribute value is returned if the attribute type has an equality matching filter, otherwise the unnormalized attribute value is returned.

Table 9. DELETE request parameters

Name	Format	Usage
SLAPI_DELETE_TARGET	char *	DN of the entry to be deleted. This is the same value returned by SLAPI_TARGET_DN.

Table 10. EXTENDED OPERATION request parameters

Name	Format	Usage
SLAPI_EXT_OP_REQ_OID	char *	Extended operation object identifier.
SLAPI_EXT_OP_REQ_VALUE	BerVal *	Extended operation value.

Table 11. MODIFY request parameters

Name	Format	Usage
SLAPI_MODIFY_MODS	LDAPMod **	A NULL-terminated array of modifications to be performed. The attribute values is represented as binary values in the LDAPMod entries (modv_bvals is used instead of modv_strvals and the LDAP_MOD_BVALUES flag is set).
SLAPI_MODIFY_TARGET	char *	DN of the entry to be modified. This is the same value returned by SLAPI_TARGET_DN.

Table 12. MODIFY DN request parameters

Name	Format	Usage
SLAPI_MODRDN_DELOLDRDN	int	1 if the old RDN is to be deleted, 0 if the old RDN is not to be deleted.
SLAPI_MODRDN_NEWRDN	char *	New RDN for the entry.

Table 12. MODIFY DN request parameters (continued)

Name	Format	Usage
SLAPI_MODRDN_NEWSUPERIOR	char *	DN of the new superior entry. The value is NULL if a new superior entry is not specified in the MODIFY DN request.
SLAPI_MODRDN_TARGET	char *	New DN for the renamed entry. This is the same value returned by SLAPI_TARGET_DN.

Table 13. SEARCH request parameters

Name	Format	Usage
SLAPI_SEARCH_ATTRS	char **	A NULL-terminated array of attribute types from the search request. The value is NULL if there are no attribute types in the search request. The attribute names are the lowercased primary attribute names as defined in the LDAP schema.
SLAPI_SEARCH_ATTRSONLY	int	1 if only attribute types are to be returned, 0 if attribute types and values are to be returned.
SLAPI_SEARCH_DEREF	int	Alias dereferencing: LDAP_DEREF_NEVER LDAP_DEREF_FINDING LDAP_DEREF_SEARCHING LDAP_DEREF_ALWAYS
SLAPI_SEARCH_FILTER	Slapi_Filter *	Search filter.
SLAPI_SEARCH_SCOPE	int	Search scope: LDAP_SCOPE_BASE LDAP_SCOPE_ONELEVEL LDAP_SCOPE_SUBTREE
SLAPI_SEARCH_SIZELIMIT	int	Search size limit. This is the smaller of the size limit from the search request and the size limit specified in the LDAP server configuration file. The configured size limit should be ignored for the LDAP administrator.
SLAPI_SEARCH_TARGET	char *	DN of the base entry for the search. This is the same value returned by SLAPI_TARGET_DN.
SLAPI_SEARCH_TIMELIMIT	int	Search time limit. This is the smaller of the time limit from the search request and the time limit specified in the LDAP server configuration file. The configured time limit is ignored for the LDAP administrator.

Table 14. Callback parameters

Name	Format	Usage
SLAPI_CALLBACK_NAME	char *	The normalized name for the callback request. The value is NULL if there is no name associated with the callback request.

Table 14. Callback parameters (continued)

Name	Format	Usage
SLAPI_CALLBACK_TYPE	int	The callback type. SLAPI_TYPE_DN_PW to obtain the password for a distinguished name SLAPI_TYPE_UID_PW to obtain the password for a user name SLAPI_TYPE_GROUPS to obtain the group list for a distinguished name SLAPI_TYPE_ALT_NAMES to obtain the Kerberos alternate names

Table 15. General result parameters

Name	Format	Usage
SLAPI_PLUGIN_OPRETURN	int	The result code for the current operation. The result code is set by the **slapi_send_ldap_result()** routine.

Table 16. Internal request result parameters

Name	Format	Usage
SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES	Slap_Entry **	A NULL-terminated array of search entries returned for an internal search request. The value is NULL if there are no search entries.
SLAPI_PLUGIN_INTOP_SEARCH_REFERRALS	char *	A NULL-terminated array of search references returned for an internal search request. The value is NULL if there are no search references.
SLAPI_PLUGIN_INTOP_ERRMSG	char *	Error message returned in the result message for an internal request. The value is NULL if there is no error message.
SLAPI_PLUGIN_INTOP_MATCHED_DN	char *	Matched DN returned in the result message for an internal request. The value is NULL if there is no matched DN.
SLAPI_PLUGIN_INTOP_REFERRALS	char *	A NULL-terminated array of referrals returned in the result message for an internal request. The value is NULL if there are no referrals.
SLAPI_PLUGIN_INTOP_RESULT	int	Result code returned in the result message for an internal request.

Related topics

The return value is 0 if the request is successful and -1 if there is an error. The *errno* variable is set to one of the following values when the function return value is -1:

EFAULT	Value address is not valid
EINVAL	A parameter is not valid
ENOENT	Value does not exist
ENOMEM	Insufficient storage is available

EPERM	Insufficient storage is available

slapi_pblock_set()

Purpose

Format

```
#include <slapi-plugin.h>

  int slapi_pblock_set (
  Slapi_PBlock *        pb,
  int                   arg,
  void *                value)
```

Parameters

Input

pb The plug-in parameter block.

arg The parameter value to be set.

value The address of the parameter value or, for a registration parameter, the callback function.

Usage

The specified parameter value is set in the plug-in parameter block. The plug-in must release any storage allocated for the parameter value since the **slapi_pblock_set()** routine makes a copy of the parameter value before returning. For SLAPI_PLUGIN_PRIVATE and SLAPI_CONN_PRIVATE, the parameter value is an address that is saved in the plug-in parameter block. EINVAL is returned if the parameter type or value is not valid while EPERM is returned if the parameter type is not allowed for the current plug-in invocation.

Suffixes, extended operations and controls can only be set during initialization.

Table 17. Registration parameters

Name	Format	Usage
SLAPI_PLUGIN_ABANDON_FN	int (*)(Slapi_PBlock *)	Routine to process a client ABANDON request.
SLAPI_PLUGIN_ADD_FN	int (*)(Slapi_PBlock *)	Routine to process a client ADD request.
SLAPI_PLUGIN_BIND_FN	int (*)(Slapi_PBlock *)	Routine to process a client BIND request.
SLAPI_PLUGIN_CALLBACK_FN	int (*)(Slapi_PBlock *)	Routine to process a server callback request. A callback routine is registered only by a client-operation plug-in.
SLAPI_PLUGIN_CLOSE_FN	void (*)(Slapi_PBlock *)	Routine to be called during LDAP server termination.
SLAPI_PLUGIN_COMPARE_FN	int (*)(Slapi_PBlock *)	Routine to process a client COMPARE request.
SLAPI_PLUGIN_DELETE_FN	int (*)(Slapi_PBlock *)	Routine to process a client DELETE request.
SLAPI_PLUGIN_DISCONNECT_FN	void (*)(Slapi_PBlock *)	Routine to be called when an LDAP client session is closed.
SLAPI_PLUGIN_EXT_OP_FN	int (*)(Slapi_PBlock *)	Routine to process a client EXTENDED OPERATION request.
SLAPI_PLUGIN_MODIFY_FN	int (*)(Slapi_PBlock *)	Routine to process a client MODIFY request.

Table 17. Registration parameters (continued)

Name	Format	Usage
SLAPI_PLUGIN_MODRDN_FN	int (*)(Slapi_PBlock *)	Routine to process a client MODIFY DN request.
SLAPI_PLUGIN_SEARCH_FN	int (*)(Slapi_PBlock *)	Routine to process a client SEARCH request.
SLAPI_PLUGIN_THREAD_FN	void (*)(Slapi_PBlock *)	Routine to be called when an LDAP server worker thread terminates.
SLAPI_PLUGIN_UNBIND_FN	int (*)(Slapi_PBlock *)	Routine to process a client UNBIND request.

Table 18. Operational parameters

Name	Format	Usage
SLAPI_CONN_PRIVATE	void *	Private value for the current connection. Each plug-in can have its own set of private connection values.
SLAPI_PLUGIN_CTLLIST	char **	NULL-terminated array of server control object identifiers supported by the current plug-in. The LDAP server accepts an unrecognized critical control if the object identifier is registered by one or more plug-ins. The plug-in is responsible for any processing required by the server control.
SLAPI_PLUGIN_DB_SUFFIX	char **	NULL-terminated array of database suffixes for the current plug-in. This parameter is set only by a client-operation plug-in.
SLAPI_PLUGIN_EXT_OP_OIDLIST	char **	NULL-terminated array of extended operation object identifiers for the current plug-in. This parameter is set only by a client-operation plug-in.
SLAPI_PLUGIN_PRIVATE	void *	Private value that is retrieved by the **slapi_pblock_get()** routine. Each plug-in has its own private value.

Table 19. Callback parameters

Name	Format	Usage
SLAPI_CALLBACK_ERRMSG	char *	An error message is returned to the LDAP client if an error occurred. Specify NULL if there is no error message.
SLAPI_CALLBACK_LIST	char **	A NULL-terminated array of names. This is the return value for a group list or alternate names callback. Specify NULL if there are no names.
SLAPI_CALLBACK_PASSWORD	char *	The user password. This is a return value for a password callback. Specify NULL if there is no password for the supplied name.

Table 19. Callback parameters (continued)

Name	Format	Usage
SLAPI_CALLBACK_STATUS	int	This is the LDAP result code for the request. It is set to LDAP_SUCCESS if the callback request was processed, LDAP_UNWILLING_TO_PERFORM if the plug-in doesn't recognize the callback type, or an LDAP error code if an error occurred. The return status is LDAP_SUCCESS if there is no password, group or alternate name for the supplied name and the appropriate return value (SLAPI_CALLBACK_LIST or SLAPI_CALLBACK_PASSWORD) is set to NULL.
SLAPI_CALLBACK_TARGET_DN	char *	The entry name associated with the password returned for the SLAPI_CALLBACK_PASSWORD parameter. This is a return value for a password callback. Specify NULL if there is no entry.

Table 20. General result parameters

Name	Format	Usage
SLAPI_RETCONTROLS	LDAPControl **	A NULL-terminated array of controls is returned in the result message. This parameter may be set only by a client-operation plug-in.

Table 21. EXTENDED OPERATION result parameters

Name	Format	Usage
SLAPI_EXT_OP_RET_OID	char *	Extended operation object identifier.
SLAPI_EXT_OP_RET_VALUE	BerVal *	Extended operation value.

Related topics

The return value is 0 if the request is successful and -1 if there is an error. The *errno* variable is set to one of the following values when the function return value is -1:

EEXIST	Value already exists
EFAULT	Value address is not valid
EINVAL	A parameter is not valid
ENOMEM	Insufficient storage is available
EPERM	A parameter is not allowed

slapi_search_internal()

Purpose

Issue a SEARCH request.

Format

```
#include <slapi-plugin.h>

Slapi_PBlock * slapi_search_internal (
 const char *         base,
 int                  scope,
 const char *         filter,
 LDAPControl **       controls,
 char **              attrs,
 int                  attrsonly)
```

Parameters

Input

base The base DN for the search.

scope The scope for the search must be:
LDAP_SCOPE_BASE
LDAP_SCOPE_ONELEVEL
LDAP_SCOPE_SUBTREE

filter The filter for the search. The filter is set to (`objectClass=*`) if NULL is specified for this parameter.

controls A NULL-terminated array of server controls for the SEARCH request. Specify NULL if there are no server controls.

attrs A NULL-terminated array of attributes is returned for the search entries. Specify NULL if all attributes are returned. Note that operational attributes are returned only if they are explicitly listed in the `attrs` parameter.

attrsonly Specify 1 if just the attribute types are to be returned or 0 if both attribute types and attribute values are to be returned.

Usage

The **slapi_search_internal()** routine issues a SEARCH request and returns the results to the plug-in for processing. The LDAP Version 3 protocol and the current client authentication are used for the SEARCH request. The request is unauthenticated if a client request is not being processed. Call the **slapi_pblock_get()** routine to obtain the search results from the returned parameter block. The following values can be retrieved from the parameter block:

- `SLAPI_PLUGIN_INTOP_RESULT` - The result code from the result message
- `SLAPI_PLUGIN_INTOP_ERRMSG` - The error message from the result message
- `SLAPI_PLUGIN_INTOP_MATCHED_DN` - The matched DN from the result message
- `SLAPI_PLUGIN_INTOP_REFERRALS` - The referrals from the result message
- `SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES` - The search entries
- `SLAPI_PLUGIN_INTOP_SEARCH_REFERRALS` - The search references

Related topics

The function return value is the address of the plug-in parameter block or NULL if the SEARCH request is not issued. Call the **slapi_pblock_destroy()** routine to release the plug-in parameter block when it is no longer needed. The *errno* variable is set to one of the following values when the function return value is NULL:

EINVAL	A parameter is not valid
EIO	Unable to process the SEARCH request
ENOMEM	Insufficient storage is available

slapi_send_ldap_referral()

Purpose

Send an LDAP search referral message to the client

Format

```
#include <slapi-plugin.h>

int slapi_send_ldap_referral (
 Slapi_PBlock *         pb,
 Slapi_Entry *          entry,
 BerVal **              refs,
 BerVal ***             urls)
```

Parameters

Input

pb The plug-in parameter block.

entry The directory entry containing the referrals. The entry name is used if a referral value does not already contain a distinguished name. NULL is specified for this parameter if the referral values are complete and do not need to have the distinguished name added.

refs The referral values from the directory entry.

Input/Output

urls
This variable points to a NULL-terminated array of referral urls for LDAP Version 2 clients. The variable is initialized to NULL before the first call to the **slapi_send_ldap_referral()** routine for the current search request. If the client is using the LDAP Version 2 protocol, the **slapi_send_ldap_referral()** routine allocates and expands this array to contain the new referral `urls`. Call the **slapi_ch_free_values()** routine to release the array when it is no longer needed. NULL is specified for this parameter if the client is using the LDAP Version 3 protocol.

Usage

The **slapi_send_ldap_referral()** routine processes a referral entry that is within the scope of a search request. The **slapi_send_ldap_result()** routine is called with a result code of LDAP_PARTIAL_RESULTS (LDAP Version 2) or LDAP_REFERRAL (LDAP Version 3) if the base entry for the search is a referral entry. The **slapi_send_ldap_referral()** routine is called only by a pre-operation or client-operation plug-in.

If the client is using the LDAP Version 3 protocol, a search referral message is created and sent to the client. The `urls` parameter is not used in this case.

If the client is using the LDAP Version 2 protocol, the referral urls are accumulated using the `urls` parameter. Upon completion of the search request, the application calls the **slapi_send_ldap_result()** routine with a result code of LDAP_PARTIAL_RESULTS and provide the referral `urls`. The referral array is freed by calling the **slapi_ch_free_values()** routine.

The referral `urls` are modified based on the directory entry name and the search scope. The directory name is used for the distinguished name if the referral url does not contain a distinguished name. The referral scope is base if the search scope is one-level and the referral scope is sub if the search scope is sub. The **slapi_send_ldap_referral()** routine is not called if the search scope is base since the referral is returned in the LDAP result message and not as an LDAP search referral message.

Related topics

The function return value is 0 if the referrals have been processed or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

EINVAL	A parameter is not valid
EIO	Unable to send the message
ENOMEM	Insufficient storage is available
EPERM	The plug-in is not a pre-operation or client-operation plug-in or the current request is not a search request

slapi_send_ldap_result()

Purpose

Send the LDAP result message to the client.

Format

```
#include <slapi-plugin.h>

void slapi_send_ldap_result (
 Slapi_PBlock *         pb,
 int                    resultCode,
 char *                 matchedDN,
 char *                 errorText,
 int                    numEntries,
 BerVal **              referrals)
```

Parameters

Input

pb The plug-in parameter block.

resultCode The result code to be returned to the client.

matchedDN The matched DN returned to the client. Specify NULL for this parameter if no matched DN is returned. A matched DN must not be specified unless the result code is:
LDAP_ALIAS_DEREF_PROBLEM
LDAP_ALIAS_PROBLEM
LDAP_INVALID_DN_SYNTAX
LDAP_NO_SUCH_OBJECT

errorText The error text returned to the client. Specify NULL for this parameter if no error text is returned. Error text is not specified if the result code is LDAP_SUCCESS.

numEntries The number of search entries returned for the current search request. This parameter is specified as 0. This parameter is obsolete and is included for compatibility with other LDAP implementations.

referrals A NULL-terminated array of referral URLs returned to the client. Specify NULL for this parameter if no referrals are returned to the client. For the LDAP Version 3 protocol, referrals must not be specified unless the result code is LDAP_REFERRAL. For the LDAP Version 2 protocol, referrals are specified for any result code other than LDAP_SUCCESS (LDAP Version 2 referrals are appended to the error text).

Usage

The **slapi_send_ldap_result()** routine sends an LDAP result message to the LDAP client. Only one result message is returned for each LDAP request. The **slapi_send_ldap_result()** routine is called only by a pre-operation or client-operation plug-in. The **slapi_pblock_set()** routine can be called before calling the **slapi_send_ldap_result()** routine if the result message includes server controls, an extended result object identifier or an extended result value.

Related topics

There is no function return value.

slapi_send_ldap_search_entry()

Purpose

Send an LDAP search entry message to the client.

Format

```
#include <slapi-plugin.h>

int slapi_send_ldap_search_entry (
 Slapi_PBlock *          pb,
 Slapi_Entry *           entry,
 LDAPControl **          controls,
 char **                 attrs,
 int                     attrsonly)
```

Parameters

Input

pb The plug-in parameter block.

entry The directory entry.

controls A NULL-terminated array of LDAP controls returned with the search entry message. Specify NULL for the array address if no controls should be returned.

attrs A NULL-terminated array of attribute types returned in the search entry message. Specify NULL for the array address if all attributes are returned. Specify an array with just the NULL entry if no attributes are returned. Operational attributes are returned only if they are explicitly specified.

attrsonly Specify 1 to return only the attribute types and 0 to return the attribute types and values.

Usage

The **slapi_send_ldap_search_entry()** routine sends an LDAP search entry message to the LDAP client. The **slapi_send_ldap_search_entry()** routine is called only by a pre-operation or client-operation plug-in. The **slapi_send_ldap_search_entry()** routine is called for each directory entry that matches the search parameters. The **slapi_send_ldap_referral()** routine is called to return a search referral message to the client.

Related topics

The function return value is 0 if the search entry message is sent or -1 if an error occurred. The *errno* variable is set to one of the following values when the function return value is -1:

ECANCELED	Client has cancelled the request
EINVAL	A parameter is not valid
EIO	Unable to send the message
ENOMEM	Insufficient storage is available
EPERM	The plug-in is not a pre-operation or client-operation plug-in or the current request is not a search request
ESRCH	Attribute type is not defined in LDAP schema

slapi_trace()

Purpose

Writes an LDAP server trace message.

Format

```
#include <slapi-plugin.h>

void * slapi_trace (
   long long     traceLevel,
   char *        subsystem,
   char *        fmt, ... )
```

Parameters

Input

traceLevel Trace level of the message. Trace level must be one of the following:

- LDAP_DEBUG_ACL
- LDAP_DEBUG_ARGS
- LDAP_DEBUG_BE_CAPABILITIES
- LDAP_DEBUG_BER
- LDAP_DEBUG_CACHE
- LDAP_DEBUG_CONNS
- LDAP_DEBUG_ERROR
- LDAP_DEBUG_FILTER
- LDAP_DEBUG_INFO
- LDAP_DEBUG_LDAPBE
- LDAP_DEBUG_LDBM
- LDAP_DEBUG_MESSAGE
- LDAP_DEBUG_MULTISERVER
- LDAP_DEBUG_PACKETS
- LDAP_DEBUG_PERFORMANCE
- LDAP_DEBUG_PLUGIN
- LDAP_DEBUG_REFERRAL
- LDAP_DEBUG_REPL
- LDAP_DEBUG_SCHEMA
- LDAP_DEBUG_SDBM
- LDAP_DEBUG_STATS
- LDAP_DEBUG_STRBUF
- LDAP_DEBUG_SYSPLEX
- LDAP_DEBUG_TDBM
- LDAP_DEBUG_THREAD
- LDAP_DEBUG_TRACE

The trace level can be combined with (logical or) LDAP_USE_CTRACE, to write the message using the LDAP server CTRACE in-memory tracing.

fmt, ... Message you want traced. This message can be in printf()-style format. Only the following printf()-style substitution codes are supported:

Table 22. printf()-style substitution codes

%d	signed integer
%ld	signed long integer
%u	unsigned integer
%lu	unsigned long integer
%x	lowercase hexadecimal unsigned integer (specify %08x or %8.8x for an 8-character value with zero-fill)
%lx	lowercase hexadecimal unsigned long integer
%X	uppercase hexadecimal unsigned integer (specify %08X or %8.8X for an 8-character value with zero-fill)
%lX	uppercase hexadecimal unsigned long integer
%p	pointer
%c	EBCDIC character
%s	EBCDIC string

Usage

1. The **slapi_trace()** routine formats a message and uses either the LDAP server debug trace functions or the CTRACE in-memory trace functions to write the message.
2. When initially writing a plug-in, you should use the LDAP_DEBUG_PLUGIN trace level. As the complexity of the plug-in grows, use the other trace levels to refine or reduce LDAP server trace output.
3. See *IBM Tivoli Directory Server Administration and Use for z/OS* for more information on LDAP server debug level tracing and CTRACE in-memory tracing in the *Running the LDAP server* chapter.
4. The message will be written using the LDAP server CTRACE in-memory trace functions by combining LDAP_USE_CTRACE with the **slapi_trace()** trace level.
5. Examples (<*Italics*> are filled in with the appropriate system and LDAP server information):
   - `slapi_trace ( LDAP_DEBUG_PLUGIN, "MyPLUG", "Attempting to read data." );`

     When LDAP server debugging is enabled and the debug level includes PLUGIN, formats, and traces the message:

     ```
     <date time>(<thread info>) PLUGIN:MyPLUG: <function name>: Attempting to read data.
     ```
   - `slapi_trace ( LDAP_DEBUG_TRACE, ThisPLUG, "%d data bytes were read.", bytesIn );`

     When LDAP server debugging is enabled and the debug level includes TRACE, formats, and traces the message:

     ```
     <date time>(<thread info>) TRACE: ThisPLUG: <function name>:<value of bytesIn> data bytes were read.
     ```
   - `slapi_trace ( LDAP_DEBUG_PLUGIN | LDAP_USE_CTRACE, "PLUG", "I'm at this point." );`

     When LDAP server debugging is enabled and the debug level includes PLUGIN, formats, and traces the message using CTRACE in-memory tracing:

     ```
     <date time>(<thread info>) PLUGIN:PLUG: <function name>: I'm at this point.
     ```

Related topics

None.

Appendix A. Plug-in sample

The sample plug-in and its makefile are located in **/usr/lpp/ldap/examples**.

The sample plug-in, **/usr/lpp/ldap/examples/plugin_sample.c** creates a post-operation plug-in that logs LDAP server BIND requests and results codes to a specified file. The specified file is an input parameter to the sample plug-in.

The makefile, **/usr/lpp/ldap/examples/makefile.plugin** can be used to build **plugin_sample.c**.

Steps for building and running a sample plug-in

How to build and run a sample plug-in:

1. Start by creating either a PDS or a PDSE dataset with the same attributes as SYS1.SIEALNKE. A PDSE dataset is required when building the plug-in sample as a 64-bit module.
2. APF authorize the dataset created.
3. Ensure the dataset is in the load list for the LDAP server, either through a STEPLIB statement or the system LNKLST.
4. Edit **/usr/lpp/ldap/examples/makefile.plugin** and update **PLUGSAMP_DLL** with the name of the dataset you created. For example:

   ```
   PLUGSAMP_DLL = "//'GLD.PLUGIN.SIEALNKE(PLUGSAMP)'"
   ```

   Also, if you are building a 64-bit DLL, set **PLUGSAMP_ADDR_MODE** to 64.
5. Save **makefile.plugin**
6. To compile and linkedit the sample plug-in using the **makefile.plugin**, enter **make -f makefile.plugin**.
7. Verify that no build or link errors occurred. Verify that your dataset now contains the member **PLUGSAMP**, or a member with the name you updated.
8. Stop the server.
9. Edit the LDAP server configuration file and add the **plugin** configuration option to the global section:

   ```
   plugin postOperation PLUGSAMP plugin_init "logFilename"
   ```

   where, `"logFilename"` is the name of the file you wish to have the log records written to, and it must be in double quotes.
10. If you are building a 64-bit DLL, then add the **plugin** configuration option in the following format:

    ```
    plugin postOperation PLUGSM31/PLUGSAMP plugin_init "logFilename"
    ```

    Note: For this 64-bit example, it is assumed **PLUGSAMP** is the name used when the 64-bit DLL was built, as shown above. The name **PLUGSM31** is a place holder name for the **plugin** configuration option. It can be any name and no DLL with that name needs to exist.

    See *IBM Tivoli Directory Server Administration and Use for z/OS*, *Chapter 8. Customizing the LDAP server configuration* , for a complete description of the **plugin** configuration option and its parameters.
11. Restart the LDAP server.

    If you use the debug parameter **PLUGIN**, sample plug-in trace messages will be written to the LDAP server job log. For example:

    ```
    START LDAPSRV,PARMS='-d PLUGIN'
    ```

    where, **LDAPSRV** is an example name and represents the name of your LDAP server start-up procedure.

Once started, browse your LDAP server job log for plug-in initialization and trace messages. Also, the sample plug-in creates an empty log file. Verify it was created.

To test, perform an LDAP operation binding to the LDAP server. The sample plug-in will write a message to the log including the result code of the bind operation and the bind DN. For example:

```
Result: 0 DN: o=your company
```

See Chapter 2, “Building an LDAP server plug-in” for more information on building and writing a z/OS LDAP server plug-in.

Appendix B. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

`http://www.ibm.com/systems/z/os/zos/bkserv/`

Notices

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

IBM Tivoli Directory Server Plug-in Reference for z/OS primarily documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/OS LDAP.

IBM Tivoli Directory Server Plug-in Reference for z/OS also documents information that is not intended to be used as Programming Interfaces of z/OS LDAP. This information is identified where it occurs with an introductory statement to a topic.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS™, HCD, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- IBMLink
- Tivoli
- z/OS

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Kerberos is a trademark of the Massachusetts Institute of Technology (MIT).

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be the trademarks or service marks of others.

Bibliography

This bibliography provides a list of publications that are useful when using the LDAP programming interface:

- *IBM Tivoli Directory Server Administration and Use for z/OS*, SC23-5191
- *IBM Tivoli Directory Server Client Programming for z/OS*
- *z/OS Collection*, SK3T-4269
- *z/OS Communications Server: IP Configuration Guide*, SC31-8775
- *z/OS Cryptographic Services System SSL Programming*, SC24-5901
- *z/OS XL C/C++ Programming Guide*, SC09-4765
- *z/OS DCE Application Development Guide: Directory Services*, SC24-5906
- *z/OS Information Roadmap*, SA22-7500
- *z/OS Integrated Security Services Network Authentication Service Administration*, SC24-5926
- *z/OS Language Environment Customization*, SA22-7564
- *z/OS UNIX System Services Command Reference*, SA22-7802

Index

A
ABANDON request parameters 64
accessibility 83
ADD request parameters 64

B
bibliography 87
BIND request parameters 64
books
  related 87
Building an LDAP plug-in 3

C
Callback parameters 66, 70
client plug-in 5
COMPARE request parameters 65
conventions in this document vii

D
DELETE request parameters 65
disability 83

E
EXTENDED OPERATION request parameters 65
EXTENDED OPERATION result parameters 71

G
General request parameters 63
General result parameters 67, 71

I
interface
  programming interface information 86
Internal request result parameters 67
Introduction 1

K
keyboard 83

M
MODIFY DN request parameters 65
MODIFY request parameters 65

N
Notices 85

O
Operation plug-in 5
Operational parameters 62, 70

P
Plug-in sample 81
Plug-in supported APIs
  APIs 9
post-operation plug-in 5
pre-operation plug-in 5
programming interface information 86
publications
  related 87

R
Registration parameters 69
routines 11
  slapi_add_internal() 10
  slapi_attr_get_numvalues() 12
  slapi_attr_get_type() 13
  slapi_attr_get_values() 14
  slapi_attr_value_cmp() 15
  slapi_ch_calloc() 16
  slapi_ch_free_values() 18
  slapi_ch_free() 17
  slapi_ch_malloc() 19
  slapi_ch_realloc() 20
  slapi_ch_strdup() 21
  slapi_compare_internal() 22
  slapi_control_present() 23
  slapi_delete_internal() 24
  slapi_dn_ignore_case_v3() 25
  slapi_dn_isparent() 27
  slapi_dn_normalize_case_v3() 29
  slapi_dn_normalize_v3() 28
  slapi_entry_add_value() 31
  slapi_entry_add_values() 32
  slapi_entry_alloc() 33
  slapi_entry_attr_delete() 34
  slapi_entry_attr_find() 35
  slapi_entry_delete_value() 36
  slapi_entry_delete_values() 37
  slapi_entry_dup() 38
  slapi_entry_first_attr() 39
  slapi_entry_free() 40
  slapi_entry_get_dn() 41
  slapi_entry_merge_value() 42
  slapi_entry_merge_values() 43
  slapi_entry_next_attr() 44
  slapi_entry_replace_value() 45
  slapi_entry_replace_values() 46
  slapi_entry_schema_check() 47
  slapi_entry_set_dn() 48
  slapi_filter_get_attribute_type() 49
  slapi_filter_get_ava() 50

routines *(continued)*
slapi_filter_get_choice() 51
slapi_filter_get_subfilt() 52
slapi_filter_list_first() 53
slapi_filter_list_next() 54
slapi_isSDBM_authenticated() 55
slapi_log_error() 56
slapi_modify_internal() 58
slapi_modrdn_internal() 59
slapi_op_abandoned() 60
slapi_pblock_destroy() 61
slapi_pblock_get() 62
slapi_pblock_set() 69
slapi_search_internal() 72
slapi_send_ldap_referral() 74
slapi_send_ldap_result() 76
slapi_send_ldap_search_entry() 77
slapi_trace() 78

S

SEARCH request parameters 66
shortcut keys 83
slapi_add_internal 10
slapi_attr_get_normalized_values 11
slapi_attr_get_numvalues 12
slapi_attr_get_type 13
slapi_attr_get_values 14
slapi_attr_value_cmp 15
slapi_ch_calloc 16
slapi_ch_free 17
slapi_ch_free_values 18
slapi_ch_malloc 19
slapi_ch_realloc 20
slapi_ch_strdup 21
slapi_compare_internal 22
slapi_control_present 23
slapi_delete_internal 24
slapi_dn_ignore_case_v3 25
slapi_dn_isparent() 27
slapi_dn_normalize_case_v3 29
slapi_dn_normalize_v3 28
slapi_entry_add_value 31
slapi_entry_add_values 32
slapi_entry_alloc 33
slapi_entry_attr_delete 34
slapi_entry_attr_find 35
slapi_entry_delete_value 36
slapi_entry_delete_values 37
slapi_entry_dup 38
slapi_entry_first_attr 39
slapi_entry_free 40
slapi_entry_get_dn 41
slapi_entry_merge_value 42
slapi_entry_merge_values 43
slapi_entry_next_attr 44
slapi_entry_replace_value 45
slapi_entry_replace_values 46
slapi_entry_schema_check 47
slapi_entry_set_dn 48
slapi_filter_get_attribute_type 49
slapi_filter_get_ava 50
slapi_filter_get_choice 51
slapi_filter_get_subfilt 52
slapi_filter_list_first 53
slapi_filter_list_next 54
slapi_isSDBM_authenticated 55
slapi_log_error 56
slapi_modify_internal 58
slapi_modrdn_internal 59
slapi_op_abandoned 60
slapi_pblock_destroy 61
slapi_pblock_get 62
slapi_pblock_set 69
slapi_search_internal 72
slapi_send_ldap_referral 74
slapi_send_ldap_result 76
slapi_send_ldap_search_entry 77
slapi_trace 78

Readers' Comments — We'd Like to Hear from You

z/OS
IBM Tivoli Directory Server Plug-in Reference for z/OS

Publication No. SA76-0148-00

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send your comments via e-mail to: mhvrcfs@us.ibm.com

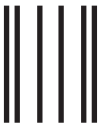
If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	E-mail address

Readers' Comments — We'd Like to Hear from You
SA76-0148-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY
12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

SA76-0148-00

Cut or Fold
Along Line

Program Number: 5694-A01

Printed in USA

SA76-0148-00