IBM Directory Server

Server Plug-ins Reference

Version 5.1

IBM

IBM Directory Server

Server Plug-ins Reference

Version 5.1

> **Note**
>
> Before using this information and the product it supports, read the general information under Appendix E, "Notices" on page 47.

Contents

Preface

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

Database plug-ins can be used to integrate your own database as a back-end to the server. For example, you can write a plug-in that integrates a relational database with the Directory Server.

Chapter 1. Introduction to server plug-ins

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. After 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.

This release of the IBM Directory Server supports the following types of server plug-ins:

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

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 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 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.

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 must 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 a 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++ 5.0.x

For Linux platforms:
EGCS 2.91.66

For Solaris platforms:
IBM CSet++ 1.1 WorkShop Pro 6 update 1

For HP platforms:
aCC A.03.25

To write your own plug-in:

1. Start by writing your functions. Include the slapi-plugin.h file (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. Input parameters to 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 13 for more information.

   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 default 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.

output parameter

You might want to update parameters in the pblock that was passed to your function. See Appendix A, "Supported database functions" on page 13 for possible output parameters.

3. Call slapi APIs in the libslapi library file. See Appendix B, "Supported iPlanet APIs" on page 17 for information about the APIs supported in this release.

4. Write an initialization function for your plug-in to call the slapi_pblock_set API in order to register your plug-in functions.

5. 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 initialization is automatic when you create the shared library.

6. Write your server plug-in functions. Set the include path to include slapi-plugin.h, and to link to libslapi.lib. Compile and link your server plug-in object files with whatever libraries you need, and libslapi library file.

7. Add a plug-in directive in the server configuration file. The syntax of the plug-in directive is:

```
keyword plugin-type  plugin-path init-func args ...
```

8. On a Windows NT operating system, in the ibmslapd.conf file, the plug-in directive is as follows:

```
dn: Directory, cn=RDBM Backends, cn=IBM SecureWay, cn=Schemas, cn=Configuration
ibm-slapdPlugin: database /lib/libldap-2dbm.dll rdbm -backend -init
```

Note: For the AIX platforms, the **.dll** extension is replaced with an **.a** extension.

For the Linux, Solaris and HP platforms, the **.dll** extension is replaced with one of the following extensions:

- **.a**
- **.so**
- **.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 that 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 the ibmslapd.conf file, there is an entry that identifies the rdbm back-end. 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 SecureWay,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 from the directory.

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 in the directory.

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_CLOSE_FN
> An LDAP back-end close routine.

Note: Stand-alone, user-supplied server back-end plug-ins are not supported except when they are 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, during, 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 input parameters from the parameter block passed in and communicate back from the server front-end with the output parameters. See the following sections for information about input and 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 sends 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 a 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 the ibmslapd.conf file can be obtained by getting the SLAPI_PLUGIN_ARGV parameter from the pblock passed in.

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

```
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 installation directory; /tmp is changed to *ldap_installation_directory*\tmp, but C:\tmp is unchanged. This indicates that the function myExtendedOpInit, which can be found in the /path/myextop.dll shared library, is executed when the server starts. The myExtendedOp that is registered in the initialization is used to handle extended-operation plug-in function. This function handles extended operations with the Object Identifier (OID) 123.456.789.

Note: For the UNIX platforms the **.dll** extension is replaced with the **.a** extension. Remember that plug-in directives are per-database.

Audit plug-ins

Administrators on some platforms 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, is sent by the server to the audit plug-in when such events occurs.

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, which defines the audit plug-in interface and data structures, is shipped with the IBM 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 Directory Server Web Administration Tool.

Configuration options

The Audit Service has the following configuration options:

ibm-auditLog
> Specifies the path name of the audit log. The default is /var/ldap/audit for UNIX platforms and *LDAP install directory*\var\audit for Windows platforms.

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.

These options are stored in the LDAP directory to allow dynamic configuration. A directory entry, cn=audit, cn=localhost, is created to contain these options. The access to the values of these options are controlled through the access control list (ACL) model. By default, the LDAP administrator is the owner of this cn=audit entry. However, with the current ACL functionality, an auditor role can be created so that only the auditor can change the option values and location of the audit log.

Note: For each modification of these option values, a message is logged in the ibmslapd 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:

```
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=ibm_us,c=us
        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=ibm_us,c=us
        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=ibm_us,c=us


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=ibm_us,c=us
        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=ibm_us,c=us
   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
```

Appendix A. Supported database functions

The following sections show input and output parameters for the default database functions.

Input parameters

This section shows the input parameters of the default 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. See the LDAPMod structure in the ldap.h file to see how to traverse the link list using a 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
```

Note: the SLAPI_PLUGIN_PRIVATE parameter is deprecated. Do not use it.

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. 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* );
```

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.

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.

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_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: This API is deprecated. See "slapi_dn_normalize_v3()" on page 22.

slapi_dn_normalize() converts a 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.

Note: A variable passed in as the DN argument is also converted in-place.

slapi_dn_normalize_case()

Note: This API is deprecated. See "slapi_dn_normalize_case_v3()" on page 23.

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.

Note: A variable passed in as the DN argument is also converted in-place.

slapi_dn_ignore_case()

Note: This API is deprecated. See "slapi_dn_ignore_case_v3()" on page 24.

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.

Note: A variable passed in as the DN argument is also converted in-place.

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

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 caller's responsibility 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 uppercase in all cases. The case of the attribute values is converted to uppercase 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=ibm,  c=us the function returns
  CN=A \+ B,O=IBM,C=US
```

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

- 1 if the specified DN is the same as the specified suffix.
- 0 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:

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_entry2str() 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 have finished 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 whether 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. This is a pointer to the actual DN in the entry, 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 must 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 have finished 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 have finished 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   The attribute for which you want to get the flags.

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 into a filter of the Slapi_Filter type. When you have finished 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, and 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, recursively frees all filters that comprise this filter. If set to 0, frees only 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. 1 is returned if the connection is through SSL; 0 is returned if it is not through SSL.

Returns

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

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:

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

Returns

Call slapi_free_search_results_internal() and slapi_pblock_destroy() 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.

        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 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 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 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   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 OIDs that specify LDAP operations that support each control.

Returns
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 to determine whether the control is supported by its server or its plug-ins.

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).

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

Returns

- 1 is returned if the specified control is present in the list of controls.
- 0 is returned 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 you specify is displayed 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

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

Appendix C. Plug-in examples

The following is an example of a pre-operation plug-in for bind in C.

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

#define NEXTPLUGIN 0
#define STOP_PLUGIN_SEARCH 1
#define PWATTR "userpassword"
#define CRAMMD5_MECH "CRAM-MD5"

int DSPBind(Slapi_PBlock *pb);

// This function registers function DSPBind as a pre-operation plug-in for
// bind operations
int DSPInit(Slapi_PBlock *pb) {
        int argc;
        char ** argv;

        if ( slapi_pblock_set( pb, SLAPI_PLUGIN_PRE_BIND_FN,
                (void *)DSPBind) != 0 ||
                slapi_pblock_get( pb, SLAPI_PLUGIN_ARGC, &argc ) != 0 ||
                slapi_pblock_get( pb, SLAPI_PLUGIN_ARGV, &argv ) != 0 ) {
                printf("DSPInit couldn't get init args\n");
                return(-1);
        }
        // call the following function so that CRAM-MD5 will show up
        // as one of the supported saslmechanisms when rootdse is queried.
        slapi_register_supported_saslmechanism(CRAMMD5_MECH);
        return 0;
};
//Use internal search function to search backend to get password for the specified dn
int
getSecret(char *dn, char **secret) {
        Slapi_PBlock *pb;
        int rc, numEntries;
        Slapi_Entry **entries;
        Slapi_Attr *a;
        struct berval **attr_vals;

        if((pb = slapi_search_internal(dn, LDAP_SCOPE_BASE,FILTER,NULL,
                NULL, 0)) == NULL) {
                printf("Search Internal failed for dn = %s\n",dn);
                return(LDAP_OPERATIONS_ERROR);
        }
        // get the return code from the above internal search function
        slapi_pblock_get(pb, SLAPI_PLUGIN_INTOP_RESULT, &rc);
        if(rc == LDAP_SUCCESS) {
                // find out how many entries the above internal search function returned
                slapi_pblock_get(pb,SLAPI_NENTRIES,&numEntries);
                if(numEntries == 0)
                   rc=LDAP_NO_SUCH_OBJECT;
                else
                  slapi_pblock_get(pb,SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES, &entries);

                //the above internal search function should return 1 entry.
                //from the entry, get the value of the attribute "userpassword"
                if(slapi_entry_attr_find(entries[0],PWATTR,&a) != NULL)
                   rc=LDAP_INAPPROPRIATE_AUTH;
                else
                   slapi_attr_get_values(a,&attr_vals);

                //make a copy of the password and return
```

```
                        if(rc == LDAP_SUCCESS) {
                                if((*secret=slapi_ch_strdup(attr_vals[0]->bv_val)) == NULL)
                                        rc=LDAP_OPERATIONS_ERROR;
                        }
                }
                slapi_free_search_results_internal(pb);
                slapi_pblock_destroy(pb);
                return(rc);
        }

        //actual pre-operation code for bind operations. This function handles CRAM-MD5
        //bind requests.
        int DSPBind(Slapi_PBlock *pb) {
                char *mechanism;
                char *chal=NULL;
                char *dn;
                char *aString=NULL;
                char *secret;
                char *connDn;
                int rc;
                struct berval   *credentials;
                struct berval    returnCreds;

                //get the input parameters from the pblock
                if (slapi_pblock_get( pb, SLAPI_BIND_TARGET, &dn) != 0 ||
                        slapi_pblock_get( pb, SLAPI_BIND_CREDENTIALS, &credentials ) != 0 ||
                        slapi_pblock_get( pb, SLAPI_BIND_SASLMECHANISM,&mechanism ) != 0 ) {
                        printf( "Could not get parameters for bind operation\n" );
                        return(NEXTPLUGIN);
                }
                //this function supports CRAM-MD5 mechanism. If the mechanism requested is
                //not CRAM-MD5, execute the next pre-bind plug-in or the default database
                //    bind operation
                if ((mechanism == NULL) || (stricmp(mechanism, CRAMMD5_MECH) != 0 )) {
                        return(NEXTPLUGIN);
                }

                //CRAM-MD5 related code here

                // calling getSecret to get the user's password
                        if((rc=getSecret(dn,&secret)) == LDAP_SUCCESS) {

                        //CRAM-MD5 related code here

                        // make a copy of the DN and the authentication method and
                        // set them to the pblock. The server will use them for the connection
                            if((connDn=slapi_ch_strdup(dn)) == NULL) {
                                    printf("Could not duplicate connection DN");
                                    rc=LDAP_NO_MEMORY;
                            } else if((aString=slapi_ch_malloc(strlen(SLAPD_AUTH_SASL)
                                                    + strlen(CRAMMD5_MECH) + 2)) == NULL) {
                                        printf("Could not duplicate connection authString");
                                        rc=LDAP_NO_MEMORY;
                            } else {
                                    sprintf(aString,"%s%s",SLAPD_AUTH_SASL,CRAMMD5_MECH);
                                    if ((slapi_pblock_set( pb, SLAPI_CONN_DN, (void *)connDn)
                                        != NULL) ||
                                      (slapi_pblock_set( pb, SLAPI_CONN_AUTHTYPE,(void *)aString)
                != NULL )) {
                                            printf("Could not set CONN_DN, CONN_AUTHTYPE");
                                            rc=LDAP_OPERATIONS_ERROR;
                                    }
                            }
                        }
                //CRAM-MD5 related code here

                slapi_send_ldap_result(pb,rc,NULL,NULL,NULL,NULL);
```

```
            if (connDn != NULL)
                slapi_ch_free(connDn);
            if (aString != NULL)
                slapi_ch_free(aString);
            return(STOP_PLUGIN_SEARCH);
}
```

Appendix D. 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 22.
- slapi_dn_normalize_case. See "slapi_dn_normalize_case_v3()" on page 23.
- slapi_dn_ignore_case. See "slapi_dn_ignore_case_v3()" on page 24.

Appendix E. 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 LZKS
11400 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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

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
VisualAge

Windows and Windows NT are registered trademarks of Microsoft® Corporation.

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

IBM ®

Printed in U.S.A.