IBM Tivoli Directory Server

Server Plug-ins Reference

Version 6.1

GC32-1565-00

IBM Tivoli Directory Server

Server Plug-ins Reference

Version 6.1

GC32-1565-00

Note

Before using this information and the product it supports, read the general information under Appendix G, “Notices,” on page 79.

This edition applies to version 6, release 1, of IBM Tivoli Directory Server and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Appendix D. SLAPI API Categories. . . . 57

Appendix E. Plug-in examples. 67

Appendix F. Deprecated plug-in APIs 77

Appendix G. Notices 79

Index 83

About this book

IBM® Tivoli® Directory Server is the IBM implementation of Lightweight Directory Access Protocol for supported Windows®, AIX®, Linux® (xSeries®, zSeries®, pSeries®, and iSeries™), Solaris, and Hewlett-Packard UNIX® (HP-UX) operating systems.

IBM Tivoli Directory Server Version 6.1 Server Plug-ins Reference contains information about using and writing plug-ins that extend the capabilities of your IBM Tivoli Directory Server.

Intended audience for this book

This book is intended for directory server administrators who are responsible for maintaining and troubleshooting IBM Tivoli Directory Server and programmers.

Publications

This section lists publications in the IBM Tivoli Directory Server version 6.1 library and related documents. The section also describes how to access Tivoli publications online and how to order Tivoli publications.

IBM Tivoli Directory Server version 6.1 library

The following documents are available in the IBM Tivoli Directory Server version 6.1 library:

- *IBM Tivoli Directory Server Version 6.1 What's New for This Release*, SC23-6539-00

  Provides information about the new features in the IBM Tivoli Directory Server Version 6.1 release.
- *IBM Tivoli Directory Server Version 6.1 Quick Start Guide*, GI11-8172-00

  Provides help for getting started with IBM Tivoli Directory Server 6.1. Includes a short product description and architecture diagram, as well as a pointer to the product Information Center and installation instructions.
- *IBM Tivoli Directory Server Version 6.1 System Requirements*, SC23-7835-00

  Contains the minimum hardware and software requirements for installing and using IBM Tivoli Directory Server 6.1 and its related software. Also lists the supported versions of corequisite products such as DB2® and GSKit.
- *IBM Tivoli Directory Server Version 6.1 Installation and Configuration Guide*, GC32-1560-00

  Contains complete information for installing, configuring, and uninstalling IBM Tivoli Directory Server. Includes information about upgrading from a previous version of IBM Tivoli Directory Server.
- *IBM Tivoli Directory Server Version 6.1 Administration Guide*, GC32-1564-00

  Contains instructions for performing administrator tasks through the Web Administration Tool and the command line.
- *IBM Tivoli Directory Server Version 6.1 Command Reference*, SC23-7834-00

  Describes the syntax and usage of the command-line utilities included with IBM Tivoli Directory Server.
- *IBM Tivoli Directory Server Version 6.1 Server Plug-ins Reference*, GC32-1565-00

  Contains information about writing server plug-ins.

- *IBM Tivoli Directory Server Version 6.1 Programming Reference*, SC23-7836-00

  Contains information about writing Lightweight Directory Access Protocol (LDAP) client applications in C and Java™.
- *IBM Tivoli Directory Server Version 6.1 Performance Tuning and Capacity Planning Guide*, SC23-7836-00

  Contains information about tuning the directory server for better performance. Describes disk requirements and other hardware needs for directories of different sizes and with various read and write rates. Describes known working scenarios for each of these levels of directory and the disk and memory used; also suggests rough rules of thumb.
- *IBM Tivoli Directory Server Version 6.1 Problem Determination Guide*, GC32-1568-00

  Contains information about possible problems and corrective actions that can be tried before contacting IBM Software Support.
- *IBM Tivoli Directory Server Version 6.1 Messages Guide*, GC32-1567-00

  Contains a list of all informational, warning, and error messages associated with IBM Tivoli Directory Server 6.1.
- *IBM Tivoli Directory Server Version 6.1 White Pages*, SC23-7837-00

  Describes the Directory White Pages application, which is provided with IBM Tivoli Directory Server 6.1. Contains information about installing, configuring, and using the application for both administrators and users.

Related publications

The following documents also provide useful information:

- *Java Naming and Directory Interface™ 1.2.1 Specification* on the Sun Microsystems Web site at http://java.sun.com/products/jndi/1.2/javadoc/index.html.

  IBM Tivoli Directory Server Version 6.1 uses the Java Naming and Directory Interface (JNDI) client from Sun Microsystems. See this document for information about the JNDI client.

Accessing terminology online

The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available at the following Tivoli software library Web site:

http://publib.boulder.ibm.com/tividd/glossary/tivoliglossarymst.htm

The IBM Terminology Web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology Web site at the following Web address:

http://www.ibm.com/software/globalization/terminology

Accessing publications online

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Information Center Web site at http://publib.boulder.ibm.com/tividd/td/link/tdprodlist.html.

In the Tivoli Information Center window, click **Tivoli product manuals**. Click the letter that matches the first letter of your product name to access your product library. For example, click **M** to access the IBM Tivoli Monitoring library or click **O** to access the IBM Tivoli OMEGAMON® library.

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File → Print** window that allows Adobe® Reader to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi.

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:

1. Go to http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi.
2. Select your country from the list and click **Go**.
3. Click **About this site** in the main panel to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

For additional information, see the Accessibility Appendix in the *IBM Tivoli Directory Server Version 6.1 Installation and Configuration Guide*.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site at http://www.ibm.com/software/tivoli/education.

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

- IBM Support Assistant: You can search across a large collection of known problems and workarounds, Technotes, and other information at http://www.ibm.com/software/support/isa.
- Obtaining fixes: You can locate the latest fixes that are already available for your product.
- Contacting IBM Software Support: If you still cannot solve your problem, and you need to work with someone from IBM, you can use a variety of ways to contact IBM Software Support.

For more information about resolving problems, see the *IBM Tivoli Directory Server Version 6.1 Problem Determination Guide*.

Conventions used in this book

This book uses several conventions for special terms and actions, operating system-dependent commands and paths, and margin graphics.

Typeface conventions

This book uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations**:)
- Keywords and parameters in text

Italic

- Citations (examples: titles of books, diskettes, and CDs)
- Words defined in text (example: a nonswitched line is called a *point-to-point line*)
- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")
- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data.
- Variables and values you must provide: ... where *myname* represents....

`Monospace`

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This book uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows command line, replace **$***variable* with **%** *variable***%** for environment variables and replace each forward slash (**/**) with a backslash (****) in directory paths. The names of environment variables are not always the same in the Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX environments.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Chapter 1. Introduction to server plug-ins

Use the IBM Tivoli Directory Server Plug-ins Reference to help you create plug-ins that extend the capabilities of your IBM Tivoli Directory Server.

Server plug-ins extend the capabilities of your Directory Server. They are dynamically loaded into the LDAP server's address space when it is started. Once the plug-ins are loaded, the server calls the functions in a shared library by using function pointers.

A server front-end listens to the wire, receives and parses requests from clients, and then processes the requests by calling an appropriate database back-end function.

A server back-end reads and writes data to the database containing the directory entries. In addition to the default database operations, the LDAP server Database 2™ (DB2) back-end also provides functions for supporting replication and dynamic schema updates.

If the front-end fails to process a request it returns an error message to the client; otherwise, the back-end is called. After the back-end is called, it must return a message to the client. Either the front-end or the back-end, but not both can return a message to the client.

Note: This differs from the iPlanet server plug-in in that only the front-end of the iPlanet plug-in can send a message back to the client.

In this release of the IBM Tivoli Directory Server the following types of server plug-ins are supported:

Database plug-ins
Can be used to integrate your own database as a back-end to the server. A database plug-in can consist of all or only a portion of the functions discussed in this document. For example, the rdbm database back-end is a database plug-in. It provides functions that enable the server to interact with the DB2 database.

Pre-operation plug-ins
Functions that are executed before an LDAP operation is performed. For example, you can write a plug-in that checks new entries before they are added to the directory.

Post-operation plug-ins
Functions that are executed after an LDAP operation is performed.

Extended operation plug-ins
Are used to handle extended operations protocol that are defined in the LDAP V3 protocol.

Audit plug-ins
Are used to improve the security of the directory server. A default audit plug-in is provided with the server. Depending on the audit configuration parameters, this plug-in might log an audit entry in the default or specified audit log for each LDAP operation the server processed. The IBM Tivoli Directory Server administrator can use the activities stored in the audit log

to check for suspicious patterns of activity in an attempt to detect security violations. If security is violated, the audit log can be used to determine how and when the problem occurred and perhaps the amount of damage done. This information is very useful, both for recovery from the violation and, possibly, in the development of better security measures to prevent future problems. You can also write your own audit plug-ins to either replace, or add more processing to, the default audit plug-in.

DN partitioning plug-ins

Tivoli Directory Server provides an option to users to dynamically load customer written DN partitioning function during server runtime. The existing hash algorithm that is used to partition data is statically linked by the Tivoli Directory Server. However, with DN partitioning function implemented as a plug-in, the existing hash algorithm can be easily replaced with the customer written DN partitioning plug-in resulting in the Tivoli Directory Server being more flexible and adaptive. The existing hash algorithm however remains as the default DN partitioning plug-in, which is loaded during server startup if no customized code is available.

A server plug-in can return a message to the client as well. However, make sure that the server returns only one message.

Chapter 2. Writing a plug-in

A pblock is an opaque structure in which many parameters are stored. It is used to communicate between the server and your plug-ins. Application program interfaces (APIs) are provided for your plug-ins to get (or set) parameters in this structure.

Notes:

1. Plug-ins must be written using reentrant system calls.
2. There is no global mutex issue that the plug-in writer has to be concerned about in terms of interacting with the server. As long as the plug-ins call server-provided slapi APIs, a server's shared resource is protected by the APIs. However, because each request is serviced by a thread, and each thread might exercise the plug-in code, if there is any shared resource that the plug-in code creates, then mutex might be needed to protect the resources.

The following are examples of supported compilers:

- For Windows platforms—MS Visual C++ 6.0 and IBM VisualAge® C++ 3.5
- For AIX platforms—IBM VisualAge C++ 6.0
- For Linux/x86 platforms—GCC 3.2.3
- Linux/s390 platforms—GCC 3.2
- Linux/ppc platforms—GCC 3.2
- For Solaris platforms—Forte 6.1
- For HP-PARISC platforms—aCC A.03.30
- For HP IA64 platforms—aCC A.03.30

To write your own plug-in:

1. Start by writing your functions. Include slapi-plugin.h (where you can find all the parameters that can be defined in the pblock). You also can find a set of function prototypes for the available functions in the slapi-plugin.h file.
2. Decide the input parameters for your functions. Depending on the type of plug-in you are writing, you might need to work with a different set of parameters. See Appendix A, "Supported database functions," on page 17 for more information.
3. The following output is received from your functions:

   return code

   You can have the return code set to 0, which means that the server continues the operation. A return code of non-zero means that the server stops processing the operation. For example, if you have a pre-operation bind function that authenticates a user, it returns a non-zero after the authentication has been completed successfully. Otherwise, you can return a 0 and let the default bind operation continue the authentication process.

   return a message to the client

   You might want your plug-in (a pre-operation, a database operation, or a post-operation) to send an LDAP result to the client. For each operation, make sure there is only one LDAP result sent.

Note: The IBM Tivoli Directory Server default database plug-in always sends back a message. If you use the default database, do not have the post-operation return a message to the client.

output parameter
You might want to update parameters in the pblock that were passed to your function. For example, after your pre-operation bind function authenticates a user, you might want your plug-in to return the bound user's DN to the server. The server can then use it to continue with the processing of the operations requested by the user. See Appendix A, "Supported database functions," on page 17 for possible output parameters.

4. Call slapi APIs in the libslapi library file. See Appendix C, "Supported iPlanet APIs," on page 33 for information about the APIs supported in this release.
5. Write an initialization function for your plug-in to register your plug-in functions.
6. Export your initialization function from your plug-in shared library. Use an **.exp** file for AIX or a **.def** (or dllexport) file for Windows NT® to export your initialization. For other UNIX platforms, the exportation of the function is automatic when you create the shared library.
7. Compile and link your server plug-in object files with whatever libraries you need, and libslapi library file.
8. Add a plug-in directive in the server configuration file. The syntax of the plug-in directive is:

   ```
   attributeName: plugin-type  plugin-path init-func args ...
   ```
9. On a Windows NT operating system, in the ibmslapd.conf file, the plug-in directive is as follows:

   ```
   dn: cn=Directory, cn=RDBM Backends, cn=IBM Directory,
    cn=Schemas, cn=Configuration
   ibm-slapdPlugin: database    /lib/libback-rdbm.dll rdbm_backend_init
   ```

   Note: For the AIX, Linux, Solaris and HP operating platforms, the **.dll** extension is replaced with the appropriate extension:
   - For AIX and Linux operating systems - **.a**
   - For Solaris operating systems - **.so**
   - For HP-UX operating systems - **.sl**

The following rules apply when you place a plug-in directive in the configuration file:

- Multiple pre- or post-operations are called in the order they appear in the configuration file.
- The server can pass parameters to your plug-in initialization function by way of the argument list that is specified in the plug-in directive.

ibm-slapdPlugin is the attribute used to specify a plug-in which can be loaded by the server. This attribute is one of the attributes contained in objectclasses, such as ibm-slapdRdbmBackend and ibm-slapdLdcfBackend. For instance, in ibmslapd.conf, there is an entry which identifies the rdbm backend. In this entry, a database plug-in is specified by using the ibm-slapdPlugin attribute so that the server knows where and how to load this plug-in. If there is another plug-in to be loaded, such as a changelog plug-in, then specify it using another ibm-slapdPlugin attribute.

```
dn: cn=Directory, cn=RDBM Backends, cn=IBM Directory, cn=Schemas, cn=Configuration
...
objectclass: ibm-slapdRdbmBackend
ibm-slapdPlugin:  database  libback-rdbm.dll   rdbm_backend_init
ibm-slapdPlugin:  preoperation  libcl.dll CLInit "cn=changelog"
```

Chapter 3. Database plug-ins

Database plug-ins can be used to integrate your own database as a back-end to the server. A database plug-in can consist of all or a portion of the functions discussed in this section.

LDAP protocol-related functions
Are the default database functions. When you write a database plug-in you might not want to provide every function to handle the default database operations. You might need to provide some stub functions, however, which are used to send back an `unwilling to perform` message to the client when a particular function is not active.

Back-end-related functions
Are used to initialize or shut down the back-end and to handle back-end-specific configuration.

LDAP protocol-related functions

The following LDAP protocol-related functions are also the default database functions:

SLAPI_PLUGIN_DB_BIND_FN
Allows authentication information to be exchanged between the client and server.

SLAPI_PLUGIN_DB_UNBIND_FN
Terminates a protocol session.

SLAPI_PLUGIN_DB_ADD_FN
Adds an entry to the directory.

SLAPI_PLUGIN_DB_DELETE_FN
Deletes an entry.

SLAPI_PLUGIN_DB_SEARCH_FN
An LDAP back-end search routine.

SLAPI_PLUGIN_DB_COMPARE_FN
Gets the entry DN information and compares it with the attributes and values used in the compare function.

SLAPI_PLUGIN_DB_MODIFY_FN
Modifies an entry.

SLAPI_PLUGIN_DB_MODRDN_FN
Changes the last component of the name of an entry.

Back-end-related functions

These database back-end-related functions are used to initialize or shut down the back-end and to handle back-end-specific configuration:

SLAPI_PLUGIN_DB_INIT_FN
An LDAP back-end initialization routine.

SLAPI_PLUGIN_START_FN
An LDAP routine called at server startup.

SLAPI_PLUGIN_CLOSE_FN
An LDAP back-end close routine.

Note: Stand-alone, user-supplied server back-end plug-ins are not supported. However, they are supported when used in parallel with IBM-supplied server back-end plug-ins.

Chapter 4. Operation plug-ins

The following plug-in functions can be performed before or after an LDAP operation.

Pre-operation plug-ins

The following pre-operation functions can be executed before an LDAP operation is performed:

SLAPI_PLUGIN_PRE_BIND_FN
A function to call before the Directory Server executes an LDAP bind operation.

SLAPI_PLUGIN_PRE_UNBIND_FN
A function to call before the Directory Server executes an LDAP unbind operation.

SLAPI_PLUGIN_PRE_ADD_FN
A function to call before the Directory Server executes an LDAP add operation.

SLAPI_PLUGIN_PRE_DELETE_FN
A function to call before the Directory Server executes an LDAP delete operation.

SLAPI_PLUGIN_PRE_SEARCH_FN
A function to call before the Directory Server executes an LDAP search operation.

SLAPI_PLUGIN_PRE_COMPARE_FN
A function to call before the Directory Server executes an LDAP compare operation.

SLAPI_PLUGIN_PRE_MODIFY_FN
A function to call before the Directory Server executes an LDAP modify operation.

SLAPI_PLUGIN_PRE_MODRDN_FN
A function to call before the Directory Server executes a modify RDN database operation.

Post-operation plug-ins

The following post-operation plug-in functions can be executed after an LDAP operation is performed:

SLAPI_PLUGIN_POST_BIND_FN
A function to call after the Directory Server executes an LDAP bind operation.

SLAPI_PLUGIN_POST_UNBIND_FN
A function to call after the Directory Server executes an LDAP unbind operation.

SLAPI_PLUGIN_POST_ADD_FN
A function to call after the Directory Server executes an LDAP add operation.

SLAPI_PLUGIN_POST_DELETE_FN
A function to call after the Directory Server executes an LDAP delete operation.

SLAPI_PLUGIN_POST_SEARCH_FN
A function to call after the Directory Server executes an LDAP search operation.

SLAPI_PLUGIN_POST_COMPARE_FN
A function to call after the Directory Server executes an LDAP compare operation.

SLAPI_PLUGIN_POST_MODIFY_FN
A function to call after the Directory Server executes an LDAP modify operation.

SLAPI_PLUGIN_POST_MODRDN_FN
A function to call after the Directory Server executes an LDAP modify RDN database operation.

Extended operation plug-ins

LDAP operations can be extended with your own extended operation functions provided by a plug-in. An extended operation function might have an interface such as:

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

In this function, you can obtain the following two input parameters from the pblock passed in and communicate back to the server front-end with the following two output parameters:

Input parameters

These parameters can be obtained by calling the slapi_pblock_get API.

SLAPI_EXT_OP_RET_OID (char *)
The object identifier specified in a client's request.

SLAPI_EXT_OP_REQ_VALUE (struct berval *)
The information in a form defined by that request.

Output parameters

These parameters can be put to the parameter block passed in by the server by calling the slapi_pblock_set API.

SLAPI_EXT_OP_RET_OID (char *)
The object identifier that the plug-in function wants to send back to the client.

SLAPI_EXT_OP_RET_VALUE (struct berval *)
The value that the plug-in function wants to send back to the client.

After receiving and processing an extended operation request, an extended operation plug-in function might itself send an extended operation response back to a client or let the server send such a response. If the plug-in decides to send a response, it might call the slapi_send_ldap_result() function and return a result code SLAPI_PLUGIN_EXTENDED_SEND_RESULT to the server indicating that the plug-in has already sent an LDAP result message to the client. If the plug-in has not sent an LDAP result message to the client, the plug-in returns an LDAP result code and the server sends this result code back to the client.

To register an extended operation function, the initialization function of the extended operation plug-in might call slapi_pblock_set() to set the SLAPI_PLUGIN_EXT_OP_FN to the extended operation function and the SLAPI_PLUGIN_EXT_OP_OIDLIST parameter to the list of extended operation OIDs supported by the function. The list of OIDs which is listed in the ibm-slapdPlugin directive in ibmslapd.conf can be obtained by getting the SLAPI_PLUGIN_ARGV parameter from the pblock passed in.

The server keeps a list of all the OIDs that are set by plug-ins using the parameter SLAPI_PLUGIN_EXT_OP_OIDLIST. This list of extended operations can be queried by performing a search of the root DSE.

For example, in the Windows NT environment to specify an extended operation plug-in in the ibmslapd.conf file for the database rdbm add the following:

```
dn: cn=Directory, cn=RDBM Backends, cn=IBM SecureWay, cn=Schemas, cn=Configuration
   ibm-slapdPlugin database /bin/libback-rdbm.dll rdbm_backend_init
   ibm-slapdPlugin extendedop /tmp/myextop.dll myExtendedOpInit 123.456.789
```

File paths starting with a forward slash (/) are relative to the LDAP install directory; /tmp is changed to <ldap>\tmp, but C:\tmp is unchanged. This indicates that the function myExtendedOpInit that can be found in the /path/myextop.dll shared library is executed when the server starts. The myExtendedOp function that is registered in the initialization is used to handle the extended-operations. This function handles extended operations with the Object Identifier (OID) 123.456.789.

Note: For the AIX, Linux, Solaris and HP operating platforms, the **.dll** extension is replaced with the appropriate extension:

- For AIX and Linux operating systems - **.a**
- For Solaris operating systems - **.so**
- For HP-UX operating systems - **.sl**

Remember that plug-in directives are per-database.

Audit plug-ins

Administrators on some operating systems might want to use the system audit facilities to log the LDAP audit record with the system-defined record format. To allow flexibility in logging and record formats, a plug-in interface is provided. The server uses this interface to provide three types of auditing-related data to the external audit plug-ins if the auditing configuration is set to **on**. The data is passed to the external audit plug-ins through the standard plug-in's pblock interfaces, slapi_pblock_set() and slapi_pblock_get().

The three types of audit data available to the external audit plug-ins are:

Audit Configuration Information

This information is used to inform the external audit plug-in that at least one of the audit configuration options has been changed. The server expects the plug-in to determine whether to log the audit data associated with a particular LDAP operation, so it is important for the plug-in to have the current audit configuration information maintained by the server.

Audit Event Information

This information is used to inform the audit plug-in that certain events have happened. Event IDs, such as Auditing Started, Auditing Ended, or

Audit Configuration Options Changed, along with a message text describing the event, are sent by the server to the audit plug-in when such events occur.

Audit Record Information
This information is the audit data associated with each LDAP request received by the server. For each LDAP request, if the ibm-audit configuration option is set, the server provides the header data, control structure (if available), and operation-specific data to the audit plug-in. It is up to the audit plug-in to check its own copy of the LDAP audit configuration options or its platform-specific audit policy to determine whether to log and how to log the audit data.

The header file, audit-plugin.h, that defines the audit plug-in interface and data structures is shipped with the IBM Tivoli Directory Server C-Client SDK.

A default audit plug-in is provided and configured with the server. This plug-in performs the logging and formatting of the LDAP audit record. This default plug-in can be replaced with the platform-specific audit plug-in, if available, by changing the plug-in configuration lines in the ibmslapd.conf configuration file or through the IBM Tivoli Directory Server Web Administration Tool.

Note: There is no plug-in interface to the administration daemon audit.

Configuration options

The Audit Service has the following configuration options:

ibm-slapdLog
Specifies the path name of the audit log. The default is *directory_server_instance_name*/logs for AIX, Linux, Solaris, and HP-UX systems and *directory_server_instance_name*\logs for Windows systems.

ibm-audit: TRUE|FALSE
Enables or disables the audit service. Default is FALSE.

ibm-auditFailedOPonly: TRUE|FALSE
Indicates whether to log only failed operations. Default is TRUE.

ibm-auditBind: TRUE|FALSE
Indicates whether to log the Bind operation. Default is TRUE.

ibm-auditUnbind: TRUE|FALSE
Indicates whether to log the Unbind operation. Default is TRUE.

ibm-auditSearch: TRUE|FALSE
Indicates whether to log the Search operation. Default is FALSE.

ibm-auditAdd: TRUE|FALSE
Indicates whether to log the Add operation. Default is FALSE.

ibm-auditModify: TRUE|FALSE
Indicates whether to log the Modify operation. Default is FALSE.

ibm-auditDelete: TRUE|FALSE
Indicates whether to log the Delete operation. Default is FALSE.

ibm-auditModifyDN: TRUE|FALSE
Indicates whether to log the ModifyRDN operation. Default is FALSE.

ibm-auditExtOPEvent: TRUE|FALSE
Indicates whether to log LDAP V3 Event Notification extended operations. Default is FALSE.

ibm-auditExtOp: TRUE|FALSE
Indicates whether to log extended operations other than event notification extended operations. Default is FALSE.

ibm-auditCompare: TRUE|FALSE
Indicates whether to log compare operations. Default is FALSE.

ibm-auditVersion: 1|2|3
Indicates the auditing version. Default is 3. The audit versions are:

Audit Version 1
Basic Audit functionality.

Audit Version 2
Audit version 2 was introduced in IBM Tivoli Directory Server 5.2. Audit version 2 writes the audit version into the audit header, enables the auditing of Transport Layer Security (TLS) in the audit header, and enables auditing of additional information about controls.

Audit Version 3
Audit version 3 was introduced in IBM Tivoli Directory Server 6.0. Audit version 3 does everything that is done in audit versions 1 and 2 and also enables auditing of unique IDs.

ibm-auditAttributesOnGroupEvalOp: TRUE|FALSE
Indicates whether to log the attributes sent on a group evaluation extended operation. This setting is used only if ibm-auditExtOp is set to TRUE. Default is FALSE.

ibm-auditGroupsOnGroupControl: TRUE|FALSE
Indicates whether to log the groups sent on a group control. This setting is only used if ibm-auditVersion is set to 2 or greater. Default is FALSE.

These options are stored in the LDAP directory to allow dynamic configuration. They are contained in the **cn=Audit, cn=Log Management, cn=Configuration** entry. Only the directory administrator has access to modify this entry.

Note: For each modification of these option values, a message is logged in the slapd error log as well as the audit log to indicate the change.

The values of the audit configuration options are returned when a search of **cn=monitor** is requested by the LDAP administrator. These include:

- The value of the audit configuration options.
- The number of audit entries sent to the Audit plug-in for the current auditing session and for the current server session.

Examples

The following are examples of the various operations.

For auditing version 1:

```
2001-07-24-15:01:01.345-06:00--V3 Bind--
    bindDN:cn=test--client:9.1.2.3:12345--ConnectionID:12--
    received:2001-07-24-15:01:01.330-06:00--adminAuthority:Y--success
        name: cn=test
        authenticationChoice: simple


2001-07-24-15:01:02.367-06:00--V3 Search--
    bindDN:cn=test--client:9.1.2.3:12345--ConnectionID:12--
    received:2001-07-24-15:01:02.360-06:00--adminAuthority:Y--success
        base: o=sample
        scope: wholeSubtree
        derefAliases: neverDerefAliases
        typesOnly: false
        filter:  (&(cn=c*)(sn=a*))
```

Note: See the following examples for the format differences between authenticated and unauthenticated requests:

```
2001-07-24-15:22:33.541-06:00--V3 unauthenticated Search--
    bindDN: <*CN=NULLDN*>--client:9.1.2.2:32412--ConnectionID:18--
    received:2001-07-24-15:22:33.539-06:00--adminAuthority:Y--success


2001-07-24-15:22:34.555-06:00--V3 SSL unauthenticated Search--
    bindDN: <*CN=NULLDN*>--client:9.1.2.2:32412--ConnectionID:19--
    received:2001-07-24-15:22:34.550-06:00--adminAuthority:Y--success


2001-07-24-15:01:03.123-06:00--V3 Add--
    bindDN:cn=test--client:9.1.2.3:12345--ConnectionID:12--
    received:2001-07-24-15:01:03.100-06:00--adminAuthority:Y--entryAlreadyExists
       entry: cn=Jim Brown, ou=sales,o=sample
       attributes: objectclass, cn, sn, telphonenumber


2001-07-24-15:01:04.378-06:00--V3 Delete--
    bindDN:cn=test--client:9.1.2.3:12345--ConnectionID:12--
    received:2001-07-24-15:01:04.370-06:00--adminAuthority:Y--success
       entry:  cn=Jim Brown, ou=sales,o=sample


2001-07-24-15:01:05.712-06:00--V3 Modify--
    bindDN:cn=test--client:9.1.2.3:12345--ConnectionID:12--
    received:2001-07-24-15:01:05.708-06:00--adminAuthority:Y--noSuchObject
       object: cn=Jim Brown, ou=sales,o=sample
       add: mail
       delete: telephonenumber


2001-07-24-15:01:06.534-06:00--V3 ModifyDN--
    bindDN:cn=test--client:9.1.2.3:12345--ConnectionID:12--
    received:2001-07-24-15:01:06.530-06:00--adminAuthority:Y--noSuchObject
       entry:  cn=Jim Brown, ou=sales,o=sample
       newrdn: ou=r&d
       deleteoldrdn: true

""
2001-07-24-15:01:07.913-06:00--V3 Unbind--
    bindDN:cn=test--client:9.1.2.3:12345--ConnectionID:12--
    received:2001-07-24-15:01:07.910-06:00--adminAuthority:Y--success
```

For auditing versions 2 and 3:

Note: The format is explicitly stated in the *IBM Tivoli Directory Server Administration Guide* in the Appendix named "Audit format."

- **Bind**: (Administrator account status is displayed only if the bind is an administrator bind.)

```
AuditV3--2005-07-19-10:01:12.630-06:00DST--V3 Bind--bindDN: cn=root--client:
127.0.0.1:43021--connectionID: 1--received: 2005-07-19-10:01:12.389-06:00DST--Success
name: cn=root
authenticationChoice: simple
Admin Acct Status: Not Locked
```

- **Search**:

  ```
  AuditV3--2005-09-09-10:49:01.863-06:00DST--V3 Search--bindDN: cn=root--client:
  127.0.0.1:40722--connectionID: 2--received: 2005-09-09-10:49:01.803-06:00DST--Success
  controlType: 1.3.6.1.4.1.42.2.27.8.5.1
  criticality: false
  base: o=sample
  scope: wholeSubtree
  derefAliases: neverDerefAliases
  typesOnly: false
  filter: (&(cn=C*)(sn=A*))
  ```

- **Compare**:

  ```
  AuditV3--2005-09-09-10:51:45.959-06:00DST--V3 Compare--bindDN:
  cn=root--client:9.53.21.70:17037--connectionID: 5--received:
  2005-09-09-10:51:45.949-06:00DST--Success
  entry: cn=U1,ou=Austin,o=sample
  attribute: postalcode
  ```

- **Add**:

  ```
  AuditV3--2005-09-09-10:50:55.316-06:00DST--V3 Add--bindDN: cn=root--client:
  9.53.21.70:16525--connectionID: 3--received: 2005-09-09-10:50:52.652-06:00DST--Success
  entry: cn=U1,ou=Austin,o=sample
  attributes: objectclass, cn, sn, telephonenumber, internationaliSDNNumber,
  title, seealso, postalcode,facsimiletelephonenumber, ibm-entryuuid
  ```

- **Modify**:

  ```
  AuditV3--2005-09-09-10:51:07.103-06:00DST--V3 Modify--bindDN: cn=root--client:
  9.53.21.70:16781--connectionID: 4--received: 2005-09-09-10:51:06.923-06:00DST--Success
  object: cn=U1,ou=Austin,o=sample
  replace: postalcode
  ```

- **Modify DN**:

  ```
  AuditV3--2005-09-09-10:52:14.590-06:00DST--V3 ModifyDN--bindDN: cn=root--client:
  9.53.21.70:17293--connectionID: 6--received: 2005-09-09-10:52:14.230-06:00DST--Success
  entry: cn=U1,ou=Austin,o=sample
  newrdn: cn=U1A
  deleteoldrdn: true
  ```

- **Delete**:

  ```
  AuditV3--2005-09-09-10:52:36.381-06:00DST--V3 Delete--bindDN: cn=root--client:
  9.53.21.70:17549--connectionID: 7--received: 2005-09-09-10:52:35.971-06:00DST--Success
  controlType: 1.3.6.1.4.1.42.2.27.8.5.1
  criticality: false
  entry: cn=U1A,ou=Austin,o=sample
  ```

- **Unbind**:

  ```
  AuditV3--2005-09-09-10:51:07.143-06:00DST--V3 Unbind--bindDN: cn=root--client:
  9.53.21.70:16781--connectionID: 4--received: 2005-09-09-10:51:07.143-06:00DST--Success
  ```

- **Extended Operation**:

  ```
  AuditV3--2005-09-09-10:57:11.647-06:00DST--V3 extended operation--bindDN: cn=root--client:
  9.53.21.70:17805--connectionID: 8--received: 2005-09-09-10:57:11.557-06:00DST--Success
  OID: 1.3.18.0.2.12.6
  ```

  Each extended operation can have its own specific data. See the description of each extended operation in the *IBM Tivoli Directory Server Programming Reference* for specific details.

- **Auditing of Controls**: Each control audited contains the controlType and the criticality. If the audit version is set to version 2 or higher, the server audits additional information about the controls sent on an operation. This information is placed just after the header and before the operation specific data. The following example is an add operation with the password policy control.

```
AuditV3--2005-09-09-10:50:55.316-06:00DST--V3 Add--bindDN: cn=root--client:
9.53.21.70:16525--connectionID: 3--received: 2005-09-09-10:50:52.652-06:00DST--Success
controlType: 1.3.6.1.4.1.42.2.27.8.5.1
criticality: false
entry: cn=U1,ou=Austin,o=sample
attributes: objectclass, cn, sn, telephonenumber, internationaliSDNNumber, title,
seealso, postalcode, facsimiletelephonenumber, ibm-entryuuid
```

- **Auditing of a transaction**: When the server receives an operation within a transaction, the transaction ID is audited in both the audit header and in the list of controls. Note that the transaction ID is placed just before the results of the operation in the header. The following is an example of an add operation within a transaction.

```
AuditV3--2005-09-09-10:57:11.607-06:00DST--V3 Add--bindDN: cn=root--client:
9.53.21.70:17805--connectionID: 8--received: 2005-09-09-10:57:11.447-06:00DST--transactionID:
11262814319.53.21.7017805--Success
controlType: 1.3.18.0.2.10.5
criticality: true
entry: cn=U1,ou=Austin,o=sample
attributes: objectclass, cn, sn, telephonenumber, internationaliSDNNumber, title,
seealso, postalcode, facsimiletelephonenumber, ibm-entryuuid
```

- **Auditing of operation with the Proxy Authorization Control**: The following is an example of a control with additional information that is audited only if the version is set to 2 or higher:

```
AuditV3--2005-09-09-14:45:08.844-06:00DST--V3 Search--bindDN: cn=root--client: 1
27.0.0.1:4371--connectionID: 10--received: 2005-09-09-14:45:04.858-06:00DST--Suc
cess
controlType: 2.16.840.1.113730.3.4.18
criticality: true
ProxyDN: dn:cn=user1,o=sample
base: o=sample
scope: wholeSubtree
derefAliases: neverDerefAliases
typesOnly: false
filter: (cn=A*)
```

Appendix A. Supported database functions

The three parameters in the first stanza are passed to the nine default database functions as input:

```
/* backend, connection, operation */
SLAPI_BACKEND
SLAPI_CONNECTION
SLAPI_OPERATION

/* arguments that are common to all operations */
SLAPI_CONN_DN
SLAPI_CONN_AUTHTYPE
SLAPI_REQCONTROLS

/* add arguments */
SLAPI_ADD_TARGET
SLAPI_ADD_ENTRY

/* bind arguments */
SLAPI_BIND_TARGET
SLAPI_BIND_METHOD
SLAPI_BIND_CREDENTIALS
SLAPI_BIND_SASLMECHANISM
/* bind return values */
SLAPI_BIND_RET_SASLCREDS

/* compare arguments */
SLAPI_COMPARE_TARGET
SLAPI_COMPARE_TYPE
SLAPI_COMPARE_VALUE

/* delete arguments */
 SLAPI_DELETE_TARGET

/* modify arguments
```

Note: The input and output value for setting and getting SLAPI_MODIFY_MODS in the slapi_pblock_set() and slapi_pblock_get() functions is a pointer to a list of LDAPMod structures. This differs from the iPlanet implementation which is a pointer to an array of LDAPMod pointers. Go to the LDAPMod structure in the ldap.h file to see how to traverse the linked list using the pointer to the next LDAPMod structure.

```
                             */
SLAPI_MODIFY_TARGET
SLAPI_MODIFY_MODS

/* modrdn arguments */
SLAPI_MODRDN_TARGET
SLAPI_MODRDN_NEWRDN
SLAPI_MODRDN_DELOLDRDN
SLAPI_MODRDN_NEWSUPERIOR

/* search arguments */
SLAPI_SEARCH_TARGET
SLAPI_SEARCH_SCOPE
SLAPI_SEARCH_DEREF
SLAPI_SEARCH_SIZELIMIT
SLAPI_SEARCH_TIMELIMIT
SLAPI_SEARCH_FILTER
SLAPI_SEARCH_STRFILTER
```

```
SLAPI_SEARCH_ATTRS
SLAPI_SEARCH_ATTRSONLY

/* abandon arguments */
SLAPI_ABANDON_MSGID

/* plugin types supported */
#define SLAPI_PLUGIN_DATABASE
#define SLAPI_PLUGIN_EXTENDEDOP
#define SLAPI_PLUGIN_PREOPERATION
#define SLAPI_PLUGIN_POSTOPERATION
#define SLAPI_PLUGIN_AUDIT

/* plugin configuration params */
#define SLAPI_PLUGIN
#define SLAPI_PLUGIN_PRIVATE
#define SLAPI_PLUGIN_TYPE
#define SLAPI_PLUGIN_ARGV
#define SLAPI_PLUGIN_ARGC

/* audit plugin defines */
#define SLAPI_PLUGIN_AUDIT_DATA
#define SLAPI_PLUGIN_AUDIT_FN

/* managedsait control */
#define SLAPI_MANAGEDSAIT

/* config stuff */
#define SLAPI_CONFIG_FILENAME
#define SLAPI_CONFIG_LINENO
#define SLAPI_CONFIG_ARGC
#define SLAPI_CONFIG_ARGV

/*  operational params */
#define SLAPI_TARGET_DN
#define SLAPI_REQCONTROLS

/* modrdn params */
#define SLAPI_MODRDN_TARGET_UP
#define SLAPI_MODRDN_TARGET
#define SLAPI_MODRDN_NEWRDN
#define SLAPI_MODRDN_DELOLDRDN
#define SLAPI_MODRDN_NEWSUPERIOR

/* extended operation params */
#define SLAPI_EXT_OP_REQ_OID
#define SLAPI_EXT_OP_REQ_VALUE

/* Search result params */
#define SLAPI_NENTRIES
```

Output parameters

The following are the output parameters of the default database functions:

```
/* common for internal plugin_ops */
SLAPI_PLUGIN_INTOP_RESULT
SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES

SLAPI_CONN_DN
SLAPI_CONN_AUTHTYPE

/# Types of authentication (for SLAPI_CONN_AUTHTYPE) */
#define SLAPD_AUTH_NONE   "none"
#define SLAPD_AUTH_SIMPLE "simple"
#define SLAPD_AUTH_SSL    "SSL"
#define SLAPD_AUTH_SASL   "SASL " /* followed by the mechanism name */
```

Appendix B. Parameter Reference

This chapter describes the parameters available in the Slapi_PBlock parameter block, the type of data associated with each parameter, and the plug-in functions in which these parameters are accessible.

To get the values of these parameters, call the slapi_pblock_get() function. To set the values of these parameters, call the slapi_pblock_set() function. Using these parameters, you can get and set the following information:

- "Parameters for Registering Plug-in Functions"
- "Parameters Accessible to All Plug-ins" on page 22
- "Parameters for the Configuration Function" on page 26
- "Parameters for the Bind Function" on page 26
- "Parameters for the Search Function" on page 26
- "Parameters for the Add Function" on page 28
- "Parameters for the Compare Function" on page 28
- "Parameters for the Delete Function" on page 28
- "Parameters for the Modify Function" on page 28
- "Parameters for the Modify RDN Function" on page 29
- "Parameters for the Abandon Function" on page 29
- "Parameters for Database Import" on page 29
- "Parameters for Database Export" on page 30
- "Parameters for Database Archive" on page 30
- "Parameters for Database Restore" on page 30
- "Parameters for Extended Operations" on page 31
- "Parameters for Internal LDAP Operations" on page 31
- "Parameters for the DN Partitioning Function" on page 32

Parameters for Registering Plug-in Functions

The parameters listed in this section identify plug-in functions recognized by the server. To register your plug-in function, set the value of the appropriate parameter to the name of your function.

Note: You do not need to get the value of any of the plug-in function parameters. The parameters for registering plug-in functions are organized in the following sections:

- "Database Plug-ins"
- "Pre-Operation/Data Validation Plug-ins" on page 20
- "Post-Operation/Data Notification Plug-ins" on page 21
- "Extended Operation Plug-ins" on page 21
- "DN Partitioning Plug-ins" on page 22

Database Plug-ins

The following parameters are used to register database plug-in functions.

Each parameter corresponds to an operation performed by the back-end database. When integrating your own database with the IBM Tivoli Directory Server, you need to write and register your own plug-in functions that handle these operations.

To register your plug-in function, write an initialization function that sets the values of the following parameters to your functions:

Table 1. Parameters and their descriptions of database plug-ins

Parameter ID	**Description**
SLAPI_PLUGIN_DB_BIND_FN	Called in response to an LDAP bind request.
SLAPI_PLUGIN_DB_UNBIND_FN	Called in response to an LDAP unbind request.
SLAPI_PLUGIN_DB_SEARCH_FN	Called in response to an LDAP search request. The function collects a set of candidates for the search results.
SLAPI_PLUGIN_DB_COMPARE_FN	Called in response to an LDAP compare request.
SLAPI_PLUGIN_DB_MODIFY_FN	Called in response to an LDAP modify request.
SLAPI_PLUGIN_DB_MODRDN_FN	Called in response to an LDAP modify RDN request.
SLAPI_PLUGIN_DB_ADD_FN	Called in response to an LDAP add request.
SLAPI_PLUGIN_DB_DELETE_FN	Called in response to an LDAP delete request.
SLAPI_PLUGIN_DB_ABANDON_FN	Called in response to an LDAP abandon operation request.
SLAPI_PLUGIN_DB_CONFIG_FN	Called when the server is parsing the slapd.conf configuration file. If the IBM Tivoli Directory Server encounters a directive that it does not recognize, then IBM Tivoli Directory Server passes the directive to the back-end using this routine.
SLAPI_PLUGIN_START_FN	Called during the server startup.
SLAPI_PLUGIN_CLOSE_FN	Called when the server is shutting down. You can use this function to prepare your back-end database for shutdown.

Pre-Operation/Data Validation Plug-ins

The following parameters are used to register pre-operation/data validation plug-in functions.

To register your plug-in function, write an initialization function that sets the values of the following parameters to your functions:

Table 2. Parameters and their descriptions of pre-operation/data validation plug-ins

Parameter ID	**Description**
SLAPI_PLUGIN_PRE_BIND_FN	Called before an LDAP bind operation is completed.
SLAPI_PLUGIN_PRE_UNBIND_FN	Called before an LDAP unbind operation is completed.

Table 2. Parameters and their descriptions of pre-operation/data validation plug-ins (continued)

SLAPI_PLUGIN_PRE_SEARCH_FN	Called before an LDAP search operation is completed.
SLAPI_PLUGIN_PRE_COMPARE_FN	Called before an LDAP compare operation is completed.
SLAPI_PLUGIN_PRE_MODIFY_FN	Called before an LDAP modify operation is completed.
SLAPI_PLUGIN_PRE_MODRDN_FN	Called before an LDAP modify RDN operation is completed.
SLAPI_PLUGIN_PRE_ADD_FN	Called before an LDAP add operation is completed.
SLAPI_PLUGIN_PRE_DELETE_FN	Called before an LDAP delete operation is completed.
SLAPI_PLUGIN_START_FN	Called at server startup.
SLAPI_PLUGIN_CLOSE_FN	Called before the server shuts down. You can specify a close function for each pre-operation plug-in.

Post-Operation/Data Notification Plug-ins

The following parameters are used to register post-operation/data notification plug-in functions:

Table 3. Parameters and their descriptions of post-operation/data notification plug-ins

Parameter ID	Description
SLAPI_PLUGIN_POST_BIND_FN	Called after an LDAP bind operation is completed.
SLAPI_PLUGIN_POST_UNBIND_FN	Called after an LDAP unbind operation is completed.
SLAPI_PLUGIN_POST_SEARCH_FN	Called after an LDAP search operation is completed.
SLAPI_PLUGIN_POST_COMPARE_FN	Called after an LDAP compare operation is completed.
SLAPI_PLUGIN_POST_MODIFY_FN	Called after an LDAP modify operation is completed.
SLAPI_PLUGIN_POST_MODRDN_FN	Called after an LDAP modify RDN operation is completed.
SLAPI_PLUGIN_POST_ADD_FN	Called after an LDAP add operation is completed.
SLAPI_PLUGIN_POST_DELETE_FN	Called after an LDAP delete operation is completed.
SLAPI_PLUGIN_START_FN	Called at server startup.
SLAPI_PLUGIN_CLOSE_FN	Called before the server shuts down. You can specify a close function for each post-operation plug-in.

Extended Operation Plug-ins

The following parameters are used to register extended operation plug-in functions:

Table 4. Parameters and their descriptions of extended operation plug-ins

Parameter ID	Data Type	Description
SLAPI_PLUGIN_EXT_OP_FN	void *	Your plug-in function for handling an extended operation.
SLAPI_PLUGIN_EXT_OP_OIDLIST	char **	NULL-terminated array of OIDs identifying the extended operations handled by the plug-in function.
SLAPI_PLUGIN_START_FN	void *	Called at server startup.
SLAPI_PLUGIN_CLOSE_FN	void *	Called before the server shuts down. You can specify a close function for each extended operation plug-in.

DN Partitioning Plug-ins

The main purpose of the initialization function is to call slapi_pblock_set API to register the user provided DN partitioning function. The parameter SLAPI_PLUGIN_PROXY_DN_PARTITION_FN should be used to set the function address.

The following parameters are used to register DN partitioning plug-in functions:

Table 5. Parameters and their descriptions of DN partitioning plug-ins

Parameter ID	Description
SLAPI_PLUGIN_PROXY_DN_PARTITION_FN	Address of a customized DN partitioning function.

Parameters Accessible to All Plug-ins

The parameters listed in this section are accessible to all types of plug-ins. The parameters in this section are organized in the following sections:

- “Information About the Database”
- “Information About the Connection” on page 23
- “Information About the Operation” on page 24
- “Information About the Plug-ins” on page 25

Information About the Database

The following parameters specify information about the back-end database. These parameters are available for all types of plug-ins.

Note: These specific parameters cannot be set by calling slapi_pblock_set(). You can get these parameters by calling slapi_pblock_get().

Table 6. Parameters specifying information about the back-end database

Parameter ID	Data Type	Description

Table 6. Parameters specifying information about the back-end database (continued)

SLAPI_BE_MONITORDN	char *	**Note:** Netscape Directory Server 3.x releases only.DN used to monitor the back-end database. **Note:** This is no longer supported in the Netscape Directory Server 4.0 release.
SLAPI_BE_TYPE	char *	Type of back-end database. This is the type of back-end database specified by the database directive in the slapd.conf file.
SLAPI_BE_READONLY	int	Specifies whether or not the back-end database is read-only. This is determined by the read-only directive in the slapd.conf file: • **1** means that the database back-end is read-only. • **0** means that the database back-end is writable.
SLAPI_DBSIZE	int	Specifies the size of the back-end database. If you are using your own database instead of the default database, your SLAPI_DB_SIZE_FN function must set the value of this parameter.

Information About the Connection

The following parameters specify information about the connection. These parameters are available for all types of plug-ins:

Table 7. Parameters specifying information about the connection

Parameter ID	Data Type	Description
SLAPI_CONN_ID	int	ID identifying the current connection.
SLAPI_CONN_DN	char *	DN of the user authenticated on the current connection. The caller should call slapi_ch_free() on this value only if slapi_pblock_set() is called to set SLAPI_CONN_DN to a new value.

Table 7. Parameters specifying information about the connection (continued)

SLAPI_CONN_AUTHTYPE	char *	Method used to authenticate the current user. This parameter can have one of the following values: **SLAPD_AUTH_NONE** Specifies that no authentication mechanism was used (for example, in cases of anonymous authentication). **SLAPD_AUTH_SIMPLE** Specifies that simple authentication (user name and password) was used to authenticate the current user. **SLAPD_AUTH_SSL** Specifies that SSL (certificate-based authentication) was used to authenticate the current user. **SLAPD_AUTH_SASL** Specifies that a SASL (simple authentication and security layer) mechanism was used to authenticate the current user.
SLAPI_CONN_CLIENTNETADDR_STR	char *	IP address of the client requesting the operation.
SLAPI_CONN_SERVERNETADDR_STR	char *	IP address of the server to which the client is connecting. You can use this parameter if, for example, your server accepts connections on multiple IP addresses.

Information About the Operation

The following parameters specify information about the current operation. These parameters are available for all types of plug-ins:

Table 8. Parameters specifying information about the current operation

Parameter ID	Data Type	Description
SLAPI_OPINITIATED_TIME	time_t	Time when the server began processing the operation.
SLAPI_TARGET_DN	char *	Specifies the DN to which the operation applies, for example, the DN of the entry being added or removed.
SLAPI_REQCONTROLS	LDAPControl **	Array of the controls specified in the request.

Information About the Plug-ins

The following parameters specify information about the plug-in that is available to all plug-in functions defined in the current library. These parameters are available for all types of plug-ins.

Table 9. Parameters specifying information about the plug-in that is available to all plug-in functions

Parameter ID	Data Type	Description
SLAPI_PLUGIN_PRIVATE	void *	Private data that you want passed to your plug-in functions.
SLAPI_PLUGIN_TYPE	int	Specifies the type of plug-in function.
SLAPI_PLUGIN_ARGV	char **	NULL-terminated array of command-line arguments specified for the plug-in directive in the slapd.conf file.
SLAPI_PLUGIN_ARGC	int	Number of command-line arguments specified for the plug-in directive in the slapd.conf file.

Types of Plug-ins

The SLAPI_PLUGIN_TYPE parameter can have one of the following values, which identifies the type of the current plug-in:

Table 10. Defined constants and their description of SLAPI_PLUGIN_TYPE parameter value

Defined Constant	Description
SLAPI_PLUGIN_DATABASE	Database plug-in
SLAPI_PLUGIN_EXTENDEDOP	Extended operation plug-in
SLAPI_PLUGIN_PREOPERATION	Pre-operation/data validation plug-in
SLAPI_PLUGIN_POSTOPERATION	Post-operation/data notification plug-in
SLAPI_PLUGIN_SYNTAX	Syntax plug-in

Parameters for the Configuration Function

The following table lists the parameters in the parameter block passed to the database configuration function. If you are writing a pre-operation, database, or post-operation configuration function, you can get these values by calling the slapi_pblock_get() function.

Table 11. Parameters for the database configuration function

Parameter ID	Data Type	Description
SLAPI_CONFIG_FILENAME	char *	Name of the configuration file that is being read, for example, slapd.conf.
SLAPI_CONFIG_LINENO	int	Line number of the current directive in the configuration file.
SLAPI_CONFIG_ARGC	int	Number of arguments in the current directive.
SLAPI_CONFIG_ARGV	char **	Array of the arguments from the current directive.

Parameters for the Bind Function

The following table lists the parameters in the parameter block passed to the database bind function. If you are writing a pre-operation, database, or post-operation bind function, you can get these values by calling the slapi_pblock_get() function.

Table 12. Parameters for the database bind function

Parameter ID	Data Type	Description
SLAPI_BIND_TARGET	char *	DN of the entry to bind as.
SLAPI_BIND_METHOD	int	Authentication method used, for example, LDAP_AUTH_SIMPLE or LDAP_AUTH_SASL.
SLAPI_BIND_ CREDENTIALS	struct berval *	Credentials from the bind request.
SLAPI_BIND_RET_ SASLCREDS	struct berval *	Credentials that you want sent back to the client. **Note:** Set this before calling slapi_send_ldap_result().
SLAPI_BIND_ SASLMECHANISM	char *	SASL mechanism used, for example, LDAP_SASL_EXTERNAL.

Parameters for the Search Function

The following table lists the parameters in the parameter block passed to the database search function. If you are writing a pre-operation, database, or post-operation search function, you can get these values by calling the slapi_pblock_get() function.

Table 13. Parameters for the database search function

Parameter ID	Data Type	Description
SLAPI_SEARCH_TARGET	char *	DN of the base entry in the search operation (the starting point of the search).
SLAPI_SEARCH_SCOPE	int	The scope of the search. The scope can be one of the following values: • LDAP_SCOPE_BASE • LDAP_SCOPE_ ONELEVEL • LDAP_SCOPE_SUBTREE
SLAPI_SEARCH_DEREF	int	Method for handling aliases in a search. This method can be one of the following values: • LDAP_DEREF_NEVER • LDAP_DEREF_ SEARCHING • LDAP_DEREF_FINDING • LDAP_DEREF_ALWAYS
SLAPI_SEARCH_SIZELIMIT	int	Maximum number of entries to return in the search results.
SLAPI_SEARCH_ TIMELIMIT	int	Maximum amount of time (in seconds) allowed for the search operation.
SLAPI_SEARCH_FILTER	Slapi_Filter *	Slapi_Filter struct (an opaque data structure) representing the filter to be used in the search.
SLAPI_SEARCH_STRFILTER	char *	String representation of the filter to be used in the search.
SLAPI_SEARCH_ATTRS	char **	Array of attribute types to be returned in the search results.
SLAPI_SEARCH_ ATTRSONLY	int	Specifies whether the search results return attribute types only or attribute types and values: • **0** means return both attributes and values. • **1** means return attribute types only.

The following parameters are set by the front-end and back-end as part of the process of executing the search:

Parameter ID	Data Type	Description
SLAPI_NENTRIES	int	Number of search results found.

Parameters for the Add Function

The following table lists the parameters in the parameter block passed to the database add function. If you are writing a pre-operation, database, or post-operation add function, you can get these values by calling the slapi_pblock_get() function.

Table 14. Parameters for the database add function

Parameter ID	Data Type	Description
SLAPI_ADD_TARGET	char *	DN of the entry to be added.
SLAPI_ADD_ENTRY	Slapi_Entry *	The entry to be added (specified as the opaque Slapi_Entry datatype).

Parameters for the Compare Function

The following table lists the parameters in the parameter block passed to the database compare function. If you are writing a pre-operation, database, or post-operation compare function, you can get these values by calling the slapi_pblock_get() function.

Table 15. Parameters for the database compare function

Parameter ID	Data Type	Description
SLAPI_COMPARE_TARGET	char *	DN of the entry to be compared.
SLAPI_COMPARE_TYPE	char *	Attribute type to use in the comparison.
SLAPI_COMPARE_VALUE	struct berval *	Attribute value to use in the comparison.

Parameters for the Delete Function

The following table lists the parameters in the parameter block passed to the database delete function. If you are writing a pre-operation, database, or post-operation delete function, you can get these values by calling the slapi_pblock_get() function.

Table 16. Parameters for the database delete function

Parameter ID	Data Type	Description
SLAPI_DELETE_TARGET	char *	DN of the entry to delete.

Parameters for the Modify Function

The following table lists the parameters in the parameter block passed to the database modify function. If you are writing a pre-operation, database, or post-operation modify function, you can get these values by calling the slapi_pblock_get() function.

Table 17. Parameters for the database modify function

Parameter ID	Data Type	Description
SLAPI_MODIFY_TARGET	char *	DN of the entry to be modified.
SLAPI_MODIFY_MODS	LDAPMod **	A NULL-terminated array of LDAPMod structures, which represent the modifications to be performed on the entry.

Parameters for the Modify RDN Function

The following table lists the parameters in the parameter block passed to the database modify RDN function. If you are writing a pre-operation, database, or post-operation modify RDN function, you can get these values by calling the slapi_pblock_get() function.

Table 18. Parameters for the database modify RDN function

Parameter ID	Data Type	Description
SLAPI_MODRDN_TARGET	char *	DN of the entry that you want to rename.
SLAPI_MODRDN_NEWRDN	char *	New RDN to assign to the entry.
SLAPI_MODRDN_ DELOLDRDN	int	Specifies whether you want to delete the old RDN: • **0** means don't delete the old RDN. • **1** means delete the old RDN.
SLAPI_MODRDN_ NEWSUPERIOR	char *	DN of the new parent of the entry, if the entry is being moved to a new location in the directory tree.

Parameters for the Abandon Function

The following table lists the parameters in the parameter block passed to the database abandon function. If you are writing a pre-operation, database, or post-operation abandon function, you can get these values by calling the slapi_pblock_get() function.

Table 19. Parameters for the database abandon function

Parameter ID	Data Type	Description
SLAPI_ABANDON_MSGID	unsigned long	Message ID of the operation to abandon.

Parameters for Database Import

The following table lists the parameters in the parameter block passed to the database import function, which is responsible for importing LDIF files into the database. If you are writing your own plug-in function for performing this work, you can get these values by calling the slapi_pblock_get() function.

Table 20. Parameters to the database import function

Parameter ID	Data Type	Description
SLAPI_LDIF2DB_FILE	char *	LDIF file that needs to be imported into the database.
SLAPI_LDIF2DB_ REMOVEDUPVALS	int	Specifies whether or not the duplicate values of attributes must be removed: • If **1**, remove any duplicate attribute values when creating an entry. • If **0**, do not remove any duplicate attribute values when creating an entry.

Parameters for Database Export

The following table lists the parameters in the parameter block passed to the database export function, which is responsible for exporting LDIF files into the database. If you are writing your own plug-in function for performing this work, you can get these values by calling the slapi_pblock_get() function.

Table 21. Parameters to the database export function

Parameter ID	Data Type	Description
SLAPI_DB2LDIF_PRINTKEY	int	Specifies whether or not the database keys must be printed out as well: • If **1**, include the database key for each entry. • If **0**, do not include the database key for each entry.

Parameters for Database Archive

The following table lists the parameters in the parameter block passed to the database archive function, which is responsible for archiving the database. If you are writing your own plug-in function for performing this work, you can get these values by calling the slapi_pblock_get() function.

Table 22. Parameters to the database archive function

Parameter ID	Data Type	Description
SLAPI_SEQ_VAL	char *	Specifies the directory in which you want to store the archive.

Parameters for Database Restore

The following table lists the parameters in the parameter block passed to the database restore function, which is responsible for restoring the database from an archive. If you are writing your own plug-in function for performing this work, you can get these values by calling the slapi_pblock_get() function.

Table 23. Parameters to the database restore function

Parameter ID	Data Type	Description
SLAPI_SEQ_VAL	char *	Specifies the directory containing the archive.

Parameters for Extended Operations

The following table lists the parameters in the parameter block passed to extended operation functions. If you are writing your own plug-in function for performing this work, you can get these values by calling the slapi_pblock_get() function.

Table 24. Parameters to extended operation functions

Parameter ID	Data Type	Description
SLAPI_EXT_OP_REQ_OID	char *	Object ID (OID) of the extended operation specified in the request.
SLAPI_EXT_OP_REQ_ VALUE	struct berval*	Value specified in the request.
SLAPI_EXT_OP_RET_OID	char *	OID that you want sent back to the client.
SLAPI_EXT_OP_RET_ VALUE	struct berval*	Value that you want sent back to the client.

Parameters for Internal LDAP Operations

The following parameters are used in conjunction with functions that you can call to perform LDAP operations from a plug-in (these internal operations do not return any data to a client).

Table 25. Parameters used in conjunction with functions that performs LDAP operations from a plug-in

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTOP_ RESULT	int	Result code of the internal LDAP operation.
SLAPI_PLUGIN_INTOP_ SEARCH_ENTRIES	Slapi_Entry **	Array of entries found by an internal LDAP search operation.

The following functions set both parameters:

- slapi_search_internal()
- slapi_search_internal_callback()

The following functions set only the SLAPI_PLUGIN_INTOP_RESULT parameter:

- slapi_add_internal()
- slapi_add_entry_internal()
- slapi_delete_internal()
- slapi_modify_internal()
- slapi_modrdn_internal()

Parameters for the DN Partitioning Function

The following table lists the parameters in the parameter block that are passed between the Proxy Server backend and the plug-in using the slapi_pblock_set() and slapi_pblock_get() functions. If you are writing your own DN partitioning plug-in, you can get value of these parameters by calling slapi_pblock_get().

Table 26. Parameters for the DN partitioning function

Parameter ID	**Description**
SLAPI_TARGET_DN	Address of a DN for which the partition value needs to be calculated. This DN is normalized and is in the UTF-8 format.
SLAPI_PARTITION_BASE	Address of a base DN that is the base or suffix of the target DN. This base DN is normalized and is in the UTF-8 format.
SLAPI_NUMBER_OF_PARTITIONS	The number of partitions used for the calculation of DN partition value.
SLAPI_PARTITION_NUMBER	A plug-in calculated partition value.

Appendix C. Supported iPlanet APIs

The following iPlanet APIs are supported in this release.

For pblock:

```
int slapi_pblock_get( Slapi_PBlock *pb, int arg, void *value );
int slapi_pblock_set( Slapi_PBlock *pb, int arg, void *value );
Slapi_PBlock *slapi_pblock_new();
void slapi_pblock_destroy( Slapi_PBlock *pb);
```

For memory management:

```
char *slapi_ch_malloc( unsigned long size );
void slapi_ch_free( void *ptr );
char *slapi_ch_calloc( unsigned long nelem, unsigned long size );
char *slapi_ch_realloc(char *block, unsigned long size );
char *slapi_ch_strdup(char *s );
```

For sending results:

```
void slapi_send_ldap_result( Slapi_PBlock *pb, int err, char
*matched, char *text,
      int nentries, struct berval **urls);
```

For LDAP specific objects:

```
char *slapi_dn_normalize( char *dn );
char *slapi_dn_normalize_case( char *dn );
char *slapi_dn_ignore_case( char *dn );
char *slapi_dn_normalize_v3( char *dn );
char *slapi_dn_normalize_case_v3( char *dn );
char *slapi_dn_ignore_case_v3( char *dn );
char *slapi_dn_compare_v3(char *dn1,
        char* dn2);
int slapi_dn_issuffix(char *dn, char *suffix);
char *slapi_entry2str( Slapi_Entry *e, int
        *len );
Slapi_Entry *slapi_str2entry( char *s, int flags );
int slapi_entry_attr_find( Slapi_Entry *e, char *type,
        Slapi_Attr **attr );
int slapi_entry_attr_delete( Slapi_Entry *e, char *type );
        char *slapi_entry_get_dn( Slapi_Entry *e );
void slapi_entry_set_dn(Slapi_Entry *e, char *dn);
Slapi_Entry *slapi_entry_alloc();
Slapi_Entry *slapi_entry_dup( Slapi_Entry *e);

init slapi_send_ldap_search_entry( Slapi_PBlock *pb,
        Slapi_Entry *e, LDAPControl **ectrls,
char **attrs, int attrsonly);
void slapi_entry_free( Slapi_Entry *e );
int slapi_attr_get_values( Slapi_Attr *attr, struct berval
        ***vals );

Slapi_Filter *slapi_str2filter( char *str );
init slapi_filter_get_choice( Slapi_Filter*f );
init slapi_filter_get_ava( Slapi_Filter*f, char
        *type, struct berval **bvals );
void slapi_filter_free( Slapi_Filter*f, int recurse );
Slapi_Filter *slapi_filter_list_first( Slapi_Filter*f );
Slapi_Filter *slapi_filter_list_next(Slapi_Filter*f,
        Slapi_Filter*fprev);

int slapi_is_connection_ssl( Slapi_PBlock *pPB, int *isSSL );
init slapi_get_client_port( Slapi_PBlock *pPB, int *fromPort );
```

For internal database operations:

```
Slapi_PBlock *slapi_search_internal( char *base, int scope, char *filter,
        LDAPControl **controls, char **attrs, int attrsonly );
Slapi_PBlock *slapi_modify_internal( char *dn, LDAPMod **mods,
        LDAPControl **controls );
Slapi_PBlock *slapi_add_internal( char * dn, LDAPMod **attrs,
        LDAPControl **controls );
Slapi_PBlock *slapi_add_entry_internal( Slapi_Entry * e,
        LDAPControl **controls,
        int log_change );
Slapi_PBlock *slapi_delete_internal( char * dn,
        LDAPControl **controls );
Slapi_PBlock *slapi_modrdn_internal( char * olddn,
        char * newrdn, char *newParent,
        int deloldrdn, LDAPControl **controls);
void slapi_free_search_results_internal( Slapi_PBlock *pb );

/* logging routines */
void slapi_printmessage(int catid, int level, int num, ... );
int slapi_log_error( int severity, char *subsystem, char *fmt, ... );
```

For querying server information:

```
char **slapi_get_supported_saslmechanisms();

char **slapi_get_supported_extended_ops();

void slapi_register_supported_saslmechanism( char *mechanism );

int slapi_get_supported_controls(char ***ctrloidsp,
        unsigned long **ctrlopsp);
void slapi_register_supported_control(char *controloid,
        unsigned long controlops);
int slapi_control_present( LDAPControl **controls,
        char *oid, struct berval **val,
        int * iscritical);
```

For logging routines:

```
 int slapi_log_error( int severity, char *subsystem, char *fmt, ... );
```

slapi_pblock_get()

slapi_pblock_get() receives the value of a name-value pair from a parameter block.

Syntax

```
#include "slapi-plugin.h"
int slapi_pblock_get( Slapi_PBlock *pb, int arg, void *value );
```

Parameters

pb A parameter block.

arg A pblock parameter that represents the data you want to receive.

value A pointer to the value retrieved from the parameter block.

Returns

0 if successful, or -1 if there is an error.

slapi_pblock_set()

slapi_pblock_set() sets the value of a name-value pair in a parameter block.

Syntax

```
#include "slapi-plugin.h"
int slapi_pblock_set( Slapi_PBlock *pb, int arg, void *value );
```

Parameters

pb A pointer to a parameter block.

arg The ID of the name-value pair that you want to set.

value A pointer to the value that you want to set in the parameter block. The value should be freed only if the caller is replacing the value in the pblock with a new value by calling slapi_pblock_set().

Returns

0 if successful, or -1 if an error occurs.

slapi_pblock_new()

slapi_pblock_new() creates a new parameter block.

Syntax

```
#include "slapi-plugin.h"
Slapi_PBlock *slapi_pblock_new();
```

Returns

A pointer to the new parameter block is returned.

slapi_pblock_destroy()

slapi_pblock_destroy() frees the specified parameter block from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_pblock_destroy( Slapi_PBlock *pb );
```

Parameters

pb A pointer to the parameter block that you want to free.

slapi_ch_malloc()

slapi_ch_malloc() allocates space in memory, and calls the standard malloc() C function. The slapd server is terminated with an accompanying `out of memory` error message if memory cannot be allocated.

Syntax

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

Parameters

size The amount of space that you want memory allocated for.

slapi_ch_calloc()

slapi_ch_calloc() allocates space for an array of elements of a specified size. It calls the calloc() C function. The slapd server is terminated with an accompanying `out of memory` error message if memory cannot be allocated.

Syntax

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

Parameters

nelem The number of elements that you want to allocate memory for.

size The amount of memory of each element that you want to allocate memory for.

slapi_ch_realloc()

slapi_ch_realloc() changes the size of a block of allocated memory. It calls the standard realloc() C function. The slapd server is terminated with an accompanying `out of memory` error message if memory cannot be allocated.

Syntax

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

Parameters

block A pointer to an existing block of allocated memory.

size The new amount of the block of memory you want allocated.

Returns

A pointer to a newly-allocated memory block with the requested size is returned.

slapi_ch_strdup()

slapi_ch_strdup() makes a copy of an existing string. It calls the standard strdup() C function. The slapd server is terminated with an accompanying `out of memory` error message if memory cannot be allocated.

Syntax

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

Parameters

s Refers to the string you want to copy.

Returns

A pointer to a copy of the string is returned. If space cannot be allocated (for example, if no more virtual memory exists), a NULL pointer is returned.

slapi_compare_internal()

Plug-in functions call slapi_compare_internal() to compare an entry in the backend directly.

Syntax

```
*slapi_compare_internal( const char *dn, const char *type,
         struct berval *value, LDAPControl **controls) {
```

Parameters

dn The dn of the entry on which to perform the compare. This parameter cannot have a value of NULL.

type The attribute type on which to perform the compare. This parameter cannot have a value of NULL.

value The berval value of the attribute being compared. This parameter cannot have a value of NULL.

controls

Any controls requested on the operation.

Returns

The slapi_pblock containing the return code.

slapi_ch_free()

slapi_ch_free() frees space allocated by the slapi_ch_malloc(), slapi_ch_calloc(), slapi_ch_realloc(),and slapi_ch_strdup() functions. It does not set the pointer to NULL.

Syntax

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

Parameters

ptr A pointer to the block of memory that you want to free. If it is NULL, no action occurs.

slapi_send_ldap_result()

slapi_send_ldap_result() sends an LDAP result code back to the client.

Syntax

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

Parameters

pb A pointer to a parameter block.

err The LDAP result code that you want sent back to the client.

matched
Used to specify the portion of the target DN that can be matched when you send back an LDAP_NO_SUCH_OBJECT result. Otherwise you must pass NULL.

text The error message that you want sent back to the client. If you do not want an error message sent back, pass a NULL.

nentries
Used to specify the number of matching entries found when you send back the result code for an LDAP search operation.

urls Used to specify the array of the berval structure or to specify referral URLs when you send back either an LDAP_PARTIAL_RESULTS result code to an LDAP V2 client or an LDAP_REFERRAL result code to an LDAP V3 client.

slapi_dn_normalize()

Note: A variable passed in as the DN argument is also converted in-place, therefore this API is deprecated. See “slapi_dn_normalize_v3()” on page 39.

slapi_dn_normalize() converts a distinguished name (DN) to canonical format (that is, no leading or trailing spaces, no spaces between components, and no spaces around the equals sign). As an example, for the following DN: `cn = John Doe, ou = Engineering , o = Darius` the function returns:

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

Syntax

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

Parameters

dn The DN that you want to normalize.

Returns

The normalized DN.

slapi_dn_normalize_case()

Note: A variable passed in as the DN argument is also converted in-place, therefore this API is deprecated. See "slapi_dn_normalize_case_v3()" on page 39.

slapi_dn_normalize_case() converts a distinguished name (DN) to canonical format (that is, no leading or trailing spaces, no spaces between components, and no spaces around the equals sign) and converts all characters to lower case. As an example, for the following DN: `cn = John Doe, ou = Engineering, o = Darius` the function returns:

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

Note: This function has the same effect as calling the slapi_dn_normalize() function followed by the slapi_dn_ignore_case() function.

Syntax

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

Parameters

dn The DN that you want to normalize and convert to lower case.

Returns

The normalized DN with all characters in lower case.

slapi_dn_ignore_case()

Note: A variable passed in as the DN argument is also converted in-place, therefore this API is deprecated. See "slapi_dn_ignore_case_v3()" on page 40.

slapi_dn_ignore_case() converts all of the characters in a distinguished name (DN) to lower case. As an example, for the following DN: `cn = John Doe, ou = Engineering , o = Darius` the function returns:

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

Syntax

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

Parameters

dn The DN that you want to convert to lower case.

Returns

The DN with all characters in lower case.

slapi_dn_normalize_v3()

slapi_dn_normalize_v3() converts a distinguished name(DN) to canonical format (that is, no leading or trailing spaces, no spaces between components and no spaces around the equals sign). The API normalizes the attribute type name to the first textual type name in the schema definition. Any semicolons used to separate relative distinguished names (RDN) are converted to commas. A compound RDN is sorted alphabetically by attribute name. The following is an example DN:

```
userName=johnDOE +   commonName = John Doe ;
ou = Engineering , o = Darius the function returns:
cn=John Doe+userName=johnDOE,ou=Engineering,o=Darius
```

Special characters in a DN, if escaped using double-quotes, are converted to use backslash (\ ) as the escape mechanism. For example, the following DN:

```
cn="a + b",  o=sample the function returns
cn=a \+ b,o=sample
```

An attribute value containing a backslash followed by a two-digit hex representation of a UTF-8 character is converted to the character representation. For example, the following DN:

```
cn=\4A\6F\68\6E Doe,ou=Engineering,o=Darius
the function returns cn=John Doe,ou=Engineering,o=Darius
```

A ber-encoded attribute value is converted to a UTF-8 value. For example, the following DN:

```
cn=#04044A6F686E20446F65,ou=Engineering,o=Darius
the function returns cn=John Doe,ou=Engineering,o=Darius
```

An invalid DN returns NULL.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_normalize_v3(char *dn);
```

Parameters

dn The DN that you want to normalize. It is not modified by the function.

Returns

The normalized DN in newly allocated space.

Note: It is the responsibility of the caller to free the normalized DN.

slapi_dn_normalize_case_v3()

slapi_dn_normalize_v3() converts a distinguished name (DN) to canonical format (that is, no leading or trailing spaces, no spaces between components and no spaces around the equals sign). The API normalizes the attribute type name to the first textual type name in the schema definition. Any semicolons used to separate relative distinguished names (RDN) are converted to commas. A compound RDN is sorted alphabetically by attribute name. The case of attribute types is changed to upper case in all cases. The case of the attribute values is converted to upper case only when the matching rules are case insensitive. If the matching rules for the attribute are case sensitive, the case of the attribute value is preserved. In the following example, userName is a case sensitive attribute and cn, ou and o are case insensitive. For example, the following DN:

```
userName=johnDOE +   commonName = John Doe ;
ou = Engineering , o = Darius the function returns:
CN=JOHN DOE+USERNAME=johnDOE,OU=ENGINEERING,O=DARIUS
```

Special characters in a DN, if escaped using double-quotes, are converted to use backslash (\ ) as the escape mechanism. For example, the following DN:

```
cn="a + b",  o=sample the function returns
  CN=A \+ B,o=sample
```

An attribute value containing a backslash followed by a two-digit hex representation of a UTF-8 character is converted to the character representation. For example, the following DN:

```
cn=\4A\6F\68\6E Doe,ou=Engineering,o=Darius
the function returns CN=JOHN DOE,OU=ENGINEERING,O=DARIUS
```

A ber-encoded attribute value is converted to a UTF-8 value. For example, the following DN:

```
cn=#04044A6F686E20446F65,ou=Engineering,o=Darius
the function returns CN=JOHN DOE,OU=ENGINEERING,O=DARIUS
```

An invalid DN returns NULL.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_normalize_case_v3(char *dn);
```

Parameters

dn The DN that you want to normalize and convert to lower case. It is not modified by the function.

Returns

The normalized DN in newly allocated space.

Note: It is the caller's responsibility to free the normalized DN.

slapi_dn_ignore_case_v3()

slapi_dn_ignore_case_v3() normalizes a distinguished name (DN) and converts all of the characters to lower case. For example, the following DN:

```
userName=johnDOE +   commonName = John Doe ;
ou = Engineering , o = Darius
the function returns:
cn=john doe+username=johndoe,ou=engineering,o=darius
```

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_ignore_case _v3(char *dn);
```

Parameters

dn The DN that you want to normalize and convert to lower case.

Returns

The DN normalized with all characters in lower case.

Note: It is the caller's responsibility to free the normalized DN.

slapi_dn_compare_v3()

slapi_dn_compare_v3() compares two distinguished names (DN).

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_compare_v3(char *dn1, char* dn2);
```

Parameters

dn1 A DN that you want to compare.

dn2 A DN that you want to compare.

Returns

- Less than 0 if the value of dn1 is lexicographically less than dn2.
- 0 if the value of dn1 is lexicographically equal to dn2.
- Greater than 0 if the value of dn1 is lexicographically greater than dn2.

slapi_dn_issuffix()

slapi_dn_issuffix() determines whether a DN is equal to the specified suffix.

Syntax

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

Parameters

dn The DN that you want to check.

suffix The suffix you want compared against the DN.

Returns

A 1 is returned if the specified DN is the same as the specified suffix, or a 0 is returned if the DN is not the same as the suffix.

slapi_entry2str()

slapi_entry2str() generates a description of an entry as a string. The LDIF string has the following format:

```
dn: <dn>\n
*[<attr>: <value>\n]
*[<attr>:: <base_64_encoded_value>]
```

where:

* The operator "*" when it precedes an element indicates repetition. The full form is: <a>*<b> where <a> and <b> are optional decimal values, indicating at least <a> and at most <b> occurrences of element.

Default values are 0 and infinity so that *<*element*> allows any number, including zero; 1*<*element*> requires at least one; 3*3<*element*> allows exactly 3 and 1*2<*element*> allows one or two.

dn Distinguished name

attr Attribute name

\n New line

value Attribute value

For example:

```
dn: uid=rbrown2, ou=People, o=airius.com

cn: Robert Brown
```

```
sn: Brown
...
```

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

Call the slapi_str2entry() function to convert a string description in this format to an entry of the Slapi_Entry data type.

Syntax

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

Parameters

e Address of the entry that you want to generate a description for.

len Address of the length of the returned string.

Returns

The description of the entry as a string is returned or NULL if an error occurs.

slapi_str2entry()

slapi_str2entry() converts an LDIF description of a directory entry (a string value) into an entry of the Slapi_Entry data type that can be passed to other API functions.

Note: This function modifies the ***s*** string argument, and you must make a copy of this string before it is called.

If there are errors during the conversion process, the function returns a NULL instead of the entry.

When you are through working with the entry, call the slapi_entry_free() function. To convert an entry to a string description, call slapi_entry2str().

Syntax

```
#include "slapi-plugin.h"
Slapi_Entry *slapi_str2entry( char *s, int flags );
```

Parameters

s The description of an entry that you want to convert.

flags Specifies how the entry must be generated.

The ***flags*** argument can be one of the following values:

SLAPI_STR2ENTRY_REMOVEDUPVALS
Removes any duplicate values in the attributes of the entry.

SLAPI_STR2ENTRY_ADDRDNVALS
Adds the relative distinguished name (RDN) components.

Returns

A pointer to the Slapi_Entry structure representing the entry is returned, or a NULL is returned if the string cannot be converted, for example, if no DN is specified in the string.

slapi_entry_attr_find()

slapi_entry_attr_find() determines if an entry has a specified attribute. If it does, this function returns that attribute.

Syntax

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

Parameters

e An entry that you want to check.

type Indicates the name of the attribute that you want to check.

attr A pointer to the attribute (assuming that the attribute is in the entry).

Returns

A 0 is returned if the entry contains the specified attribute, or -1 is returned if it does not.

slapi_entry_attr_delete()

slapi_entry_attr_delete() deletes an attribute from an entry.

Syntax

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

Parameters

e The entry from which you want to delete the attribute.

type Indicates the name of the attribute that you want to delete.

Returns

A 0 is returned if the attribute is successfully deleted, a 1 is returned if the specified attribute is not part of the entry, or -1 is returned if an error has occurred.

slapi_entry_get_dn()

slapi_entry_get_dn() receives the DN of the specified entry.

Syntax

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

Parameters

e Indicates an entry that contains the DN you want.

Returns

The DN of the entry is returned. A pointer to the actual DN in the entry is returned, not a copy of the DN.

slapi_entry_set_dn()

slapi_entry_set_dn() sets the DN of an entry. It sets the pointer to the DN that you specify.

Note: Because the old DN is not overwritten and is still in memory, you need to first call slapi_entry_get_dn() to get the pointer to the current DN, free the DN, and then call slapi_entry_set_dn() to set the pointer to your new DN.

Syntax

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

Parameters

e Indicates the entry to which you want to assign the DN.

dn The DN that you want to assign to the entry.

slapi_entry_alloc()

slapi_entry_alloc() allocates memory for a new entry of the Slapi_Entry data type. It returns an empty Slapi_Entry structure. You can call other front-end functions to set the DN and attributes of this entry. When you are through working with the entry, free it by calling the slapi_entry_free() function.

Syntax

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

Returns

A pointer to the newly allocated entry of the Slapi_Entry data type is returned. If space cannot be allocated (for example, if no more virtual memory exists), the server program ends.

slapi_entry_dup()

slapi_entry_dup() makes a copy of an entry, its DN, and its attributes. You can call other front-end functions to change the DN and attributes of this copy of an existing Slapi_Entry structure. When you are through working with the entry, free it by calling the slapi_entry_free() function.

Syntax

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

Parameters

e The entry that you want to copy.

Returns

The new copy of the entry. If the structure cannot be duplicated (for example, if no more virtual memory exists), the server program ends.

slapi_send_ldap_search_entry()

slapi_send_ldap_search_entry() sends an entry found by a search back to the client.

Syntax

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

Parameters

pb The parameter block.

e The pointer to the Slapi_Entry structure representing the entry that you want to send back to the client.

ectrls The pointer to the array of LDAPControl structures that represent the controls associated with the search request.

attrs Attribute types specified in the LDAP search request.

attrsonly
Specifies whether the attribute values must be sent back with the result.

- If set to 0, the values are included.
- If set to 1, the values are not included.

Returns
A 0 is returned if successful, a 1 is returned if the entry is not sent (for example, if access control did not allow it to be sent), or a -1 is returned if an error occurs.

slapi_entry_free()

slapi_entry_free() frees an entry, its DN, and its attributes from memory.

Syntax

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

Parameters

e An entry that you want to free. If it is NULL, no action occurs.

slapi_attr_get_values()

slapi_attr_get_values() receives the value of the specified attribute.

Syntax

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

Parameters

attr An attribute that you want to get the flags for.

vals When slapi_attr_get_values() is called, ***vals*** is set to a pointer that indicates a NULL-terminated array of berval structures (representing the values of the attribute). Do not free the array; the array is part of the actual data in the attribute, not a copy of the data.

Returns
A 0 is returned if it is successful.

slapi_str2filter()

slapi_str2filter() converts a string description of a search filter into a filter of the Slapi_Filter type. When you are done working with this filter, free the Slapi_Filter structure by calling slapi_filter_free().

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_str2filter( char *str );
```

Parameters

str A string description of a search filter.

Returns

The address of the Slapi_Filter structure representing the search filter is returned, or a NULL is returned if the string cannot be converted (for example, if an empty string is specified or if the filter syntax is incorrect).

slapi_filter_get_choice()

slapi_filter_get_choice() gets the type of the specified filter (for example, LDAP_FILTER_EQUALITY).

Syntax

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

Parameters

f The filter type that you want to get.

Returns

One of the following values is returned:

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

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

LDAP_FILTER_NOT (NOT filter)
For example: `(!(l=Sunnyvale))`

LDAP_FILTER_EQUALITY (equals filter)
For example: `(ou=Accounting)`

LDAP_FILTER_SUBSTRINGS (substring filter)
For example: `(ou=Account*Department)`

LDAP_FILTER_GE (″greater than or equal to″ filter)
For example: `(supportedLDAPVersion>=3)`

LDAP_FILTER_LE (″less than or equal to″ filter)
For example: `(supportedLDAPVersion<=2)`

LDAP_FILTER_PRESENT (presence filter)
For example: `(mail=*)`

LDAP_FILTER_APPROX (approximation filter)
For example: `(ou~=Sales)`

slapi_filter_get_ava()

slapi_filter_get_ava() gets the attribute type and the value from the filter. This applies only to filters of the types LDAP_FILTER_EQUALITY, LDAP_FILTER_GE, LDAP_FILTER_LE, LDAP_FILTER_APPROX. These filter types generally compare a value against an attribute. For example: `(cn=John Doe)` This filter finds entries in which the value of the cn attribute is equal to John Doe.

Calling the slapi_filter_get_ava() function gets the attribute type and value from this filter. In the case of the example, calling the slapi_filter_get_ava() function gets the attribute type cn and the value John Doe.

Syntax

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

Parameters

f The address of the filter from which you want to get the attribute and value.

type The pointer to the attribute type of the filter.

bval The pointer to the address of the berval structure that contains the value of the filter.

Returns

A 0 is returned if successful, or a -1 is returned if the filter is not one of the types listed.

slapi_filter_free()

slapi_filter_free() frees the specified filter and (optionally) the set of filters that comprise it. For example, the set of filters in an LDAP_FILTER_AND type filter.

Syntax

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

Parameters

f The filter that you want to free.

recurse

If set to 1, it recursively frees all filters that comprise this filter. If set to 0, it only frees the filter specified by the **f** parameter.

slapi_filter_list_first()

slapi_filter_list_first() gets the first filter that makes up the specified filter. This applies only to filters of the types LDAP_FILTER_EQUALITY, LDAP_FILTER_GE, LDAP_FILTER_LE, and LDAP_FILTER_APPROX. These filter types generally consist of one or more other filters. For example, if the filter is: `(&(ou=Accounting)(l=Sunnyvale))` the first filter in this list is: `(ou=Accounting)`. Use the slapi_filter_list_first() function to get the first filter in the list.

Syntax

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

Parameters

f The filter from which you want to get the first component.

Returns

The first filter that makes up the filter specified by the **f** parameter is returned.

slapi_filter_list_next()

slapi_filter_list_next() gets the next filter (following fprev) that makes up the specified filter **f**. This applies only to filters of the types LDAP_FILTER_EQUALITY, LDAP_FILTER_GE, LDAP_FILTER_LE, and LDAP_FILTER_APPROX. These filter types generally consist of one or more other filters. For example, if the filter is: `(&(ou=Accounting)(l=Sunnyvale))` the next filter after `(ou=Accounting)` in this list is: `(l=Sunnyvale)`. Use the slapi_filter_list_first() function to get the first filter in the list.

To iterate through all filters that make up a specified filter, call the slapi_filter_list_first() function and then call slapi_filter_list_next().

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_list_next( Slapi_Filter
*f, Slapi_Filter *fprev );
```

Parameters

f The filter from which you want to get the next component (after fprev).

fprev A filter within the filter specified by the **f** parameter.

Returns

The next filter (after fprev) that makes up the filter specified by the **f** parameter is returned.

slapi_is_connection_ssl()

slapi_is_connection_ssl() is used by the server to determine whether the connection between it and a client is through a Secure Socket Layer (SSL).

Syntax

```
#include "slapi-plugin.h"
int slapi_is_connection_ssl( Slapi_PBlock *pPB,
int *isSSL);
```

Parameters

pPB Address of a Parameter Block.

isSSL Address of the output parameter. A 1 is returned if the connection is through SSL or a 0 is returned if it is not through SSL.

Returns

A 0 is returned if successful.

slapi_get_client_port()

slapi_get_client_port() is used by the server to determine the port number used by a client to communicate to the server.

Syntax

```
#include "slapi-plugin.h"
int slapi_get_client_port( Slapi_PBlock *pPB,
int *fromPort);
```

Parameters

pPB Address of a Parameter Block.

fromPort
Address of the output parameter. It is the port number used by the client.

Returns
A 0 is returned if successful.

slapi_search_internal()

slapi_search_internal() performs an LDAP search operation to search the directory from your plug-in.

Syntax

```
#include "slapi-plugin.h"
Slapi_PBlock *slapi_search_internal( char *base, int scope,
        char *filter, LDAPControl **controls,
        char **attrs, int attrsonly );
```

Parameters

base The DN of the entry that serves as the starting point for the search. For example, setting base `o=Acme Industry, c=US` restricts the search to entries at Acme Industry located in the United States.

scope Defines the scope of the search. It can be one of the following values:

- LDAP_SCOPE_BASE searches the entry that is specified by *base*.
- LDAP_SCOPE_ONELEVEL searches all entries one level beneath the entry specified by *base*.
- LDAP_SCOPE_SUBTREE searches the entry specified by *base*. It also searches all entries at all levels beneath the entry specified by base .

filter The string representation of the filter to apply in the search.

controls
The NULL-terminated array of LDAP controls that you want applied to the search operation.

attrs The NULL-terminated array of attribute types to return from entries that match the filter. If you specify a NULL, all attributes are returned.

attrsonly
Specifies whether or not attribute values are returned along with the attribute types. It can have the following values:

- A 0 specifies that both attribute types and attribute values are returned.
- A 1 specifies that only attribute types are returned.

Returns
slapi_free_search_results_internal() and slapi_pblock_destroy() need to be called to free the search results and the pblock that is returned by slapi_search_internal.

slapi_modify_internal()

slapi_modify_internal() performs an LDAP modify operation to modify an entry in the directory from a plug-in.

Unlike the standard LDAP modify operation, no LDAP result code is returned to a client; the result code is placed instead in a parameter block that is returned by the function.

Syntax

```
#include "slapi-plugin.h"
Slapi_PBlock *slapi_modify_internal( char *dn,
        LDAPMod **mods,
        LDAPControl **controls, int l );
```

Parameters

dn A distinguished name (DN) of the entry that you want to modify.

mods A pointer to a NULL-terminated array of pointers to LDAPMod structures representing the attributes that you want to modify.

controls
A NULL-terminated array of LDAP controls.

l Included for compatibility only. It is not used.

Returns

A new parameter block with the following parameter set is returned:

- SLAPI_PLUGIN_INTOP_RESULT specifies the LDAP result code for the internal LDAP operation.

slapi_add_internal()

slapi_add_internal() performs an LDAP add operation in order to add a new directory entry (specified by a DN and a set of attributes) from your plug-in. Unlike the standard LDAP add operation, no LDAP result code is returned to a client. The result code is instead placed in a parameter block that is returned by the function.

Syntax

```
#include "slapi-plugin.h"
Slapi_PBlock *slapi_add_internal( char * dn,
        LDAPMod **mods,
        LDAPControl **controls, int l );
```

Parameters

dn The Distinguished name (DN) of the entry that you want to add.

mods A pointer to a NULL-terminated array of pointers to LDAPMod structures representing the attributes of the new entry that you want to add.

controls
A NULL-terminated array of LDAP controls that you want applied to the add operation.

l Included for compatibility only. It is not used.

Returns

A new parameter block with the following parameter set is returned:

- SLAPI_PLUGIN_INTOP_RESULT specifies the LDAP result code for the internal LDAP operation.

slapi_add_entry_internal()

slapi_add_entry_internal() performs an LDAP add operation to add a new directory entry (specified by an Slapi_Entry structure) from a plug-in function. Unlike the standard LDAP add operation, no LDAP result code is returned to a client. Instead, the result code is placed in a parameter block that is returned by the function.

Note: To add an entry specified by a string DN and an array of LDAPMod structures, call slapi_add_internal() instead.

Syntax

```
#include "slapi-plugin.h"
Slapi_PBlock *slapi_add_entry_internal( Slapi_Entry * e,
   LDAPControl **controls, int l );
```

Parameters

mods A pointer to an Slapi_Entry structure representing the new entry that you want to add.

controls
A NULL-terminated array of LDAP controls that you want applied to the add operation.

l Included for compatibility only. It is not used.

Returns

A new parameter block with the following the following parameter set is returned:

- SLAPI_PLUGIN_INTOP_RESULT specifies the LDAP result code for the internal LDAP operation (for example, LDAP_SUCCESS if the operation is successful or LDAP_PARAM_ERROR if an invalid parameter is used).

  If the DN of the new entry has a suffix that is not served by the Directory Server, SLAPI_PLUGIN_INTOP_RESULT is set to LDAP_REFERRAL.

slapi_delete_internal()

slapi_delete_internal() performs an LDAP delete operation in order to remove a directory entry when it is called from your plug-in.

Unlike the standard LDAP delete operation, no LDAP result code is returned to a client. The result code is instead placed in a parameter block that is returned by the function.

Syntax

```
#include "slapi-plugin.h"
Slapi_PBlock *slapi_delete_internal( char * dn,
   LDAPControl **controls, int l );
```

Parameters

dn The distinguished name (DN) of the entry that you want to delete.

controls
A NULL-terminated array of LDAP controls that you want applied to the delete operation.

l Included for compatibility only. It is not used.

Returns

A new parameter block with the following parameter set is returned:

- SLAPI_PLUGIN_INTOP_RESULT specifies the LDAP result code for the internal LDAP operation.

slapi_modrdn_internal()

slapi_modrdn_internal() performs an LDAP modify RDN operation in order to rename a directory entry from your plug-in.

Unlike the standard LDAP modify RDN operation, no LDAP result code is returned to a client. The result code is instead placed in a parameter block that is returned by the function.

Syntax

```
#include "slapi-plugin.h"
Slapi_PBlock *slapi_modrdn_internal( char * olddn,
   char * newrdn, int deloldrdn, LDAPControl **controls,
   int l);
```

Parameters

olddn The distinguished name (DN) of the entry that you want to rename.

newdn The new relative distinguished name (RDN) of the entry.

deloldrdn

Specifies whether or not you want to remove the old RDN from the entry.

- If a 1, remove the old RDN.
- If a 0, leave the old RDN as an attribute of the entry.

controls

A NULL-terminated array of LDAP controls that you want applied to the modify RDN operation.

l Included for compatibility only. It is not used.

Returns

A new parameter block with the following parameter set is returned:

- SLAPI_PLUGIN_INTOP_RESULT specifies the LDAP result code for the internal LDAP operation.

slapi_free_search_results_internal()

slapi_free_search_results_internal() frees the memory associated with LDAP entries returned by the search.

Syntax

```
#include "slapi-plugin.h"
void slapi_free_search_results_internal( Slapi_PBlock *pb);
```

Parameters

pb Is a pointer to a Parameter Block that is returned by a slapi_free_search_internal function.

slapi_get_supported_saslmechanisms()

slapi_get_supported_saslmechanisms() obtains an array of the supported Simple Authentication and Security Layer (SASL) mechanisms names. Register new SASL mechanisms by calling the slapi_register_supported_saslmechanism() function.

Syntax

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

Returns

A pointer to an array of SASL mechanisms names supported by the server is returned.

slapi_get_supported_extended_ops()

slapi_get_supported_extended_ops() gets an array of the object IDs (OIDs) of the extended operations supported by the server. Register new extended operations by putting the OID in the SLAPI_PLUGIN_EXT_OP_OIDLIST parameter and calling the slapi_pblock_set() function.

Syntax

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

Returns

A pointer to an array of the OIDs of the extended operations supported by the server is returned.

slapi_register_supported_saslmechanism()

slapi_register_supported_saslmechanism() registers the specified Simple Authentication and Security Layer (SASL) mechanism with the server.

Syntax

```
#include "slapi-plugin.h"
void slapi_register_supported_saslmechanism( char *mechanism );
```

Parameters

mechanism
Indicates the name of the SASL mechanism.

slapi_get_supported_controls()

slapi_get_supported_controls() obtains an array of OIDs, which represent the controls supported by the directory server. Register new controls by calling the slapi_register_supported_control() function.

Syntax

```
#include "slapi-plugin.h"
int slapi_get_supported_controls( char ***ctrloidsp,
   unsigned long **ctrlopsp );
```

Parameters

ctrloidsp
A pointer to an array of OIDs, which represent the controls supported by the server.

ctrlopsp
A pointer to an array of IDs which specify LDAP operations that support each control.

Returns
A 0 is returned if successful.

slapi_register_supported_control()

slapi_register_supported_control() registers the specified control with the server. It also associates the control with an OID. When the server receives a request that specifies this OID, the server makes use of this information in order to determine if the control is supported.

Syntax

```
#include "slapi-plugin.h"
void slapi_register_supported_control( char *controloid,
unsigned long controlops );
```

Parameters

controloid
The OID of the control you want to register.

controlops
The operation that the control is applicable to. It can have one or more of the following values:

SLAPI_OPERATION_BIND
The specified control that applies to the LDAP bind operation.

SLAPI_OPERATION_UNBIND
The specified control that applies to the LDAP unbind operation.

SLAPI_OPERATION_SEARCH
The specified control that applies to the LDAP search operation.

SLAPI_OPERATION_MODIFY
The specified control that applies to the LDAP modify operation.

SLAPI_OPERATION_ADD
The specified control that applies to the LDAP add operation.

SLAPI_OPERATION_DELETE
The specified control that applies to the LDAP delete operation.

SLAPI_OPERATION_MODDN
The specified control that applies to the LDAP modify DN operation.

SLAPI_OPERATION_MODRDN
The specified control that applies to the LDAP V3 modify RDN operation.

SLAPI_OPERATION_COMPARE
The specified control that applies to the LDAP compare operation.

SLAPI_OPERATION_ABANDON
The specified control that applies to the LDAP abandon operation.

SLAPI_OPERATION_EXTENDED
The specified control that applies to the LDAP V3 extended operation.

SLAPI_OPERATION_ANY
The specified control that applies to any LDAP operation.

SLAPI_OPERATION_NONE
The specified control that applies to none of the LDAP operations.

slapi_control_present()

slapi_control_present() determines whether or not the specified OID identifies a control that might be present in a list of controls.

Syntax

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

Parameters

controls
The list of controls that you want to check.

oid Refers to the OID of the control that you want to find.

val Specifies the pointer to the berval structure containing the value of the control (if the control is present in the list of controls).

iscritical
Specifies whether or not the control is critical to the operation of the server (if the control is present in the list of controls).

- A 0 means that the control is not critical to the operation.
- A 1 means that the control is critical to the operation.

Returns
A 1 is returned if the specified control is present in the list of controls, or a 0 if the control is not present.

slapi_log_error()

Writes a message to the error log for the directory server.

Syntax

```
#include "slapi-plugin.h"
int slapi_log_error( int severity, char *subsystem, char *fmt, ... );
```

Parameters

severity
Level of severity of the message. In combination with the severity level specified by ibm-slapdSysLogLevel in the ibmslapd.conf file, determines whether or not the message is written to the log. The severity must be one of the following:

- LDAP_MSG_LOW
- LDAP_MSG_MED

- LDAP_MSG_HIGH

The following entry in the ibmslapd.conf file results in a medium logging level:

```
#ibm-slapdSysLogLevel must be one of l/m/h (l=terse, h=verbose)
ibm-slapdSysLogLevel:  m
```

With this example in your ibmslapd.conf file, log messages with severity LDAP_MSG_HIGH or LDAP_MSG_MED are logged. The messages with severity LDAP_MSG_LOW are not logged. If the slapdSysLogLevel is set to `h`, all messages are logged.

subsystem
Name of the subsystem in which this function is called. The string that you specify here appears in the error log in the following format:

```
<subsystem>: <message>
```

fmt, ... Message that you want written. This message can be in printf()-style format. For example:

```
..., "%s\n", myString);
```

Returns
A 0 is returned if successful, -1 if an unknown severity level is specified.

Appendix D. SLAPI API Categories

The following SLAPI APIs are supported by IBM Tivoli Directory Server.

- “slapi_dn2ldapdn()”
- “slapi_dn_get_rdn()” on page 58
- “slapi_dn_get_rdn_count()” on page 59
- “slapi_dn_free_ldapdn()” on page 59
- “slapi_dn_free_rdn()” on page 60
- “slapi_get_response_controls()” on page 60
- “slapi_set_response_controls()” on page 61
- “slapi_moddn_internal()” on page 62
- “slapi_get_client_ip()” on page 63
- “slapi_get_proxied_dn()” on page 64
- “slapi_get_source_ip()” on page 65

slapi_dn2ldapdn()

Purpose

This routine converts a DN string to an internal SLAPI_LDAPDN structure.

Syntax

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

int slapi_dn2ldapdn(
   char          *dn,
   SLAPI_LDAPDN   **ldapdn);
```

Input parameters

dn This parameter specifies the DN to be parsed. The DN must be normalized and should be in UTF-8 format.

ldapdn
This parameter specifies the address of an internal SLAPI_LDAPDN structure. This returned structure should be used as an input parameter to other DN-related SLAPI calls.

Usage

This routine converts a DN string to a SLAPI_LDAPDN structure. This structure is an LDAP internal DN structure and should be used as an input parameter for other DN-related SLAPI calls, such as slapi_dn_get_rdn() and slapi_dn_get_rdn_count().

After using the SLAPI_LDAPDN structure, the caller should free the SLAPI_LDAPDN structure by calling slapi_dn_free_ldapdn().

Errors

This routine returns an LDAP error code if it encounters an error while parsing the DN.

See also

slapi_dn_free_ldapdn(), slapi_dn_get_rdn(), and slapi_dn_get_rdn_count()

slapi_dn_get_rdn()

Purpose

This routine gets an RDN that make up the specified DN.

Syntax

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

int slapi_dn_get_rdn(
   SLAPI_LDAPDN   *ldapdn,
   long           rdnOrder,
   char           **strRDN,
   Slapi_ldapRDN   ***ldapRDNs);
```

Input parameters

ldapdn
This parameter specifies the address of an internal SLAPI_LDAPDN structure. The address of this structure is obtained by calling slapi_dn2ldapdn().

rdnOrder
This parameter specifies the order of a RDN in a DN. The rdnOrder for the left-most RDN is 1.

Output parameters

strRDN
This parameter specifies the address of a pointer that points to the requested RDN.

ldapRDNs
This parameter specifies the address of a NULL terminated array of pointers that points to the attribute types/values which make up the specified RDN. For instance, for a compound RDN “cn=Joe Smith+uid=12345”, the output will be an array that consists of three elements with the first element pointing to a Slapi_ldapRDN structure that points to “cn” and “Joe Smith”, the second element pointing to a Slapi_ldapRDN structure that points to “uid” and “12345”, and the third element being a NULL pointer.

Usage

This routine is used to obtain the desired RDN in a DN by using the order number of the RDN. The order number of the left-most RDN is 1.

For instance, for extracting the RDN “ou=Austin” from a DN “cn=Joe Smith+uid=12345, ou=Austin,o=sample”, the input parameter to the function is a SLAPI_LDAPDN structure that can be obtained by calling slapi_dn2ldapdn(), and a rdnNumber of 2. In this case, the output will be a string value, “ou=Austin”, and an array consisting of two elements with the first element pointing to a Slapi_ldapRDN structure and the second element a NULL pointer. The Slapi_ldapRDN structure defined in the slapi-plugin.h file has two char pointers pointing to “ou” and “Austin”, respectively. The user should free the returned RDN string by calling slapi_ch_free() and the returned array of Slapi_ldapRDN structure by calling slapi_dn_free_rdn().

Errors

This routine returns an LDAP error code if it encounters an error while parsing the RDN.

See also

slapi_dn2ldapdn() and slapi_dn_get_rdn_count()

slapi_dn_get_rdn_count()

Purpose

This routine returns the number of RDNs in a DN.

Syntax

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

long slapi_dn_get_rdn_count(
   SLAPI_LDAPDN   *ldapdn);
```

Input parameters

ldapdn
This parameter specifies the address of a SLAPI_LDAPDN structure. The address of the structure can be obtained by calling slapi_dn2ldapdn().

Usage

This routine obtains the number of RDNs in a DN.

Errors

This routine returns the number of RDNs in a LDAP DN structure.

See also

slapi_dn2ldapdn() and slapi_dn_get_rdn()

slapi_dn_free_ldapdn()

Purpose

This routine frees the SLAPI_LDAPDN structure. This structure should be allocated and returned by calling slapi_dn2ldapdn().

Syntax

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

void slapi_dn_free_ldapdn(
   SLAPI_LDAPDN   **ldapdn);
```

Input parameters

ldapdn

This parameter specifies the address of an address of a SLAPI_LDAPDN structure. The address of a SLAPI_LDAPDN structure should be an address returned by slapi_dn2ldapdn().

Usage

This routine frees the memory allocated by slapi_dn2ldapdn(). This function takes the address of an address of a SLAPI_LDAPDN structure.

See also

slapi_dn2ldapdn()

slapi_dn_free_rdn()

Purpose

This routine frees all the Slapi_ldapRDN structures pointed by an array of Slapi_ldapRDN pointers including the memory allocated for the array itself. The array address should be the address returned by slapi_dn_get_rdn().

Syntax

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

void slapi_dn_free_rdn(
   Slapi_ldapRDN   **ldapRDNs);
```

Input parameters

ldapRDNs

This parameter specifies the address of an array of address to the Slapi_ldapRDN structures. This Slapi_ldapRDN address should be the address returned by slapi_dn_get_rdn().

Usage

This routine frees the memory allocated for all the components, including the attribute types and attribute values specified in a RDN.

See also

slapi_dn_get_rdn()

slapi_get_response_controls()

Purpose

This slapi routine calls and accesses the list of response controls.

Syntax

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

```
int slapi_get_response_controls(
         Slapi_PBlock       *pb,
         LDAPControl        ***responseControls);
```

Input parameters

pb A parameter block.

Slapi_PBlock must contain the following:

- SLAPI_CONNECTION - Connection structure representing the client.
- SLAPI_OPERATION - Operation structure

Output parameters

responseControls
Specifies a pointer that returns a deep copy of the response controls that the server currently has associated with the operation. Response controls are the controls that are returned to the client when the response is sent.

Return codes are listed:

- LDAP_SUCCESS – Successfully retrieved the list of controls.
- LDAP_PARAM_ERROR – Parameters passed in were invalid.

Usage

The slapi_get_response_controls() routine should be called when the program needs to access the list of response controls that the server has associated with a single operation. The caller must free the local list of controls after its use.

slapi_set_response_controls()

Purpose

This slapi routine sets the list of response controls.

Note: The current list of response controls will be entirely replaced with the new list.

Syntax

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

int slapi_set_response_controls(
         Slapi_PBlock       *pb,
         LDAPControl        ***responseControls);
```

Input parameters

pb A parameter block.

Slapi_PBlock must contain the following:

- SLAPI_CONNECTION - Connection structure representing the client.
- SLAPI_OPERATION - Operation structure

Output parameters

Returns the following LDAP return code:

- LDAP_SUCCESS - The controls were successfully set on the operation.
- LDAP_NO_MEMORY - Server ran out of memory while processing the request.
- LDAP_INVALID_PARAM - The parameters for the function are invalid.
- LDAP_UNWILLING_TO_PERFORM - The list of response controls contains an unsupported control.

Usage

The slapi_set_response_controls() routine should be called when the program needs to replace all the response controls with a new list of response controls. This list of LDAPControls that are passed should not be freed.

slapi_moddn_internal()

Purpose

This slapi routine moves an entry that is under a parent entry to another parent entry. In addition, it allows changing the RDN portion in a DN.

Syntax

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

Slapi_PBlock *slapi_moddn_internal(
          char         *olddn,
          char         *newrdn,
          char         *newsuperior,
          int          deloldrdn,
          LDAPControl  **controls,
          int          l);
```

Input parameters

olddn Specifies the distinguished name (DN) of an entry that is to be renamed.

newrdn
Specifies the new relative distinguished name (RDN) of an entry.

newsuperior
Specifies the DN of the parent entry to which the entry is being moved. This is provided when the entry is being moved to a new location in the directory tree.

deloldrdn
Specifies whether or not the old RDN from the entry should be removed.

If the value is 1, remove the old RDN.

If the value is 0, leave the RDN as an attribute of the entry.

controls
A NULL-terminated array of LDAP controls that is used in the modify RDN operation.

l Used for compatibility with slapi APIs provided by other vendors. It is not used.

Returns

A new parameter block with the following parameter set is returned. The result code SLAPI_PLUGIN_INTOP_RESULT specifies the LDAP result code for the internal LDAP operation.

Usage

This slapi routine moves an entry that is under a specific parent entry to another parent entry. In addition, it allows changing the RDN portion in a DN.

For example, if we provide the following information's to the API:

```
newsuperior = "o=ABC, c=YZ"
olddn = "cn=Modify Me, o=PQR,  c=XY"
newrdn  = "cn=The New Me"
deloldrdn = 1
```

In this case, the API modifies the RDN of the Modify Me entry from "Modify Me" to "The New Me". In addition, the entry is moved from "o=PQR, c=XY" to "o=ABC, c=YZ".

Error

This routine returns an error code SLAPI_PLUGIN_INTOP_RESULT, if it encounters an error.

See also

slapi_modrdn_internal()

slapi_get_client_ip()

Purpose

This slapi routine returns the IP address of the client that is bound to the server.

Syntax

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

int slapi_get_client_ip(
        Slapi_PBlock     *pb,
        char             **clientIP );
```

Input parameters

pb A pointer to a parameter block.

clientIP
The IP address of the client connection.

Returns

If the return code is LDAP_SUCCESS and the client IP is set, this API retrieves the IP address of the client connection.

Usage

A slapi API that returns the IP address of the client that is bound to the server.

Note: The user must free the returned client IP after its use.

Error

This API returns the following error codes:

- LDAP_PARAM_ERROR - If the pb parameter is null.
- LDAP_OPERATIONS_ERROR – If the API encounters error processing the request.
- LDAP_NO_MEMORY - Failed to allocate required memory.

See also

slapi_get_source_ip()

slapi_get_proxied_dn()

Purpose

This slapi routine returns the proxied DN of the client.

Syntax

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

int slapi_get_proxied_dn(
         lapi_PBlock      *pb,
         char             **proxiedDN );
```

Input parameters

pb A pointer to a parameter block.

proxiedDN
The DN that is used for the connection.

Returns

If the return code is LDAP_SUCCESS and proxiedDN is set, this DN is used for the operation. If the return code is LDAP_SUCCESS and proxiedDN is not set, then the proxy auth control was not called.

Usage

A slapi API that returns the proxied DN of the client.

Note: The user must free the returned proxiedDN after its use.

Error

This API returns the following error codes:

- LDAP_PARAM_ERROR - If the pb parameter is null.
- LDAP_OPERATIONS_ERROR – If the API encounter error processing the request.
- LDAP_NO_MEMORY - Failed to allocate required memory.

See also

slapi_entry_get_dn()

slapi_get_source_ip()

Purpose

This slapi routine returns the IP address sent in the audit control.

Syntax

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

int slapi_get_source_ip(
       Slapi_PBlock      *pb,
       char              ** sourceIP );
```

Input parameters

pb A pointer to a parameter block.

sourceIP
The IP address of the connection source.

Returns

If the return code is LDAP_SUCCESS and sourceIP is set, then this source IP is used for connection.

Usage

A slapi API that returns the IP address sent in the audit control.

Note: The user must free the returned sourceIP after its use. In addition, it must be checked that the clientIP is from a trusted proxy web admin or application.

Error

This API returns the following error codes:

- LDAP_PARAM_ERROR - If the pb parameter is null.
- LDAP_OPERATIONS_ERROR - If the API encounter error processing the request.
- LDAP_NO_MEMORY - Failed to allocate required memory.

See also

slapi_get_client_ip()

Appendix E. Plug-in examples

The following sample C code creates a simple SASL bind plug-in that uses the mechanism SAMPLE_BIND. It compares the password that is sent across the wire to the password stored in the directory for the given bind DN. It is important to realize that this example is meant only to illustrate the basic operation of servicing a simple bind request, and how the operations are implemented by way of a user developed plug-in. Actual processing of a simple bind request as part of the fundamental operation of the LDAP server involves significantly more processing.

```
#include <stdio.h>
#include <string.h>
#include <strings.h>

#include <slapi-plugin.h>

#define FALSE 0

/* Let the next plugin try the operation */
#define NEXTPLUGIN 0
/* We handled the operation, so don't run any other plugins */
#define STOP_PLUGIN_SEARCH 1

/* SASL mechanism type */
#define SAMPLE_MECH "SAMPLE_BIND"

/* Subsystem to use for slapi_log_error calls */
#define SAMPLE_SUBSYSTEM "SAMPLE"

/* Filter used when searching for the entry DN */
#define FILTER "objectclass=*"
/* Password attribute name */
#define PWATTR "userpassword"

/* Forward declaration of our bind plugin function */
int sampleBind(Slapi_PBlock *pb);

/* Initialization function */
int sampleInit(Slapi_PBlock *pb)
{
    int argc = 0;
    char ** argv = NULL;

    /* to register the Sample_Bind function as the pre-operation
     * bind funtion
     */
    if (slapi_pblock_set( pb, SLAPI_PLUGIN_PRE_BIND_FN, (void*) sampleBind ) != 0)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                         "sampleInit couldn't set plugin function\n");
        return (-1);
    }

    /* Get the plugin argument count.  These arguments are defined
     * in the plug-in directive in the configuration file.
     */
    if (slapi_pblock_get( pb, SLAPI_PLUGIN_ARGC, &argc ) != 0)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                         "sampleInit couldn't get argc\n");
        return (-1);
    }
    /* Get the plugin argument array */
    if(slapi_pblock_get( pb, SLAPI_PLUGIN_ARGV, &argv ) != 0)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                        "sampleInit couldn't get argv\n");
        return (-1);
    }

    /* Low "severity" means high importance. */
    slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                    "Hello from sample\n" );

    /*
     * Register SAMPLE_BIND as one of the supported SASL mechanisms
     * so that it shows up when the RootDSE is queried.
     */
    slapi_register_supported_saslmechanism(SAMPLE_MECH);

    return LDAP_SUCCESS;
}

/*
 * Function to get the password for the specified dn.
 */
int getEntryPassword(char *dn, char ** passwd)
{
    Slapi_PBlock *pb = NULL;
    int rc = LDAP_SUCCESS;
    int numEntries = 0;
    Slapi_Entry **entries = NULL;
    Slapi_Attr *a = NULL;
    struct berval **attr_vals = NULL;

    /*
     * Do an internal search to get the entry for the given dn
     */
    pb = slapi_search_internal(dn, /* Entry to retrieve */
                               LDAP_SCOPE_BASE,
                               /* Only get the entry asked for */
                               FILTER, /* Search filter */
                               NULL, /* No controls */
                               NULL, /* Get all attributes */
                               FALSE);
                               /* Get attribute values (names only is false) */

    if (pb == NULL)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                        "Search failed for dn = %s\n", dn);
        return (LDAP_OPERATIONS_ERROR);
    }

    /* Get the return code from the search */
    slapi_pblock_get( pb, SLAPI_PLUGIN_INTOP_RESULT, &rc);
    if (rc != LDAP_SUCCESS)
    {
        /* Search failed */
        slapi_pblock_destroy( pb );
        return (rc);
    }

    /* Get the number of entries returned from the search */
    slapi_pblock_get( pb, SLAPI_NENTRIES, &numEntries );
    if (numEntries == 0)
    {
        /* Couldn't find entry */
        slapi_free_search_results_internal( pb );
        slapi_pblock_destroy( pb );
        return (LDAP_NO_SUCH_OBJECT);
    }

    /* Get the entries */
    slapi_pblock_get( pb, SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES, &entries );

    /*
     * Since we did a base level search, there can only be one entry returned.
     * Get the value of the "userpassword" attribute from the entry.
     */
    if (slapi_entry_attr_find( entries[0], PWATTR, &a ) == 0)
    {
        /* Copy the password into the out parameter */
        slapi_attr_get_values( a, &attr_vals );
        (*passwd) = slapi_ch_strdup( attr_vals[0]->bv_val );
    }
    else
    {
        /* No userpassword attribute */
        slapi_free_search_results_internal( pb );
        slapi_pblock_destroy( pb );
        return (LDAP_INAPPROPRIATE_AUTH);
    }

    slapi_free_search_results_internal( pb );
    slapi_pblock_destroy( pb );
    return (LDAP_SUCCESS);
}

/* Function to handle a bind request */
int sampleBind(Slapi_PBlock *pb)
{
    char * mechanism = NULL;
    char * dn = NULL;
    char * passwd = NULL;
    char * connDn = NULL;
    char * aString = NULL;
    struct berval * credentials = NULL;
    int rc = LDAP_SUCCESS;

    /* Get the target DN */
    if (slapi_pblock_get( pb, SLAPI_BIND_TARGET, &dn ) != 0)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                        "sampleBind couldn't get bind target\n");
        return (NEXTPLUGIN);
    }

    /* Get the password */
    if (slapi_pblock_get( pb, SLAPI_BIND_CREDENTIALS, &credentials ) != 0)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                        "sampleBind couldn't get bind target\n");
        return (NEXTPLUGIN);
    }

    /* Get the bind mechanism */
    if (slapi_pblock_get( pb, SLAPI_BIND_SASLMECHANISM, &mechanism ) != 0)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                        "sampleBind couldn't get bind target\n");
        return (NEXTPLUGIN);
    }

    /*
```

```
 * If the requested mechanism isn't SAMPLE, then we're not going to
 * handle it.
 */
if ((mechanism == NULL) || (strcmp(mechanism, SAMPLE_MECH) != 0))
{
    return (NEXTPLUGIN);
}

rc = getEntryPassword( dn, &passwd);
if (rc != LDAP_SUCCESS)
{
    slapi_send_ldap_result( pb, rc, NULL, NULL, 0, NULL );
    return (STOP_PLUGIN_SEARCH);
}

/* Check if they gave the correct password */
if ((credentials->bv_val == NULL) || (passwd == NULL) ||
    (strcmp(credentials->bv_val, passwd) != 0))
{
    slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                    "Bind as %s failed\n", dn);
    rc = LDAP_INVALID_CREDENTIALS;
}
else
{
    /*
     * Make a copy of the DN and authentication method and set them
     * in the pblock. The server will use them for the connection.
     */
    connDn = slapi_ch_strdup(dn);
    if (connDn == NULL)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                        "Could not duplicate connection DN\n");
        slapi_send_ldap_result( pb, LDAP_NO_MEMORY, NULL, NULL, 0, NULL );
        slapi_ch_free(passwd);
        return (STOP_PLUGIN_SEARCH);
    }

    /*
     * The authentication method string will look something like
     * "SASL SAMPLE_BIND"
     */
    aString = slapi_ch_malloc(strlen(SLAPD_AUTH_SASL) +
                              strlen(SAMPLE_MECH) + 2);
    if (aString == NULL)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                        "Could not duplicate authString\n");
        slapi_ch_free(passwd);
        slapi_ch_free(connDn);
        slapi_send_ldap_result( pb, LDAP_NO_MEMORY, NULL, NULL, 0, NULL );
        return (STOP_PLUGIN_SEARCH);
    }
    sprintf(aString, "%s%s", SLAPD_AUTH_SASL, SAMPLE_MECH);

    /* Set the connection DN */
    if (slapi_pblock_set( pb, SLAPI_CONN_DN, (void *) connDn) != 0)
    {
        slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                        "Could not set SLAPI_CONN_DN\n");
        slapi_ch_free(passwd);
        slapi_ch_free(connDn);
        slapi_ch_free(aString);
        slapi_send_ldap_result( pb, LDAP_OPERATIONS_ERROR,
  NULL, NULL, 0, NULL );
        return (STOP_PLUGIN_SEARCH);
        }

        /* Set the authentication type */
        if (slapi_pblock_set( pb, SLAPI_CONN_AUTHTYPE, (void *) aString) != 0)
        {
            slapi_log_error( LDAP_MSG_LOW, SAMPLE_SUBSYSTEM,
                            "Could not set SLAPI_CONN_AUTHTYPE\n");
            slapi_ch_free(passwd);
                        slapi_ch_free(connDn);
            slapi_ch_free(aString);
            slapi_send_ldap_result( pb, LDAP_OPERATIONS_ERROR,
      NULL, NULL, 0, NULL );
            return (STOP_PLUGIN_SEARCH);
        }

        rc = LDAP_SUCCESS;
    }

    /* Send the result back to the client */
    slapi_send_ldap_result( pb, rc, NULL, NULL, 0, NULL );

/*Free the memory allocated by the plug-in */
    slapi_ch_free(passwd);

    return (STOP_PLUGIN_SEARCH);
}
```

To use the plug-in you must:

1. Compile it. Use the following makefile to compile the plug-in:

   ```
   CC = gcc
   LINK = gcc -shared
   WARNINGS = -Wall -Werror
   LDAP_HOME = /usr/ldap

   INCDIRS = -I${LDAP_HOME}/include
   LIBDIRS = -L${LDAP_HOME}/lib

   CFLAGS = -g ${WARNINGS} ${INCDIRS}
   LINK_FLAGS = ${LIBDIRS} ${LIBS}

   PLUGIN = libsample.so
   OBJECTS = sample.o

   .PHONY: clean

   all: ${PLUGIN}

   .c.o:
    $(CC) ${CFLAGS} -c -o $@ $<

   ${PLUGIN}: ${OBJECTS}
    ${LINK} -o $@ $< ${LINK_FLAGS}

   clean:
    ${RM} ${PLUGIN}
    ${RM} ${OBJECTS}
   ```

2. Add the following information to the ibmslapd.conf file using the **ldapmodify** command:

   ```
   ldapmodify -D <adminDN> -w<adminPW> -i<filename>
   ```

   where *<filename>* contains:

```
DN: cn=Directory, cn=RDBM Backends, cn=IBM Directory, cn=Schemas,
  cn=Configuration
changetype: modify
add: ibm-slapdPlugin
ibm-slapdPlugin: preoperation <path to plugin>/libsample.so sampleInit
```

3. Restart the server. If the plug-in was loaded, its initialization function writes a message to the ibmslapd.log file similar to the following:

   ```
   08/25/2003 01:28:50 PM SAMPLE: Hello from sample
   ```

4. Perform an LDAP operation like the following:

   ```
   ldapsearch -D cn=bob,o=sample -w hello -p 1234
             -b o=sample objectclass=*
   ```

   The search succeeds if the entry **cn=bob,o=sample** exists and has a user password attribute with the value **hello**. If the entry does not exist, an authentication denied error is returned.

An example of DN partitioning function

A sample DN partition program that gets the rdn "cn=ck" from the dn "cn=ck,ou=India,o=sample" regardless of what the base or suffix is, and generates a partition number based on the rdn value, in this case it is "ck"

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <slapi-plugin.h>

#ifdef __cplusplus
extern "C" {
#endif
   int MyDNInit(Slapi_PBlock *pb);
#ifdef __cplusplus
}
#endif

int get_value_from_dn_fn( Slapi_PBlock *pb );


static char * get_hash_rdn( const char * dn, const char * base )
{
   char * rdn = NULL;
   size_t rdnLen = 0;
   size_t dnLen = 0;
   size_t baseLen = 0;
   size_t startNdx = 0;
   size_t endNdx = 0;

   if ((dn == NULL) || (base == NULL))
      return NULL;

   dnLen = strlen( dn );
   baseLen = strlen( base );

   /* If the base is longer than the dn, there's no rdn */
   if (baseLen > dnLen)
      return NULL;

   /* If the dn and base are the same, there's no rdn */
   if ((dnLen == baseLen) && (strcmp( dn, base ) == 0))
      return NULL;

   /* Check if the dn is under the base */
   if ((dn[dnLen - baseLen - 1] != ',') ||
```

```
       (strcmp([&dn[dnLen - baseLen], base) != 0))
      return NULL;

   /* Find the next previous comma */
   endNdx = dnLen - baseLen - 2;
   for (startNdx = endNdx; startNdx > 0; startNdx--)
   {
      if (dn[startNdx] == ',')
      {
         startNdx++;
         break;
      }
   }

   rdnLen = endNdx - startNdx + 1;
   rdn = (char *) calloc( 1, rdnLen + 1 );
   memcpy( rdn, &dn[startNdx], rdnLen );

   return rdn;
}


/* The function takes the RDN as input and generates the Partition number. */
/* If you add an entry with RDN 'cn=wrong' then it generates wrong partition number.
   This will help to check if client utility gives
    Operation Error for wrong partition number.
*/


int ck_new_get_hash_value( const char * str, int numPartitions )
{
        char temp[100];
   // static int cnt = 0;
        char *sub_string;
        unsigned int sum = 0;
      int len, partitionNum,i=0;


        sub_string = strchr (str, '=');
        sub_string++;
        strcpy(temp , sub_string);


/* Remove the comment for code below if you want to check the Server
   behavior for wrong partition number generation at Start up.
*/

/* if ( strcasecmp ( "ibmpolicies",temp ) == 0 && cnt == 1)
        {
         return (numPartitions + 5) ;
        }   */


 if ( strcasecmp ( "WRONG",temp ) == 0 )
 {
  return ( numPartitions + 5 ) ;
 }
 else
 {
      len = strlen( temp );

    for(i = 0; i < len; str++, i++)
    {
       sum += temp[i] ;
    }

      partitionNum = (  (sum * len ) % numPartitions ) + 1 ;
```

```
       return ( partitionNum );
 }

}


// Function registered for generating Partition Number


int get_value_from_dn_fn( Slapi_PBlock *pb )
{
   int  rc = 0;
   char *dn = NULL;
   char *base = NULL;
   int  numPartitions = 0;
   char * rdn = NULL;
   int  value = 0;
   SLAPI_LDAPDN *ldapDn ;
   Slapi_ldapRDN **ret_rdn = NULL;



// Get the parameters from PBlock

   if ( (rc = slapi_pblock_get( (Slapi_PBlock *)pb, SLAPI_TARGET_DN,
        (void *)&dn ) != 0 ) || (rc = slapi_pblock_get( (Slapi_PBlock *)pb,
        SLAPI_PARTITION_BASE, (void *) &base ) != 0 ) || (rc =
        slapi_pblock_get( (Slapi_PBlock *)pb, SLAPI_NUMBER_OF_PARTITIONS,
        (void *) &numPartitions ) != 0 ) )
   {
      fprintf( stderr, "Cannot get the PBlock values!\n" );
      return -1;
   }

   if ( (dn == NULL) || (base == NULL) || (numPartitions <= 0) )
    {
        fprintf( stderr,"Wrong values set in PBlock" );
        return -1;
    }


   /* If the DN and base are the same, it hashes 1 */
    if ( strcasecmp( dn, base ) == 0 )
    {
          fprintf( stderr, "Since the Base and DN are same set the
                       SLAPI_PARTITION_NUMBER to 1\n");

       if ( (rc = slapi_pblock_set( (Slapi_PBlock *)pb,
                SLAPI_PARTITION_NUMBER, (void *)1 ) ) != 0 )
     {
          fprintf( stderr, "Was not able to set value in PBlock!\n" );
          return -1;
     }
     else
     {
      return 0;
     }
    }



 // Get the Partition number based on the leftmost rdn value


 rdn = get_hash_rdn( dn, base );
 value = ck_new_get_hash_value( rdn , numPartitions );
 fprintf(stderr,"\n\n*** Partition Value is : %d",value );

      if ( (rc = slapi_pblock_set( (Slapi_PBlock *)pb,
            SLAPI_PARTITION_NUMBER, (void *)value ) ) != 0 )
 {
       fprintf( stderr, "Failed to set value in PBlock!\n" );
       free( rdn );
       return -1;
 }


 slapi_dn_free_ldapdn(&ldapDn);

 slapi_dn_free_rdn(ret_rdn);

 free( rdn );

   return 0;
}


// My Initialization Function

int MyDNInit( Slapi_PBlock * pb )
{

   if ( slapi_pblock_set( pb, SLAPI_PLUGIN_PROXY_DN_PARTITION_FN,
               ( void * ) get_value_from_dn_fn ) != 0 )
    {

      fprintf(stderr,"Cannot register Function in PBlock \n");
      return ( -1 );
    }


   return ( 0 );
}
```

Appendix F. Deprecated plug-in APIs

Although the following APIs are still supported, their use is deprecated. Use of the newer replacement APIs is strongly encouraged.

- slapi_dn_normalize. See "slapi_dn_normalize_v3()" on page 39.
- slapi_dn_normalize_case. See "slapi_dn_normalize_case_v3()" on page 39.
- slapi_dn_ignore_case. See "slapi_dn_ignore_case_v3()" on page 40.

Appendix G. Notices

This information was developed for products and services offered in the U.S.A. IBM might 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 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
U.S.A.

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 information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information 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
Department MU5A46
11301 Burnet Road
Austin, TX 78758
U.S.A.

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 document 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
Database 2
DB2
IBM
iSeries
OMEGAMON
pSeries
Tivoli

VisualAge
xSeries
zSeries

Adobe, the Adobe logo, PostScript®, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

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

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

Microsoft®, Windows, Windows NT, and the Windows logo 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, and service names may be trademarks or service marks of others.

Index

Printed in USA

GC32-1565-00