z/OS

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

Version 2 Release 1

SA76-0169-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 173.

This edition applies to version 2, release 1, modification 0 of IBM z/OS (product number 5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Tables

About this document

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

This document contains reference information about using and writing plug-ins, which extend the capabilities of the IBM Tivoli Directory Server for z/OS (5650-ZOS).

Intended audience

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

Conventions used in this document

This document uses the following typographic conventions:

Bold **Bold** words or characters represent API names, functions, routines, utility names, and system elements that you must enter into the system literally, such as commands and options.

Italic *Italic* words or characters represent variables for which you must supply values.

`Example font`
Path names, attributes, environment variables, parameter values, examples, and information displayed by the system appear in `constant width type style`.

[ ] Brackets enclose optional items in format and syntax descriptions.

{ } Braces enclose a list from which you must choose an item in format and syntax descriptions.

| A vertical bar separates items in a list of choices.

... Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.

**** A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character \ as the last non-blank character on the line to be continued, and continue the command on the next line.

Where to find more information

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS library, including the z/OS Information Center, see z/OS Internet Library (http://www.ibm.com/systems/z/os/zos/bkserv/).

Internet sources

The following resources are available through the internet to provide additional information about the z/OS library and other security-related topics:

- **Online library**

  To view and print online versions of the z/OS publications, use this address:

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

- **Redbooks®**

  The documents known as IBM Redbooks that are produced by the International Technical Support Organization (ITSO) are available at the following address:

  `http://www.redbooks.ibm.com`

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  z/OS V2R1.0 IBM Tivoli Directory Server Plug-in Reference for z/OS
  SA76-0169-00
- The topic and page number that is related to your comment.
- The text of your comment.

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

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS support page (http://www.ibm.com/systems/z/support/).

z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

Part 1. Writing your own plug-in

Chapter 1. Introduction to server plug-ins

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

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

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

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

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

The following types of plug-ins are supported by the IBM Tivoli Directory Server for z/OS: (See Chapter 3, “Operation plug-ins,” on page 7 for more information.)

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

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

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

Chapter 2. Building an LDAP server plug-in

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

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

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

The plug-in must be stored as a member of a PDS or PDSE (a 64-bit plug-in DLL must be stored in a PDSE). The plug-in data set must be in the load list for the LDAP server, either through a STEPLIB statement or the system LNKLST. The PDS or PDSE must be APF-authorized. If Program Control is activated, the PDS or PDSE must be controlled (trusted).

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

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

For example:

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

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

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

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

  The plug-in initialization routine must register the following that are supported by the plug-in:

  – service functions
  – message types
  – distinguished name suffixes
  – extended operation object identifiers

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

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

Note: For this example, the name `plugin_init` would be the initialization routine name that is used with the **plugin** configuration option.

- When writing the SLAPI service functions that implement the plug-in design, see Chapter 4, "Plug-in application service routines," on page 11 for application service routines to use and for defined prototypes. You can also see **slapi-plugin.h** for defined prototypes.
- Decide on any input parameters for the plug-in.

  Plug-in input parameters can be retrieved by using the SLAPI_PLUGIN_ARGC or SLAPI_PLUGIN_ARGV parameters with the **slapi_pblock_get()** service routine.
- Include **slapi-plugin.h**, which contains defined SLAPI data structures and prototypes.
- Export the plug-in initialization routine.
- Compile the plug-in code into object files.
- Link the plug-in object files with one of the LDAP server SLAPI side files that are listed above.
- Ensure that the plug-in DLL module is in the load list of the LDAP server and is a member of either a PDS or PDSE.
- APF authorize the data set that contains the plug-in DLL. If Program Control is active, mark the data set as controlled (trusted).
- Edit and add the **plugin** configuration option to the LDAP server configuration file. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for more information about the configuration option.
- Restart the LDAP server.

You might want to program trace statements to follow processing flow in the plug-in. The trace macro, **SLAPI_TRACE()**, is provided in **slapi-plugin.h** to assist in tracing. This macro uses the **slapi_trace()** service routine, described in Chapter 4, "Plug-in application service routines," on page 11. For example:

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

If issuing messages from a message catalog, the message catalog name must be registered in the plug-in by calling the **slapi_pblock_set()** routine with the **SLAPI_PLUGIN_MSG_CAT_NP** parameter. When the message catalog is registered, a specific message can be retrieved by calling the **slapi_get_message_np()** routine. The retrieved message can then be issued by calling the **slapi_log_error()** routine. The **NLSPATH** and **LANG** environment variables must be properly set for the plug-in and the LDAP server to find the registered message catalog file. Also, the plugin and the LDAP server must have read access to the message catalog to issue messages from the registered message catalog. See "slapi_pblock_set()" on page 84, "slapi_get_message_np()" on page 66, and "slapi_log_error()" on page 68 for more information.

A sample plug-in showing several examples of using SLAPI service routines and a makefile are provided in Appendix A, "Plug-in sample," on page 167:

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

Chapter 3. Operation plug-ins

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

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

Pre-operation plug-ins

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

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

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

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

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

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

If the client request is a paged search request, pre-operation plug-ins are only executed before the initial paged search request.

Post-operation plug-ins

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

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

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

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

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

If the client request is a paged search request, post-operation plug-ins are only executed after the last page of a paged search request is returned.

Client-operation plug-ins

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

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

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

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

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

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

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

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

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

ABANDON

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

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

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

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

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

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

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

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

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

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

Search entries are returned by calling the **slapi_send_ldap_search_entry()** routine, search referrals are returned by calling the **slapi_send_ldap_referral()** routine, and the search result is returned by calling the **slapi_send_ldap_result()** routine.

If the client request is a paged search request, the client-operation plug-in is only called during the initial paged search request.

UNBIND

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

Chapter 4. Plug-in application service routines

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

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

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

slapi_add_internal()

Purpose

Issue an ADD entry request.

Format

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

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

Parameters

Input

dn The distinguished name of the new entry.

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

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

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

Usage

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

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

Related topics

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

EINVAL
A parameter is not valid

EIO Unable to process the ADD request

ENOMEM
Insufficient storage is available

slapi_attr_get_normalized_values()

Purpose

Obtain the normalized attribute values.

Format

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

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

Parameters

Input

attr
The directory entry attribute.

Output

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

Usage

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

Related topics

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

EINVAL
A parameter is not valid

slapi_attr_get_numvalues()

Purpose

Obtain the number of attribute values.

Format

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

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

Parameters

Input

attr
The directory entry attribute.

Output

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

Usage

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

Related topics

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

EINVAL
A parameter is not valid

slapi_attr_get_type()

Purpose

Obtain the attribute type.

Format

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

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

Parameters

Input

attr
The directory entry attribute.

Output

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

Usage

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

Related topics

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

EINVAL
A parameter is not valid

ENOENT
Attribute type is not set

slapi_attr_get_values()

Purpose

Obtain the attribute values.

Format

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

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

Parameters

Input

attr
The directory entry attribute.

Output

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

Usage

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

Related topics

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

EINVAL
A parameter is not valid

slapi_attr_value_cmp()

Purpose

Compare two attribute values.

Format

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

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

Parameters

Input

attr
The directory entry attribute.

values1
The first attribute value.

values2
The second attribute value.

Usage

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

Related topics

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

EILSEQ
Unable to normalize attribute value

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

ESRCH
Attribute type is not defined in LDAP schema

slapi_ch_calloc()

Purpose

Allocate storage for an array.

Format

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

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

Parameters

Input

elemCount
The number of elements in the array.

elemSize
The size of each element in the array.

Usage

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

Related topics

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

slapi_ch_free()

Purpose

Release allocated storage.

Format

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

void slapi_ch_free (
    void *         ptr)
```

Parameters

Input

ptr
The address of the storage is released.

Usage

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

Related topics

There is no function return value.

slapi_ch_free_values()

Purpose

Release an array of values.

Format

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

void slapi_ch_free_values (
    BerVal **          values)
```

Parameters

Input

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

Usage

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

Related topics

There is no function return value.

slapi_ch_malloc()

Purpose

Allocate storage.

Format

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

void * slapi_ch_malloc (
  unsigned long       size)
```

Parameters

Input

size
The number of bytes is allocated.

Usage

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

Related topics

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

slapi_ch_realloc()

Purpose

Reallocate storage.

Format

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

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

Parameters

Input

block
The address of block is reallocated.

newSize
The new size for the block.

Usage

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

Related topics

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

slapi_ch_strdup()

Purpose

Duplicate a character string.

Format

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

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

Parameters

Input

string
The string is duplicated.

Usage

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

Related topics

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

slapi_compare_internal()

Purpose

Issue a COMPARE request.

Format

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

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

Parameters

Input

dn The distinguished name of the entry.

type
The attribute name.

value
The attribute value.

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

Usage

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

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

Related topics

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

EINVAL
A parameter is not valid

EIO Unable to process the COMPARE request

ENOMEM
Insufficient storage is available

slapi_control_present()

Purpose

Determine if a server control is present.

Format

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

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

Parameters

Input

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

oid
The object identifier of the control you want.

Output

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

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

Usage

The **slapi_control_present()** routine searches an array of server controls for a control with the specified object identifier. If the control is found, a pointer to the control value is returned along with an indication of whether the control is marked as critical.

Related topics

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

EINVAL
A parameter is not valid.

ENOMEM
Insufficient storage is available.

slapi_delete_internal()

Purpose

Issue a DELETE request.

Format

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

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

Parameters

Input

dn The distinguished name of the entry.

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

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

Usage

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

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

Related topics

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

EINVAL
A parameter is not valid

EIO Unable to process the DELETE request

ENOMEM
Insufficient storage is available

slapi_dn_ignore_case_v3()

Purpose

Normalize a distinguished name and convert to lowercase.

Format

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

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

Parameters

Input

dn The distinguished name to be normalized.

Usage

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

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

is converted to

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

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

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

is converted to

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

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

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

is converted to

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

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

Related topics

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

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

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

slapi_dn_isparent()

Purpose

Determines whether a particular DN is the parent of another specified DN. Before calling this function, call **slapi_dn_ignore_case_v3** to normalize the DNs, which also converts all characters to lowercase.

Format

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

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

Parameters

Input

parentdn
Determine if this DN is the parent of childdn.

childdn
Determine if this DN is the child of parentdn.

Usage

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

Related topics

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

slapi_dn_normalize_v3()

Purpose

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

Format

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

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

Parameters

Input

dn The distinguished name to be normalized.

Usage

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

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

is converted to

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

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

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

is converted to

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

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

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

is converted to

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

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

Related topics

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

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

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

slapi_dn_normalize_case_v3()

Purpose

Normalize a distinguished name and convert not case-sensitive attribute values to uppercase.

Format

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

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

Parameters

Input

dn The distinguished name to be converted.

Usage

The **slapi_dn_normalize_case_v3()** routine:

- Converts a distinguished name (DN) to a canonical form by removing leading and trailing spaces, spaces between name components and spaces around the equals signs
- Normalizes the attribute type name to the uppercased primary attribute type name in the LDAP schema definition
- Any semicolons that are used to separate relative distinguished names (RDN) are converted to commas
- A compound RDN is sorted alphabetically by the primary attribute type names
- An attribute value is converted to uppercase if the associated matching rule is not case-sensitive, otherwise the case of the attribute value is preserved
- Special characters within a DN are represented by using the backslash (\) escape character

For example,

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

is converted to

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

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

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

is converted to

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

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

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

is converted to

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

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

Related topics

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

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

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

slapi_entry_add_value()

Purpose

Add an attribute value to a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

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

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

Usage

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

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

Related topics

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

EEXIST
The attribute value already exists

EILSEQ
Unable to normalize attribute value

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

ESRCH
Attribute type is not defined in LDAP schema

slapi_entry_add_values()

Purpose

Add an attribute value to a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

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

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

Usage

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

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

Related topics

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

EEXIST
The attribute value already exists

EILSEQ
Unable to normalize attribute value

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

ESRCH
Attribute type is not defined in LDAP schema

slapi_entry_alloc()

Purpose

Allocate a new directory entry.

Format

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

Slapi_Entry * slapi_entry_alloc ( void )
```

Parameters

None.

Usage

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

Related topics

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

ENOMEM
Insufficient storage is available

slapi_entry_attr_delete()

Purpose

Delete a directory entry attribute.

Format

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

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

Parameters

Input

entry
The directory entry.

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

Usage

The **slapi_entry_attr_delete()** routine deletes an attribute from a directory entry. A not case-sensitive compare is used when searching for the attribute type.

Related topics

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

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

slapi_entry_attr_find()

Purpose

Find a directory entry attribute.

Format

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

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

Parameters

Input

entry
The directory entry.

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

Output

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

Usage

The **slapi_entry_attr_find()** routine searches the directory entry for the specified attribute and returns the address of the attribute if it is found. A not case-sensitive compare is used when searching for the attribute type. The attribute name in the returned attribute is the primary attribute name in lowercase, as defined in the LDAP schema.

Related topics

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

EINVAL
A parameter is not valid

ENOENT
Attribute not found

ESRCH
Attribute type is not defined in LDAP schema

slapi_entry_delete_value()

Purpose

Remove an attribute value from a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

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

value
The attribute value to be deleted.

Usage

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

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

Related topics

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

EILSEQ
Unable to normalize attribute value

EINVAL
A parameter is not valid

ENOENT
The attribute value was not found

ESRCH
Attribute type is not defined in LDAP schema

slapi_entry_delete_values()

Purpose

Remove multiple attribute values from a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

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

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

Usage

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

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

Related topics

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

EILSEQ
Unable to normalize attribute value

EINVAL
A parameter is not valid

ENOENT
The attribute value was not found

ESRCH
Attribute type is not defined in LDAP schema

slapi_entry_dup()

Purpose

Duplicate a directory entry.

Format

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

Slapi_Entry * slapi_entry_dup (
   Slapi_Entry *        entry)
```

Parameters

Input

entry
The directory entry to be duplicated.

Usage

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

Related topics

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

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

slapi_entry_first_attr()

Purpose

Obtain the first attribute in a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

Output

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

Usage

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

Related topics

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

EINVAL
A parameter is not valid

ENOENT
The entry has no attributes

slapi_entry_free()

Purpose

Free a directory entry.

Format

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

void slapi_entry_free (
   Slapi_Entry *       entry)
```

Parameters

Input

entry
The directory entry to be freed.

Usage

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

Related topics

There is no function return value.

slapi_entry_get_dn()

Purpose

Obtain the directory entry name.

Format

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

char * slapi_entry_get_dn (
  Slapi_Entry *       entry)
```

Parameters

Input

entry
The directory entry.

Usage

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

Related topics

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

EINVAL
A parameter is not valid

ENOENT
Attribute type is not set

slapi_entry_merge_value()

Purpose

Add an attribute value to a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

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

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

Usage

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

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

Related topics

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

EILSEQ
Unable to normalize attribute value

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

ESRCH
Attribute type is not defined in LDAP schema

slapi_entry_merge_values()

Purpose

Add multiple attribute values to a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

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

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

Usage

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

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

Related topics

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

EILSEQ
Unable to normalize attribute value

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

ESRCH
Attribute type is not defined in LDAP schema

slapi_entry_next_attr()

Purpose

obtain the next attribute in a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

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

Output

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

Usage

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

Related topics

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

EINVAL
A parameter is not valid

ENOENT
There are no more attributes

slapi_entry_replace_value()

Purpose

Replace the attribute values in a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

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

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

Usage

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

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

Related topics

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

EILSEQ
Unable to normalize attribute value

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

ESRCH
Attribute type is not defined in LDAP schema

slapi_entry_replace_values()

Purpose

Replace the attribute values in a directory entry.

Format

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

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

Parameters

Input

entry
The directory entry.

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

value
A NULL-terminated array of replacement values.

Usage

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

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

Related topics

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

EEXIST
Duplicate value in replacement values

EILSEQ
Unable to normalize attribute value

ENOMEM
Insufficient storage is available

ESRCH
Attribute type is not defined in LDAP schema

slapi_entry_schema_check()

Purpose

Check a directory entry against the LDAP schema.

Format

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

int slapi_entry_schema_check (
  Slapi_Entry *         entry)
```

Parameters

Input

entry
The directory entry.

Usage

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

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

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

Related topics

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

EDOM
Obsolete attribute type or object class

EEXIST
Single-valued attribute has multiple values

EILSEQ
An auxiliary object class is a base object class or there is not only one structural object class

EINVAL
A parameter is not valid

ENOENT
Required attribute not found

ENOMEM
Insufficient storage is available

EPERM
Attribute cannot be modified by user

ERANGE
Attribute not allowed by object class

ESRCH
Undefined attribute type or object class

slapi_entry_set_dn()

Purpose

Set the directory entry name.

Format

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

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

Parameters

Input

entry
The directory entry.

dn The distinguished name for the entry.

Usage

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

Related topics

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

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

slapi_filter_get_attribute_type()

Purpose

Obtain the search filter attribute type.

Format

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

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

Parameters

Input

filter
The search filter.

Output

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

Usage

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

- LDAP_FILTER_APPROX
- LDAP_FILTER_EQUALITY
- LDAP_FILTER_GE
- LDAP_FILTER_LE
- LDAP_FILTER_PRESENT
- LDAP_FILTER_SUBSTRINGS

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

Related topics

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

EINVAL
A parameter is not valid

EPERM
The filter does not have an attribute type

ESRCH
Attribute type is not defined in LDAP schema

slapi_filter_get_ava()

Purpose

Obtain the search filter assertion value.

Format

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

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

Parameters

Input

filter
The search filter.

Output

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

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

Usage

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

- LDAP_FILTER_APPROX
- LDAP_FILTER_EQUALITY
- LDAP_FILTER_GE
- LDAP_FILTER_LE
- LDAP_FILTER_PRESENT

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

Related topics

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

EILSEQ
Assertion value is not normalized

EINVAL
A parameter is not valid

EPERM
The filter does not have an attribute type

ESRCH
Attribute type is not defined in LDAP schema

slapi_filter_get_choice()

Purpose

Obtain the search filter type.

Format

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

int slapi_filter_get_choice (
  Slapi_Filter *        filter)
```

Parameters

Input

filter
The search filter.

Usage

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

The following search filter types are defined:

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

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

Related topics

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

EINVAL
A parameter is not valid

slapi_filter_get_subfilt()

Purpose

Obtain the search filter substrings.

Format

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

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

Parameters

Input

filter
The search filter.

Output

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

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

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

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

Usage

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

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

Related topics

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

EILSEQ
Unable to normalize attribute value

EINVAL
A parameter is not valid

EPERM
The filter does not have substrings

ESRCH
Attribute type is not defined in LDAP schema

slapi_filter_list_first()

Purpose

Obtain the first subfilter.

Format

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

Slapi_Filter * slapi_filter_list_first (
  Slapi_Filter *      filter)
```

Parameters

Input

filter
The search filter.

Usage

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

Related topics

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

EINVAL
A parameter is not valid

ENOENT
There are no subfilters

EPERM
The filter is not an AND, OR, or NOT filter

slapi_filter_list_next()

Purpose

Obtain the next subfilter.

Format

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

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

Parameters

Input

filter
The search filter.

subfilter
The current subfilter.

Usage

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

Related topics

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

EINVAL
A parameter is not valid

ENOENT
There are no subfilters

EPERM
The filter is not an AND, OR, or NOT filter

slapi_get_message_np()

Purpose

Retrieves a message from a registered plug-in message catalog file.

Format

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

char * slapi_get_message_np (
    Slapi_PBlock *      pb,
    int                 msgNumber,
    ...)
```

Parameters

Input

pb The plug-in parameter block.

msgNumber
The message identifier in the registered plug-in message catalog file.

...
A variable argument list containing the message substitutions. See the *fmt* parameter of "slapi_log_error()" on page 68 for supported printf-style substitution codes.

Usage

The **slapi_get_message_np()** routine retrieves a message from a registered plug-in message catalog file, and creates a message string with the supplied message substitutions. A message catalog is registered in the plug-in by calling the **slapi_pblock_set()** routine with the **SLAPI_PLUGIN_MSG_CAT_NP** parameter. The NLSPATH and LANG environment variables must be properly set for the plug-in and the LDAP server to find the registered message catalog file. Also, the plug-in and the LDAP server must have read access to the message catalog in order to issue messages from the registered message catalog. If a message is successfully retrieved, it can be written to the LDAP server job log by calling the **slapi_log_error()** routine.

Related topics

If the registered plug-in message catalog cannot be opened or the message cannot be found in the catalog file, a NULL is returned. If the message is found, it must not be freed by the caller.

slapi_isSDBM_authenticated()

Purpose

Determines whether the client BIND DN is contained in the SDBM backend.

Format

```
#include <slapi_plugin.h>

int   slapi_isSDBM_authenticated (
  Slapi_PBlock *  pb )
```

Parameters

Input

pb The plug-in parameter block.

Usage

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

Related topics

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

slapi_log_error()

Purpose

Write a message to the LDAP server job log.

Format

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

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

Parameters

Input

msg_severity
Level of severity of the message. Level of severity is one of the following:

- LDAP_MSG_LOW
- LDAP_MSG_MED
- LDAP_MSG_HIGH

To force the message to the console logically on LDAP_OP_CONSOLE with the *msg_severity*, see "Usage" on page 69 for when messages are written to the log.

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

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

Table 2. printf()-style substitution codes. printf()-style substitution codes

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

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

Usage

1. The **slapi_log_error()** routine formats a message and writes it to the job log.
2. The message is written to the operator console depending on the setting of the environment variable LDAP_CONSOLE_LEVEL unless LDAP_OP_CONSOLE is logically ORed with the *msg_severity*. In this case, it is always written to the operator console. The **slapi_log_error()** severity level equates to the LDAP_CONSOLE_LEVEL severity level as follows:
   - LDAP_MSG_LOW is an Information (I) severity level
   - LDAP_MSG_MED is an Attention (W) severity level
   - LDAP_MSG_HIGH is an Error (E) severity level

   See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*SLAPI_REQUESTOR for more information about the use of LDAP_CONSOLE_LEVEL, activity logging, and LDAP server configuration.
3. Operator console messages must include a message identifier. The subsystem input field is used for the message identifier.
4. The message is written to the LDAP server activity log when activity logging is enabled.
5. Examples (<*Italics*> are completed with the appropriate system and LDAP server information):
   - ```
     slapi_log_error(LDAP_MSG_MED,
      "GLD1004I","LDAP server is ready for requests.\n" );
     ```

     Writes the following message to the job log:

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

     where

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

     Writes the following message to the job log:

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

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

     Writes the following message to the job log:

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

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

Related topics

None.
     ```

slapi_modify_internal()

Purpose

Issue a modify request.

Format

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

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

Parameters

Input

dn The distinguished name of the entry.

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

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

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

Usage

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

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

Related topics

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

EINVAL
A parameter is not valid

EIO Unable to process the MODIFY request

ENOMEM
Insufficient storage is available

slapi_modrdn_internal()

Purpose

Issue a MODIFY-DN request.

Format

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

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

Parameters

Input

dn The distinguished name of the entry.

newrdn
The new RDN® for the entry.

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

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

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

Usage

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

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

Related topics

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

EINVAL
A parameter is not valid

EIO Unable to process the MODIFY-DN request

ENOMEM
Insufficient storage is available

slapi_op_abandoned()

Purpose

Check if the current request has been abandoned.

Format

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

int slapi_op_abandoned (
  Slapi_PBlock *        pb)
```

Parameters

Input

pb The plug-in parameter block.

Usage

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

Related topics

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

EINVAL
A parameter is not valid

EPERM
There is no client request

slapi_pblock_destroy()

Purpose

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

Format

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

void slapi_pblock_destroy (
  Slapi_PBlock *         pb)
```

Parameters

Input

pb The plug-in parameter block.

Usage

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

Related topics

There is no function return value.

slapi_pblock_get()

Purpose

Retrieve a value from the plug-in parameter block.

Format

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

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

Parameters

Input

pb The plug-in parameter block.

arg
The parameter value to be retrieved.

Output

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

Usage

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

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

- SLAPI_PLUGIN_INTOP_REFERRALS
- SLAPI_PLUGIN_INTOP_RESULT
- SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES
- SLAPI_PLUGIN_INTOP_SEARCH_REFERRALS

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

Table 3. Operational parameters. Operational parameters

Name	Format	Usage
SLAPI_PLUGIN_ARGC	int	The number of arguments specified on the **plugin** configuration statement.
SLAPI_PLUGIN_ARGV	char **	A NULL-terminated array of arguments specified on the **plugin** configuration statement. See SLAPI_PLUGIN_ARGC for the number of arguments.

Table 3. Operational parameters (continued). Operational parameters

Name	Format	Usage
SLAPI_PLUGIN_CTLLIST	char **	An array of server control object identifiers registered by the current plug-in. The value is NULL if there are no server controls.
SLAPI_PLUGIN_DB_SUFFIXES	char **	A NULL-terminated array of database suffixes registered for the current plug-in. The value is NULL if there are no database suffixes registered. The database suffixes are normalized as determined by the LDAP server schema.
SLAPI_PLUGIN_EXT_OP_ OIDLIST	char **	A NULL-terminated array of extended operation object identifiers registered for the current plug-in. The value is NULL if there are no object identifiers registered.
SLAPI_PLUGIN_PRIVATE	void *	Private value set by the **slapi_pblock_set()** routine. Each plug-in can have its own private value and must be freed on termination.
SLAPI_PLUGIN_TYPE	int	Current plug-in type: • SLAPI_PLUGIN_PREOPERATION • SLAPI_PLUGIN_CLIENTOPERATION • SLAPI_PLUGIN_POSTOPERATION

Table 4. General request parameters. General request parameters

Name	Format	Usage
SLAPI_CONN_ID	unsigned long	Client connection identifier. Connection identifiers are reused when a connection is closed, abandoned, or an unbind occurs. The plug-in registers a SLAPI_PLUGIN_DISCONNECT_FN if it must be informed when a client connection is closed, abandoned, or an unbind occurs.
SLAPI_CONN_PRIVATE	void *	Private value for the current connection. Each plug-in can have its own set of private connection values and must be freed on termination. The value is NULL if the plug-in has not set a private value for the connection.
SLAPI_CONN_VERSION	int	The LDAP protocol version for the connection. This is the previous protocol version while processing a BIND request (use the SLAPI_BIND_VERSION parameter to obtain the protocol version specified in the BIND request)
SLAPI_REQCONTROLS	LDAPControl **	A NULL-terminated array of server controls specified in the request. The value is NULL if there are no controls.

Table 4. General request parameters (continued). General request parameters

Name	Format	Usage
SLAPI_REQUEST_ID	unsigned int	Message identifier for the current client request.
SLAPI_REQUESTOR_ACEE	void *	Address of the ACEE of the bound client. NULL is returned when no ACEE is associated with the client. An ACEE is only available if the client binds by way of SDBM, Native Authentication, or SSL with a certificate stored in RACF.
SLAPI_REQUESTOR_ALT_NAMES	char **	A NULL-terminated array of normalized alternate names for the authentication DN. The value is NULL if there are no alternate names.
SLAPI_REQUESTOR_DN	char *	Authenticated DN of the client requesting the operation. A zero-length string is returned if the client is not authenticated.
SLAPI_REQUESTOR_GROUPS	char **	A NULL-terminated array of normalized group names for the authentication DN. The value is NULL if the authentication DN is not a member of any groups or if group gathering was not enabled for the BIND request.
SLAPI_REQUESTOR_IS_ADMIN	int	The value is 1 if the requester is the LDAP administrator. Otherwise, the value is 0.
SLAPI_REQUESTOR_NORM_DN	char *	Normalized authenticated DN of the client requesting the operation. A zero-length string is returned if the client is not authenticated.
SLAPI_REQUESTOR_SAF_ID	char *	SAF user ID of the bound client. A zero-length string is returned when no SAF user ID is associated with the client. The value is uppercased and in local code page.
SLAPI_REQUESTOR_SECURITY_ LABEL	char *	The security label associated with the client requesting the operation. The security label is returned as a local code page string. A zero-length string is returned when the client is not authenticated or when LDAP server security label processing is not configured for client operations.
SLAPI_TARGET_DN	char *	Target DN specified in the current request. The value is NULL if the request does not have a target DN.

Table 5. ABANDON request parameters. ABANDON request parameters

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

Table 6. ADD request parameters. ADD request parameters

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

Table 7. BIND request parameters. BIND request parameters

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

Table 8. COMPARE request parameters. COMPARE request parameters

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

Table 9. DELETE request parameters. DELETE request parameters

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

Table 10. EXTENDED OPERATION request parameters. EXTENDED OPERATION request parameters

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

Table 11. MODIFY request parameters. MODIFY request parameters

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

Table 12. MODIFY DN request parameters. MODIFY DN request parameters

Name	Format	Usage
SLAPI_MODRDN_DELOLDRDN	int	1 if the old RDN is to be deleted, 0 if the old RDN is not to be deleted.
SLAPI_MODRDN_NEWRDN	char *	New RDN for the entry.
SLAPI_MODRDN_NEWSUPERIOR	char *	DN of the new superior entry. The value is NULL if a new superior entry is not specified in the MODIFY DN request.
SLAPI_MODRDN_TARGET	char *	New DN for the renamed entry. This is the same value returned by SLAPI_TARGET_DN.

Table 13. SEARCH request parameters. SEARCH request parameters

Name	Format	Usage
SLAPI_SEARCH_ATTRS	char **	A NULL-terminated array of attribute types from the search request. The value is NULL if there are no attribute types in the search request. The attribute names are the primary attribute names in lowercase as defined in the LDAP schema.

Table 13. SEARCH request parameters (continued). SEARCH request parameters

Name	Format	Usage
SLAPI_SEARCH_ATTRSONLY	int	1 if only attribute types are to be returned, 0 if attribute types and values are to be returned.
SLAPI_SEARCH_DEREF	int	Alias dereferencing: • LDAP_DEREF_NEVER • LDAP_DEREF_FINDING • LDAP_DEREF_SEARCHING • LDAP_DEREF_ALWAYS
SLAPI_SEARCH_FILTER	Slapi_Filter *	Search filter.
SLAPI_SEARCH_SCOPE	int	Search scope: • LDAP_SCOPE_BASE • LDAP_SCOPE_ONELEVEL • LDAP_SCOPE_SUBTREE
SLAPI_SEARCH_SIZELIMIT	int	Search size limit. This is the smaller of the size limit from the search request and the size limit specified in the LDAP server configuration file. The configured size limit should be ignored for the LDAP administrator.
SLAPI_SEARCH_TARGET	char *	DN of the base entry for the search. This is the same value returned by SLAPI_TARGET_DN.
SLAPI_SEARCH_TIMELIMIT	int	Search time limit. This is the smaller of the time limit from the search request and the time limit specified in the LDAP server configuration file. The configured time limit is ignored for the LDAP administrator.

Table 14. Callback parameters. Callback parameters

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

Table 14. Callback parameters (continued). Callback parameters

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

Table 15. General result parameters. General result parameters

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

Table 16. Internal request result parameters. Internal request result parameters

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

Table 16. Internal request result parameters (continued). Internal request result parameters

Name	Format	Usage
SLAPI_PLUGIN_INTOP_RESULT	int	Result code returned in the result message for an internal request.

Related topics

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

EFAULT
Value address is not valid

EINVAL
A parameter is not valid

ENOENT
Value does not exist

ENOMEM
Insufficient storage is available

EPERM
Insufficient storage is available

slapi_pblock_set()

Purpose

Format

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

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

Parameters

Input

pb The plug-in parameter block.

arg
The parameter value to be set.

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

Usage

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

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

Table 17. Registration parameters. Registration parameters

Name	Format	Usage
SLAPI_PLUGIN_ABANDON_FN	int (*)(Slapi_PBlock *)	Routine to process a client ABANDON request.
SLAPI_PLUGIN_ADD_FN	int (*)(Slapi_PBlock *)	Routine to process a client ADD request.
SLAPI_PLUGIN_BIND_FN	int (*)(Slapi_PBlock *)	Routine to process a client BIND request.
SLAPI_PLUGIN_CALLBACK_FN	int (*)(Slapi_PBlock *)	Routine to process a server callback request. A callback routine is registered only by a client-operation plug-in.
SLAPI_PLUGIN_CLOSE_FN	void (*)(Slapi_PBlock *)	Routine to be called during LDAP server termination.
SLAPI_PLUGIN_COMPARE_FN	int (*)(Slapi_PBlock *)	Routine to process a client COMPARE request.

Table 17. Registration parameters (continued). Registration parameters

Name	Format	Usage
SLAPI_PLUGIN_DELETE_FN	int (*)(Slapi_PBlock *)	Routine to process a client DELETE request.
SLAPI_PLUGIN_DISCONNECT_FN	void (*)(Slapi_PBlock *)	Routine to be called when an LDAP client session is closed.
SLAPI_PLUGIN_EXT_OP_FN	int (*)(Slapi_PBlock *)	Routine to process a client EXTENDED OPERATION request.
SLAPI_PLUGIN_MODIFY_FN	int (*)(Slapi_PBlock *)	Routine to process a client MODIFY request.
SLAPI_PLUGIN_MODRDN_FN	int (*)(Slapi_PBlock *)	Routine to process a client MODIFY DN request.
SLAPI_PLUGIN_SEARCH_FN	int (*)(Slapi_PBlock *)	Routine to process a client SEARCH request.
SLAPI_PLUGIN_THREAD_FN	void (*)(Slapi_PBlock *)	Routine to be called when an LDAP server worker thread terminates.
SLAPI_PLUGIN_UNBIND_FN	int (*)(Slapi_PBlock *)	Routine to process a client UNBIND request.

Table 18. Operational parameters. Operational parameters

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

Table 18. Operational parameters (continued). Operational parameters

Name	Format	Usage
SLAPI_PLUGIN_EXT_OP_OIDLIST	char **	NULL-terminated array of extended operation object identifiers for the current plug-in. This parameter is set only by a client-operation plug-in.
SLAPI_PLUGIN_MSG_CAT_NP	char **	The name of the message catalog that is to be registered by this plug-in.
SLAPI_PLUGIN_PRIVATE	void *	Private value that is retrieved by the **slapi_pblock_get()** routine. Each plug-in has its own private value.

Table 19. Callback parameters. Callback parameters

Name	Format	Usage
SLAPI_CALLBACK_ERRMSG	char *	An error message is returned to the LDAP client if an error occurred. Specify NULL if there is no error message.
SLAPI_CALLBACK_LIST	char **	A NULL-terminated array of names. This is the return value for a group list or alternate names callback. Specify NULL if there are no names.
SLAPI_CALLBACK_PASSWORD	char *	The user password. This is a return value for a password callback. Specify NULL if there is no password for the supplied name.
SLAPI_CALLBACK_STATUS	int	This is the LDAP result code for the request. It is set to LDAP_SUCCESS if the callback request was processed, LDAP_UNWILLING_TO_PERFORM if the plug-in does not recognize the callback type, or an LDAP error code if an error occurred. The return status is LDAP_SUCCESS if there is no password, group, or alternate name for the supplied name and the appropriate return value (SLAPI_CALLBACK_LIST or SLAPI_CALLBACK_PASSWORD) is set to NULL.
SLAPI_CALLBACK_TARGET_DN	char *	The entry name associated with the password returned for the SLAPI_CALLBACK_PASSWORD parameter. This is a return value for a password callback. Specify NULL if there is no entry.

Table 20. General result parameters. General result parameters

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

Table 21. EXTENDED OPERATION result parameters. EXTENDED OPERATION parameters

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

Related topics

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

EEXIST
Value already exists

EFAULT
Value address is not valid

EINVAL
A parameter is not valid

ENOMEM
Insufficient storage is available

EPERM
A parameter is not allowed

slapi_search_internal()

Purpose

Issue a SEARCH request.

Format

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

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

Parameters

Input

base
The base DN for the search.

scope
The scope for the search must be:

- LDAP_SCOPE_BASE
- LDAP_SCOPE_ONELEVEL
- LDAP_SCOPE_SUBTREE

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

controls
A NULL-terminated array of server controls for the SEARCH request. Specify NULL if there are no server controls. The **pagedResults** (OID `1.2.840.113556.1.4.319`) server control is not supported on an internal SEARCH request.

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

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

Usage

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

- `SLAPI_PLUGIN_INTOP_RESULT` - The result code from the result message
- `SLAPI_PLUGIN_INTOP_ERRMSG` - The error message from the result message

- SLAPI_PLUGIN_INTOP_MATCHED_DN - The matched DN from the result message
- SLAPI_PLUGIN_INTOP_REFERRALS - The referrals from the result message
- SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES - The search entries
- SLAPI_PLUGIN_INTOP_SEARCH_REFERRALS - The search references

Related topics

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

EINVAL
A parameter is not valid

EIO Unable to process the SEARCH request

ENOMEM
Insufficient storage is available

slapi_send_ldap_referral()

Purpose

Send an LDAP search referral message to the client

Format

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

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

Parameters

Input

pb The plug-in parameter block.

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

refs
The referral values from the directory entry.

Input/Output

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

Usage

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

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

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

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

Related topics

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

EINVAL
A parameter is not valid

EIO Unable to send the message

ENOMEM
Insufficient storage is available

EPERM
The plug-in is not a pre-operation or client-operation plug-in or the current request is not a search request

slapi_send_ldap_result()

Purpose

Send the LDAP result message to the client.

Format

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

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

Parameters

Input

pb The plug-in parameter block.

resultCode
The result code to be returned to the client.

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

- LDAP_ALIAS_DEREF_PROBLEM
- LDAP_ALIAS_PROBLEM
- LDAP_INVALID_DN_SYNTAX
- LDAP_NO_SUCH_OBJECT

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

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

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

Usage

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

the **slapi_send_ldap_result()** routine if the result message includes server controls, an extended result object identifier or an extended result value.

Related topics

There is no function return value.

slapi_send_ldap_search_entry()

Purpose

Send an LDAP search entry message to the client.

Format

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

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

Parameters

Input

pb The plug-in parameter block.

entry
The directory entry.

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

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

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

Usage

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

If the client search request specified a valid **pagedResults** (OID `1.2.840.113556.1.4.319`) or **SortKeyRequest** (OID `1.2.840.113556.1.4.473`) server control, the LDAP server performs paging or sorting of search entries after the search operation has returned from all pre-operation plug-ins, client-operation plug-ins, and LDAP server backend calls.

Related topics

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

ECANCELED
Client has canceled the request

EINVAL
A parameter is not valid

EIO Unable to send the message

ENOMEM
Insufficient storage is available

EPERM
The plug-in is not a pre-operation or client-operation plug-in or the current request is not a search request

ESRCH
Attribute type is not defined in LDAP schema

slapi_trace()

Purpose

Writes an LDAP server trace message.

Format

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

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

Parameters

Input

traceLevel

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

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

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

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

Table 22. printf()-style substitution codes. printf()-style substitution codes

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

Usage

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

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

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

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

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

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

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

Related topics

None.

Part 2. IBM TDS for z/OS provided plug-ins

Chapter 5. ICTX plug-in

The ICTX plug-in provides centralized remote resource management. This allows resource managers that do not reside on z/OS to centralize authorization decisions and security event logging by using RACF through the ICTX plug-in. These services are provided through two LDAP extended operations: **Remote authorization** and **Remote auditing**. These extended operations allow any remote application that has access to an LDAP client, the ability to query z/OS for authorization decisions and for logging security events. The **Remote authorization** extended operation uses the RACROUTE REQUEST=AUTH SAF service while the **Remote auditing** extended operation uses the r_auditx (IRRSAX00) RACF callable service. See *z/OS Security Server RACROUTE Macro Reference* for more information about the RACROUTE REQUEST=AUTH service. See *z/OS Security Server RACF Callable Services* for more information about the r_auditx (IRRSAX00) RACF callable service.

The Enterprise Identity Mapping (EIM) product provided the ICTX plug-in or backend since z/OS V1.8. The ICTX plug-in that is shipped with the z/OS LDAP server contains the following enhancements or features that are not provided in the EIM ICTX plug-in.

- Support for running the plug-in and LDAP server in 64-bit addressing mode.
- The ability to perform simple binds to the SDBM backend, LDBM or TDBM native authentication binds, SASL EXTERNAL binds where the certificate is mapped to a SAF or RACF user, and Kerberos binds.

Note: The ICTX plug-in or backend that is shipped with EIM is no longer being enhanced. Only the ICTX plug-in that is shipped with the z/OS LDAP server is updated or enhanced.

Configuring the ICTX plug-in

The z/OS LDAP server supports running the ICTX plug-in either in 31-bit or 64-bit. The plug-in can be configured manually or by using the **dsconfig** utility.

To manually configure the ICTX plug-in, the following **plugin** configuration line must be specified before any database configuration sections.

```
plugin clientOperation GLDBIC31/GLDBIC64 ICTX_INIT "CN=ICTX"
```

To configure the ICTX plug-in using the **dsconfig** utility, set the PLUGIN_ICTX input option to **on** in the **ds.profile** input file of the **dsconfig** utility.

Using remote authorization and audit

An application or resource manager that uses the **Remote authorization** or **Remote auditing** extended operations must be able to generate requests, send it through the network to the appropriate z/OS LDAP server and interpret the response from the server. The following steps represent the typical sequence of events that are specific to the **Remote authorization** or **Remote auditing** extended operations:

1. The authenticated user must resolve to a SAF or RACF identity that is allowed to perform the authorization check or remote auditing request. The following binds in the z/OS LDAP server can resolve to a SAF or RACF identity:

- Simple bind to the ICTX plug-in with using an authorized `racfid=userid,cn=ictx` bind distinguished name. If the RACF user ID begins with a number sign (#), it must be preceded by a backslash (\) escape character. Number sign characters in other positions of the user ID do not need to be escaped. For example:

  ```
  racfid=\#user#id,cn=ictx
  ```

- Simple bind to the SDBM backend. See SDBM authorization for more information.
- LDBM or TDBM native authentication bind. Native authentication allows the use of an LDBM or TDBM entry but the password or password phrase is stored in SAF. See Native authentication for more information.
- Kerberos (GSSAPI) bind. See Kerberos authentication for more information.
- SASL EXTERNAL certificate bind where the certificate is mapped to a SAF or RACF user. See Setting up for SSL/TLS for more information about mapping certificates to users.

2. The application must build a DER-encoded extended operation request having the defined ASN.1 syntax that is specific to the **Remote authorization** or **Remote auditing** extended operation request. That request can then be included with the LDAP handle and the specific request OID on the LDAP client application call, such as **ldap_extended_operation_s()**, to build the LDAP message and send it to the server.
3. The z/OS LDAP server receives the request and routes it to the ICTX plug-in, where it is decoded and processed. The ICTX plug-in verifies the correct syntax and the authority of the requester before invoking the SAF authorization check or audit service to satisfy the request. The result of the SAF service is a DER-encoded response that the LDAP server returns.
4. The application must decode the response to interpret the results. A nonzero **LdapResult** code indicates that the request was not processed by the ICTX plug-in. A nonzero **LdapResult** is accompanied by a reason code message in the response that might provide additional diagnostic information.

   Note: A zero **LdapResult** code does not necessarily imply the request was processed successfully (or for authorization, that a user has the specified access). It does, however, indicate that an extended operation `responseValue` was returned. The application should verify that the ICTX `responseCode` within the `responseValue` indicates success (0). A non-zero `responseCode` indicates one or more request items resulted in errors (or unauthorized users). The application should check the `MajorCode` within each response item to determine which returned failures. The application should be aware that ICTX might not return a response item corresponding to each request item in the event of a severe error, such as an error encountered in the extended operation encoding.

   The application can send as many requests as needed throughout a single bound session, and should unbind from the z/OS LDAP server when it finishes processing ICTX plug-in requests.

Setting up authorization for working with remote services

After the user successfully authenticated and issued the appropriate extended operation, the bound user must then have the appropriate authority to use the underlying SAF callable services.

For the **Remote authorization** extended operation, the bound user must have at least READ access to the FACILITY class profile IRR.LDAP.REMOTE.AUTH to

check the user's own access to a resource. To check access of another user, the user must have at least UPDATE access to FACILITY class profile IRR.LDAP.REMOTE.AUTH.

For example:

```
RDEFINE FACILITY IRR.LDAP.REMOTE.AUTH UACC(NONE)
PERMIT IRR.LDAP.REMOTE.AUTH CLASS(FACILITY) ID(BINDUSER) ACCESS(UPDATE))
SETROPTS RACLIST(FACILITY) REFRESH
```

For the **Remote audit** extended operation, the bound user must have at least READ access to the FACILITY class profile IRR.LDAP.REMOTE.AUDIT.

For example:

```
RDEFINE FACILITY IRR.LDAP.REMOTE.AUDIT UACC(NONE)
PERMIT IRR.LDAP.REMOTE.AUDIT CLASS(FACILITY) ID(BINDUSER) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Also, the user ID running the z/OS LDAP server must have at least READ access to the FACILITY class profile IRR.RAUDITX to issue the r_audix RACF callable service. See *z/OS Security Server RACF Callable Services* for more information about the r_auditx (IRRSAX00) RACF callable service.

```
For example:
RDEFINE FACILITY IRR.RAUDITX UACC(NONE)
PERMIT IRR.RAUDITX CLASSS(FACILITY) ID(LDAPSRV) ACCESS(READ))
SETROPTS RACLIST(FACILITY) REFRESH
```

Remote authorization extended operation

The **Remote authorization** extended operation request results in calls to the RACROUTE REQUEST=AUTH SAF service. The results of the RACROUTE REQUEST=AUTH service are returned to the caller. For more information about RACROUTE REQUEST=AUTH, see *z/OS Security Server RACROUTE Macro Reference*.

The **Remote authorization** extended operation request must contain the DER-encoding of the ASN.1 syntax. The request OID is 1.3.18.0.2.12.66. The following is the **Remote authorization** extended operation request syntax:

```
requestValue ::= SEQUENCE  {
    requestVersion              INTEGER,
    itemList                    SEQUENCE of
      item                        SEQUENCE {
        itemVersion                 INTEGER,
        itemTag                     INTEGER,
        userOrGroup                 OCTET STRING,
        resource                    OCTET STRING,
        class                       OCTET STRING,
        access                      INTEGER,
        logString                   OCTET STRING
      }
}
```

Where,

`requestValue`: The name for the entire sequence of authorization request data.

`requestVersion`: The format of the request value. Version 1 indicates a user authorization request; each individual `item` in the `itemList` is an authorization request for a RACF user ID. Version 2 indicates a user authorization or a group

authorization request; each individual `item` in the `itemList` is an authorization request for either a RACF user ID or a RACF group ID.

`itemList`: A sequence of one or more items, which allows for multiple authorization checks within a single ICTX request. The size of the entire encoded `requestValue` is limited to 16 million bytes unless your encoding routine or LDAP client imposes a stricter limit. If `requestVersion` is 2, the `itemList` can be a mixture of user authorization and group authorization items.

`item`: A sequence of data that represents a single authorization check.

`itemVersion`: The format of the individual `item`. Version 1 indicates an authorization request for a RACF user ID. Version 2 indicates an authorization request for a RACF group ID.

`itemTag`: An integer that is set by the client for each request `item` and echoed in each response `item`. Its purpose is to assist the client in correlating multiple request responses, and has no influence on the authorization logic or logging.

`userOrGroup`: If `itemVersion` is 1, a RACF user ID whose authority is being checked. Its length cannot exceed 8 characters. If the length is zero, the user value defaults to the user ID associated with the bind user.

If `itemVersion` is 2, a RACF group ID whose authority is being checked. Its length must be from 1 and 8 characters. Optimizations that are used when performing a user ID authorization check are not available when performing a group ID authorization check. For this reason, it is likely that group authorization check executes more slowly than user ID authorization checks.

This field must be specified in uppercase, because RACF user and group names are uppercase, and the remote authorization service does not convert lowercase characters to uppercase.

`resource`: A name to be matched against a RACF profile for authorization checking. The string may not include blank characters. Its length may be from 1 to the maximum RACF profile length defined for the specified class.

`class`: A defined RACF general resource class. It cannot be DATASET, USER, or GROUP. Its length must be from 1 to 8 characters.

If you are checking authorization to resources protected by profiles in a grouping/member class, specify the member class name in the remote authorization request. To obtain accurate results in this case, ensure that the administrator issued SETROPTS RACLIST for the member class.

`access`: The level of authority requested. It must be one of the following integer values:

```
X'01' READ
X'02' UPDATE
X'03' CONTROL
X'04' ALTER
```

`logString`: Any character data from 0 to 200 characters in length. It is appended to an ICTX-defined string in the SMF log record.

The following is the ASN.1 syntax for the **Remote authorization** extended operation. The response OID is 1.3.18.0.2.12.67.

```
responseValue ::= SEQUENCE {
    responseVersion                 INTEGER,
    responseCode                    INTEGER,
    itemList                        SEQUENCE of
      item                            SEQUENCE {
        itemVersion                     INTEGER,
        itemTag                         INTEGER,
        majorCode                       INTEGER,
        minorCode1                      INTEGER,
        minorCode2                      INTEGER,
        minorCode3                      INTEGER
      }
}
```

Where,

`responseValue`: The name for the entire sequence of authorization response data.

`responseVersion`: The format of the response value. Version 1 is the only supported format.

`responseCode`: The greatest error encountered while processing the request. See Table 23 on page 106 for more information about supported responseCodes.

`itemList`: A sequence of one or more items, which allows for multiple authorization results within a single ICTX response.

`item`: A sequence of data that represents the results from a single authorization check.

`itemVersion`: The format of the individual item. Version 1 is the only supported format.

`itemTag`: An integer echoed from the corresponding request `itemTag`. The purpose of the `itemTag` is to assist the client in correlating multiple request responses. `itemTag` has no influence on the authorization logic or logging.

`majorCode`: An integer value representing the result of the authorization check. See Table 24 on page 106 for more information about error major codes.

`minorCode1`: Additional information about the error. See Table 25 on page 107 for more information about error minor codes.

`minorCode2`: Additional information about the error.

`minorCode3`: Additional information about the error.

Remote authorization extended operation response codes

Use the following table to understand the response codes that are generated from the remote authorization processing. The `responseCode` represents the greatest error encountered. You might experience situations where a request item generates an error that is not reflected in the `responseCode`, because that value is overridden by a higher-severity error.

Table 23. Remote authorization responseCodes. Remote authorization responseCodes

responseCode (decimal)	Meaning
0	All request items were processed successfully
28	Empty item list. No items are found within the `itemList` sequence of the extended operation request, so no response items are returned.
61-70	The specified `requestVersion` is not supported. Subtract 60 from the value to determine the highest `requestVersion` that the server supports. `responseCode` 61 indicates that the server supports version 1 requests only. `responseCode` 62 indicates that the highest supported request level is 2.
other	Errors or warnings encountered while processing one or more request items. The value represents the highest `majorCode` in the set of all response items. Verify the major and minor codes that are returned for each item.

Table 24. Remote authorization majorCodes. Remote authorization majorCodes

MajorCode (decimal)	Meaning	Comment
0	Authorized	The user has the requested access to the resource.
2	Warning mode	The user has the requested access because warning mode is enabled for the resource. Warning mode is a feature of RACF that allows installations to try out security policies. Installations can define a profile with the WARNING attribute. When RACF performs an authorization check by using the profile, it logs the event (if there are audit settings) and allows the authorization check to pass successfully. The log records can be monitored to ensure that the new policy is operating as expected before putting the policy into production by turning off the WARNING attribute.
4	Undetermined	No decision is made. The specified resource is not protected by RACF, or RACF is not installed.
8	Unauthorized	The user does not have the requested access to the resource.
12	RACROUTE error	The RACROUTE REQUEST=AUTH service returned an unexpected error. Compare the returned minor codes with the SAF and RACF codes in *z/OS Security Server RACROUTE Macro Reference*.
14	initACEE error	The initACEE callable service returned an unexpected error. Compare the returned minor codes with the SAF and RACF codes in *z/OS Security Server RACF Callable Services*.

Table 24. Remote authorization majorCodes (continued). Remote authorization majorCodes

MajorCode (decimal)	Meaning	Comment
16	Request value error	A value specified in the extended operation request is incorrect or unsupported. Check the returned minor codes to narrow the reason.
20	Request encoding error	A decoding error was encountered indicating the extended operation request contains non-compliant DER encoding, or does not match the documented ASN.1 syntax.
24	Insufficient authority	The requester does not have sufficient authority for the requested function. The user ID associated with the LDAP bound user must have the appropriate access to the FACILITY class profile IRR.LDAP.REMOTE.AUTH.
100	Internal error	An internal error was encountered within the ICTX plug-in.

Table 25. Remote authorization MinorCodes. Remote authorization MinorCodes

MinorCode (decimal)	MinorCode Meaning
0-14	`minorCode1` - the SAF return code `minorCode2` - the RACF return code `minorCode3` - the RACF reason code
16-20	`minorCode1` is the extended operation request parameter number within the item. • 0 - item sequence • 1 - itemVersion • 2 - itemTag • 3 - user • 4 - resource • 5 - class • 6 - access • 7 - logString
	`minorCode2` value indicates one of the following: • 32 - incorrect length • 36 - incorrect value • 40 - encoding error
	`minorCode3` does not have a defined meaning.
24-100	`minorCode1`, `minorCode2`, and `minorCode3` do not have a defined meaning.

Remote authorization audit controls

The auditor can specify whether to log access attempts that are based on user, class, resource, or any criteria as described in *z/OS Security Server RACF Auditor's Guide*. The SMF type 80 records that are generated can be unloaded by using the IRRADU00 utility.

Remote auditing extended operation

The **Remote auditing** extended operation request results in calls to the r_auditx (IRRSAX00) SAF callable service. The results of the r_auditx service are returned to the caller. For more information about the r_auditx callable service, see *z/OS Security Server RACF Callable Services*.

The **Remote auditing** extended operation request must contain the DER-encoding of the ASN.1 syntax. The request OID is 1.3.18.0.2.12.68. The following is the **Remote auditing** extended operation request syntax:

```
requestValue ::= SEQUENCE  {
    requestVersion               INTEGER,
    itemList                     SEQUENCE of
      item                         SEQUENCE {
        itemVersion                  INTEGER,
        itemTag                      INTEGER,
        linkValue                    OCTET STRING SIZE(8),
        violation                    BOOLEAN,
        event                        INTEGER,
        qualifier                    INTEGER,
        class                        OCTET STRING,
        resource                     OCTET STRING,
        logString                    OCTET STRING,
        dataFieldList                SEQUENCE of
          dataField                    SEQUENCE {
            type                         INTEGER,
            value                        OCTET STRING
           }
       }
}
```

Where,

`requestValue`: The name for the entire sequence of audit request data.

`requestVersion`: The format of the request value. Version 1 is the only currently supported format.

`itemList`: A sequence of one or more items, allowing multiple audit records to be written with a single ICTX request. You should limit the size of the entire encoded `RequestValue` to 16 million bytes; however, your encoding routine or LDAP client might impose a stricter limit.

`item`: A sequence of data that represents a single audit record.

`itemVersion`: The format of the individual `item`. Version 1 is the only currently supported format.

`itemTag`: An integer that is set by the client for each request item and echoed in each response item. Its purpose is to assist the client in correlating multiple request responses. The `itemTag` value does not influence the audit processing, and does not appear in the audit record.

`linkValue`: 8 bytes of data that is used to mark related audit records. Specify 8 bytes of zero (X'00') if no such marking is needed.

`violation`: A boolean value that indicates whether the event represents a violation (nonzero ~ TRUE) or not (zero ~ FALSE). The value is used in the R_auditx logging decision.

`event` - An integer 1 - 7 that identifies the security event type. The possible values are:

1 Authentication

2 Authorization

3 Authorization Mapping

4 Key Management

5 Policy Management

6 Administrator Configuration

7 Administrator Action

`qualifier` - An integer 0 - 3 that describes the event result. The possible values are:

0 Success

1 Information

2 Warning

3 Failure

`class` - A defined RACF general resource class that might be used for audit logging determination. It cannot be DATASET, USER, or GROUP. Its length must be from 0 to 8 characters.

`resource` - A name that might be matched against a RACF profile in the specified class for audit logging determination. Its length may be from 0 to 246 characters.

`logString` - Any character data from 0 to 200 characters in length. It is appended to an ICTX-defined string in the SMF log record.

`dataFieldList` - A sequence of type and value pairs that are logged as SMF relocates. Any number of relocates might be included, but the R_auditx service limits the total amount of this relocate data to 20 kilobytes per record. For more information, see *z/OS Security Server RACF Callable Services*.

`dataField` - A sequence of data that represents a single relocate section in an audit record.

`type` - An integer 100 - 116 corresponding to a defined relocate number. The possible values are:

100 SAF identifier for bind user

101 Requester's bind user identifier

102 Originating security domain

103 Originating registry / realm

104 Originating user name

105 Mapped security domain

106 Mapped registry / realm

107 Mapped user name

108 Operation performed

109 Mechanism / object name

110 Method / function used

111 Key / certificate name

112 Caller subject initiating security event

113 Date and time security event occurred

114 Application specific data

115 Identifier for the client submitting the remote audit request

116 Version of the client submitting the remote audit request

`value` - Character data of the associated type that is included in the audit record.

The following is the ASN.1 syntax for the **Remote auditing** extended operation response. The response OID is 1.3.18.0.2.12.69.

```
responseValue ::= SEQUENCE {
    responseVersion                 INTEGER,
    responseCode                    INTEGER,
    itemList                        SEQUENCE of
      item                            SEQUENCE {
        itemVersion                     INTEGER,
        itemTag                         INTEGER,
        majorCode                       INTEGER,
        minorCode1                      INTEGER,
        minorCode2                      INTEGER,
        minorCode3                      INTEGER
      }
}
```

Where,

`responseValue`: The name for the entire sequence of audit response data.

`responseVersion`: The format of the response value. Version 1 is the only supported format.

`responseCode`: The greatest error encountered while processing the request. See Table 26 on page 111 for more information about supported responseCodes.

`itemList`: A sequence of one or more items, which allows for multiple audit results within a single ICTX response.

`item`: A sequence of data that represents the results from a single audit check.

`itemVersion`: The format of the individual item. Version 1 is the only supported format.

`itemTag`: An integer that is echoed from the corresponding request `itemTag`. The purpose of the `itemTag` is to assist the client in correlating multiple request responses. The `itemTag` does not influence the audit processing, and does not appear in the audit record.

`majorCode`: An integer value representing the result of the audit check. See Table 27 for more information about error major codes.

`minorCode1`: Additional information about the error. See Table 28 on page 113 for more information about error minor codes.

`minorCode2`: Additional information about the error.

`minorCode3`: Additional information about the error.

Remote auditing extended operation response codes

Use the following table to understand the response codes that are generated from the remote auditing processing. The `responseCode` represents the greatest error encountered. You might experience situations in which a request item generates an error that is not reflected in the `responseCode`, because that value is overridden by a higher-severity error.

Table 26. Remote auditing responseCodes. Remote auditing responseCodes

responseCode (decimal)	Meaning
0	All request items were processed successfully.
28	Empty item list. No items are found within the `itemList` sequence of the extended operation request, so no response items are returned.
61-70	The specified `requestVersion` is not supported. Subtract 60 from the value to determine the highest `requestVersion` that the server supports. `responseCode` 61 indicates that the server supports version 1 requests only.
other	Errors or warnings that are encountered while processing one or more request items. The value represents the highest `majorCode` in the set of all response items. Verify the major and minor codes returned for each item.

Table 27. Remote auditing majorCodes. Remote auditing majorCodes

majorCodes (decimal)	Meaning	Comment
0	Success	The event is logged successfully.

Table 27. Remote auditing majorCodes (continued). Remote auditing majorCodes

majorCodes (decimal)	Meaning	Comment
2	Warning mode	The event is logged, and warning mode is set for the specified resource. Warning mode is a feature of RACF that allows installations to try out security policies. Installations can define a profile with the WARNING attribute. When RACF performs an authorization check by using the profile, it logs the event (if there are audit settings) and allow the authorization check to pass successfully. The log records can be monitored to ensure that the new policy is operating as expected before putting the policy into production by turning off the WARNING attribute. A remote client resource manager using the remote audit service might simulate RACF warning mode logic after submitting an audit request for a failing authorization event. If the `majorCode` in the response item indicates that the matching resource profile has the warning mode set, the remote client resource manager might allow the check to pass successfully.
3	Logging not required	The event is not logged because no audit controls are set to require it.
4	Undetermined	The event is not logged. The conditions suggested by the following `minorCode` combinations might be intentional administrator settings: • **4,0,0** - RACF is not installed or not active • **8,8,8** - UAUDIT is not set, and class is not active or not RACLISTed • **8,8,12** - UAUDIT is not set, class is active and RACLISTed, and a covering resource profile is not found
8	Unauthorized	The user does not have authority the R_auditx service. The user ID associated with the LDAP server must have at least READ access to the FACILITY class profile IRR.RAUDITX.
12	R_auditx error	The R_auditx service returned an unexpected error. Compare the returned minor codes with the SAF and RACF codes that are documented in *z/OS Security Server RACF Callable Services*.
16	Request value error	A value specified in the extended operation request is incorrect or unsupported. Check the returned minor codes to narrow the reason.

Table 27. Remote auditing majorCodes (continued). Remote auditing majorCodes

majorCodes (decimal)	Meaning	Comment
20	Request encoding error	A decoding error was encountered indicating the extended operation request contains non-compliant DER encoding, or does not match the documented ASN.1 syntax.
24	Insufficient authority	The requester does not have sufficient authority for the requested function. The user ID associated with the LDAP bound user must have at least READ access to the FACILITY class profile IRR.LDAP.REMOTE.AUDIT.
100	Internal error	An internal error was encountered within the ICTX plug-in.

Table 28. Remote auditing MinorCodes. Remote auditing MinorCodes

MinorCode (decimal)	MinorCode Meaning
0-12	`minorCode1`- the SAF return code `minorCode2` - the RACF return code `minorCode3` - the RACF reason code
16-20	`MinorCode1` is the extended operation request parameter number within the item. • 0 - item sequence • 1 - itemVersion • 2 - itemTag • 3 - linkValue • 4 - violation • 5 - event • 6 - qualifier • 7 - class • 8 - resource • 9 - logString • 10 - dataFieldList sequence • 11 - dataField sequence • 12 - type • 13 - value
	`MinorCode2` value indicates one of the following: • 32 - incorrect length • 36 - incorrect value • 40 - encoding error
	`minorCode3` does not have a defined meaning.
24-100	`minorCode1`, `minorCode2`, and `minorCode3` do not have a defined meaning.

Remote audit controls

The **Remote auditing** extended operation uses the R_auditx callable service that is documented in *z/OS Security Server RACF Callable Services* to generate SMF type 83 (subtype 4) audit records. The IRRADU00 utility can then be used to unload the generated SMF type 83 subtype 4 records. Whether the R_auditx service actually writes an audit record for an event, however, depends on the RACF audit controls. If the audit controls do not direct RACF to log an event that was specified in a remote audit request, the R_auditx service does not generate an audit record. If the remote application sends a remote audit request for an operation that is not configured to be logged, is reflected in the remote audit response (a MajorCode of 3 indicates that the event was not logged because it is not required).

There are several ways the RACF auditor can enable logging of events from remote audit requests. For example, because the remote application used a RACF user ID to authenticate with z/OS through an LDAP bind operation, the auditor can enable logging for all remote audit events by setting UAUDIT for that RACF user ID.

If the remote application specified a `class` and `resource` in the remote auditing requests, the auditor can enable logging for the class (using the SETROPTS LOGOPTIONS command) or the resource (using the ADDSD AUDIT, ALTDSD AUDIT, or ALTDSD GLOBALAUDIT commands). To do this, the auditor must know the class and resource that is specified by the application submitting the remote audit requests.

An effective way to gain granular control over which remote application events are logged, is to use the RACF dynamic class descriptor table (dynamic CDT) to define custom classes to represent specific remote applications. When you define a custom class to represent a remote application, you can create resource profiles in the class to represent specific user operations that the application supports. Then, by manipulating the auditing options for the class and profiles, the RACF auditor can determine the type of information logged. For example, imagine a travel application that runs remotely and supports user operations to:

- book a flight
- cancel a flight booking
- check seat availability
- view flight information

A new custom class, @FLIGHTS is created in the dynamic CDT to represent this remote application, and profiles BOOK, CANCEL, SEATCHECK, and VIEW are created in this new class to represent the user operations that are supported. A remote audit request Item is sent by the remote application for each user operation, and the Class and Resource parameters for each Item identifies the travel application and the particular operation. The Class and Resource parameters are used on the remote audit requests for logging determination on the z/OS server. Even though the remote audit record is sent for each operation, whether these events are logged depends on how auditing is configured for the @FLIGHTS class and the BOOK, CANCEL, SEATCHECK, and VIEW profiles. This gives the RACF auditor granular control over which user operations are logged. Configuration of the audit settings on the profiles enables the RACF auditor to, for example, log all BOOK and CANCEL requests while ignoring VIEW and SEATCHECK requests.

SMF Record Type 83 subtype 4 records

The remote audit service logs events as SMF Type 83 subtype 4 records that can be unloaded by using the IRRADU00 utility. Each logged event has a unique event code with a corresponding event code qualifier, or value that indicates whether the

event succeeded, resulted in warning or failure, or was logging event information. The event codes are described in the following table:

Table 29. Remote audit event codes. Remote audit event codes

Event	Command / Service
1	Authentication
2	Authorization
3	Authorization mapping
4	Key management
5	Policy management
6	Administrator configuration
7	Administrator action

The following table describes the event code qualifiers:

Table 30. Remote audit event code qualifiers. Remote audit event code qualifiers

(Common) Event Code Qualifier Dec (Hex)	Description	(Common) Relocate type sections
0	Successful request or authorization.	Common relocates, 100-114
1	Event information.	
2	Not a failure, but might warrant investigation. For authorization event, grace period might be in effect.	
3	Unsuccessful request; unauthorized.	

The following are the remote audit specific extended relocates:

Table 31. Event-specific fields for remote audit events. Event-specific fields for remote audit events

Relocate	XML Tag	DB2® field name	Type	Length	Position		Comments
					Start	End	
100	localUser	SAF_LOCAL_USER	Char	8	3000	3007	SAF identifier for bind user
101	bindUser	SAF_BIND_USER	Char	256	3010	3265	Requesters bind user identifier
102	domain	SAF_DOMAIN	Char	512	3268	3779	Originating security domain
103	regName	SAF_REG_NAME	Char	256	3782	4037	Originating registry / real m
104	regUser	SAF_REG_USER	Char	256	4040	4295	Originating user name
105	mapDomain	SAF_MAP_DOMAIN	Char	512	4298	4809	Mapped security domain
106	mapRegName	SAF_MAP_REG_NAME	Char	256	4812	5067	Mapped registry / realm
107	mapRegUser	SAF_MAP_REG_USER	Char	256	5070	5325	Mapped user name

Table 31. Event-specific fields for remote audit events (continued). Event-specific fields for remote audit events

Relocate	XML Tag	DB2® field name	Type	Length	Position		Comments
					Start	End	
108	action	SAF_ACTION	Char	64	5328	5391	Operation performed
109	object	SAF_OBJECT	Char	64	5394	5457	Mechanism / object name
110	method	SAF_METHOD	Char	64	5460	5523	Method / function used
111	key	SAF_KEY	Char	256	5526	5781	Key / certificate name
112	subjectName	SAF_SUBJECT_NAME	Char	256	5784	6039	Caller subject initiating security event
113	dateTime	SAF_DATE_TIME	Char	32	6042	6073	Date and time security event occurred
114	otherData	SAF_OTHER_DATA	Char	2048	6076	8123	Application specific data
115	clientID	SAF_CLIENT_ID	Char	16	8126	8141	Identifier for client submitting remote audit request.
116	clientVer	SAF_CLIENT_VER	Char	8	8144	8151	Version of client submitting remote audit request.

Chapter 6. Remote crypto plug-in

The remote crypto plug-in in the z/OS LDAP server provides access to a PKCS #11 or CCA services implementation for client applications that do not have local access to one. PKCS #11 is one of the cryptographic standards of Public Key Cryptographic Standards (PKCS) that define a platform independent API to cryptographic tokens. The PKCS #11 standard defines the types of cryptographic tokens and how to use, create, and delete tokens, including how to encrypt, decrypt, and hash data with those tokens. See *Cryptographic Token Interface Standard* for more information about the PKCS #11 standard.

The remote crypto plug-in uses ICSF (Integrated Cryptographic Security Facility) support for its PKCS #11 or CCA services implementation. This plug-in supports the **RemoteCryptoPKCS#11** and **RemoteCryptoCCA** extended operations that allow any LDAP client application with a successfully bound and authorized user to perform any PKCS #11 or CCA services API by invoking the appropriate ICSF callable service. The **RemoteCryptoPKCS#11** and **RemoteCryptoCCA** extended operations are generic extended operations that allow an LDAP client application to specify the same data as if invoking the ICSF callable service locally. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* and *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information about the ICSF callable services that support the PKCS #11 standard.

The remote crypto plug-in supports the level of ICSF shipped with z/OS version 2, release 1.

Configuring the remote crypto plug-in

The remote crypto plug-in is only supported when the LDAP server is running in 64-bit. The remote crypto plug-in can be configured manually or by using the **dsconfig** utility.

To manually configure the remote crypto plug-in, the following **plugin** configuration option line must be specified before any database configuration sections:

```
plugin clientOperation /GLDRCP64 rcrypto_init "enableCCA yes enablePKCS11 yes"
```

Note:

1. Because the remote crypto plug-in runs only in 64-bit mode, this **plugin** configuration option line results in a start error if included in configuration files for servers that are started in 31-bit.
2. The default for `enableCCA` is yes.
3. The default for `enablePKCS11` is yes.

To configure the remote crypto plug-in using the **dsconfig** utility, set the PLUGIN_RCRYPTO input option to on in the **ds.profile** input file of the **dsconfig** utility.

The TCP/IP interface that is used to communicate with the remote crypto plug-in should be secured. Using System SSL is one approach to help secure sensitive data sent to the plug-in. However, there are performance implications that you must consider when using System SSL to protect the communication between clients and

the remote crypto plug-in. Therefore, you should limit the use of the remote crypto plug-in to zEnterprise 196 (z196) systems where applications or appliances deployed on IBM zEnterprise® BladeCenter® Extensions (zBX) communicates with the remote crypto plug-in over the intraensemble data network (IEDN). Use of the IEDN reduces risk of compromising sensitive data in transit, alleviating the need to use System SSL. See *z/OS Communications Server: IP Configuration Guide* for information about configuring an ensemble, defining members of the ensemble, defining IP addresses in the IEDN, and granting permissions for use of the IEDN. When the IEDN is properly configured, update the **listen** configuration options to have the z/OS LDAP server listen on the IEDN sockets.

If limiting access to the IEDN, make sure that the z/OS LDAP server instances that enable the remote crypto plug-in are dedicated to listening on IEDN sockets only. This ensures remote crypto requests always occur over the IEDN. Note that depending on the type of bind (SDBM, native authentication, Kerberos, or SASL EXTERNAL) required for use with remote crypto, that this dedicated server instance might require one or more backends defined.

Setting up authorization to ICSF callable services

When the **RemoteCryptoPKCS#11** or **RemoteCryptoCCA** extended operations are used, the authenticated user must have the appropriate access to call the underlying ICSF callable service. Before using the **RemoteCryptoPKCS#11** or **RemoteCryptoCCA** extended operation, the authenticated user must resolve to a SAF identity, which is then granted authorization to the underlying ICSF callable services. The following binds in the z/OS LDAP server can resolve to a SAF identity:

- Simple bind to the SDBM backend. See SDBM authorization for more information.
- LDBM or TDBM native authentication bind. Native authentication allows the use of an LDBM or TDBM entry but the password or password phrase is stored in SAF. See Native authentication for more information.
- Kerberos (GSSAPI) bind. See Kerberos authentication for more information.
- SASL EXTERNAL certificate bind where the certificate is mapped to a SAF user. The **sslMapCertificate** configuration option must be set to fail the SASL EXTERNAL bind request if the user cannot be mapped to a SAF or RACF user. See Setting up for SSL/TLS for more information about mapping certificates to users.

The security administrator must decide the authorization that each authenticated user must have when accessing the remote crypto plug-in and using the **RemoteCryptoPKCS#11** or **RemoteCryptoCCA** extended operation, which calls the underlying ICSF callable service. See Table 32 on page 120 for the ICSF callable services that are supported by the remote crypto plug-in. If the CSFSERV class is active on your system, ICSF performs access control checks on the underlying callable services. To provide users access to all supported ICSF callable services other than those protected by discrete profiles, the security administrator can define a generic profile in the CSFSERV class and permit users READ access to the generic profile. For example:

```
RDEFINE CSFSERV CSF* UACC(NONE)
PERMIT CSF* CLASS(CSFSERV) ACCESS(READ) ID(USER1)
SETROPTS CLASSACT(CSFSERV)
SETROPTS RACLIST(CSFSERV) REFRESH
```

Additionally, specific ICSF callable services can be protected individually with discrete profiles, regardless of the presence of a generic profile. For example, to provide USER1 access specifically to the CSFPTRC callable service, the security administrator could issue the following commands:

```
RDEFINE CSFSERV CSF1TRC UACC(NONE)
PERMIT CSF1TRC CLASS(CSFSERV) ACCESS(READ) ID(USER1)
SETROPTS CLASSACT(CSFSERV)
SETROPTS RACLIST(CSFSERV) REFRESH
```

See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* and *z/OS Security Server RACF Security Administrator's Guide* for more information about the supported CSFSERV class and granting authorization.

Setting up authorization to PKCS #11 tokens and objects

The PKCS #11 standard is for systems that grant access to token information based on a PIN (personal identification number). The types of users that are defined for this standard are the standard user (User) and security officer (SO), each having a PIN. The SO initializes a token (zero the contents) and set the User's PIN. The SO can also access the public objects on the token, but not the private ones. The User has access to the private objects on a token and has the power to change its own PIN. The User cannot reinitialize a token. The PIN that a user enters determines which role that user takes. A user can fill both roles by knowing both PINs.

z/OS does not use PINs. Instead, profiles in the SAF CRYPTOZ class control access to tokens. For each token, there are two resources in the CRYPTOZ class for controlling access to tokens:

- The resource USER.*token-name* controls the access of the User role to the token.
- The resource SO.*token-name* controls the access of the SO role to the token.

See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information about the types of SO and USER profiles in the CRYPTOZ class. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for information about the necessary authorization that is needed to access the PKCS #11 tokens and objects.

ICSF callable services supported by the RemoteCryptoPKCS#11 extended operation

The **RemoteCryptoPKCS#11** extended operation is a generic extended operation that provides remote access to the PKCS#11 related services available in ICSF (Integrated Cryptographic Service Facility). This extended operation can help if you want to maintain z/OS as the centralized PKCS#11 repository when implementing PKCS#11 on non-z/OS systems. Table 32 on page 120 includes the ICSF PKCS #11 callable service routines that are supported by the **RemoteCryptoPKCS#11** extended operation. It also includes the equivalent PKCS #11 routines and the PKCS #11 function category that each of the ICSF callable services belong to. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about these ICSF callable services. See *Cryptographic Token Interface Standard* for more information about the PKCS #11 standard.

Table 32. ICSF callable services supported by the RemoteCryptoPKCS#11 extended operation. ICSF callable services supported by the RemoteCryptoPKCS#11 extended operation

ICSF callable service and helper function	Description	Used by PKCS #11 API	PKCS #11 function category	Function code	Request tag	Response tag
CSFIQF - Query facility	Retrieve information about ICSF, the cryptographic coprocessors, and the CCA code in the coprocessors.	C_GetMechanismInfo	General purpose	FC_CSFIQF	REQ_CSFIQF	RES_CSFIQF
CSFPDMK - Derive multiple keys	Generate multiple secret key objects and protocol-dependent keying material from an existing secret key object.	C_DeriveKey	Key management	FC_CSFPDMK	REQ_CSFPDMK	RES_CSFPDMK
CSFPDVK - Derive key	Generate a new secret key object from an existing key object.	C_DeriveKey	Key management	FC_CSFPDVK	REQ_CSFPDVK	RES_CSFPDVK
CSFPGAV - Get attribute value	Retrieve the attributes of an object.	C_GetAttributeValue	Object management	FC_CSFPGAV	REQ_CSFPGAV	RES_CSFPGAV
CSFPGKP - Generate key pair	Generate an RSA, DSA, elliptic curve, or Diffie-Hellman key pair.	C_GenerateKeyPair	Key management	FC_CSFPGKP	REQ_CSFPGKP	RES_CSFPGKP
CSFPGSK - Generate secret key	Generate a secret key or set of domain parameters.	C_GenerateKey	Key management	FC_CSFPGSK	REQ_CSFPGSK	RES_CSFPGSK
CSFPHMG - Generate HMAC	Generate a hashed message authentication code (MAC).	C_SignInit C_Sign C_SignUpdate C_Sign	Signing and verifying	FC_CSFPHMG	REQ_CSFPHMG	RES_CSFPHMG
CSFPHMV - Verify HMAC	Verify a hash message authentication code (MAC).	C_VerifyInit C_Verify C_VerifyUpdate C_VerifyFinal	Signing and verifying	FC_CSFPHMV	REQ_CSFPHMV	RES_CSFPHMV
CSFPOWH - One-way hash, sign, or verify	Generate a one-way hash on specified text, sign specified text, or verify a signature on specified text.	C_DigestInit C_Digest C_DigestUpdate C_DigestFinal	Message digesting	FC_CSFPOWH	REQ_CSFPOWH	RES_CSFPOWH
CSFPPKS - Private key sign	Decrypt or sign data by using an RSA private key using zero-pad or PKCS #1 V1.5 formatting, sign data by using a DSA private key, or sign data by using an elliptic curve private key in combination with DSA.	C_SignInit C_Sign C_SignUpdate C_SignFinal	Signing and verifying	FC_CSFPPKS	REQ_CSFPPKS	RES_CSFPPKS
CSFPPKV - Public key verify	Encrypt or verify data by using an RSA public key using zero-pad or PKCS #1 V1.5 formatting, verify a signature by using a DSA public key, or verify a signature by using an elliptic curve public key in combination with DSA.	C_VerifyInit C_Verify C_VerifyUpdate C_VerifyFinal	Signing and verifying	FC_CSFPPKV	REQ_CSFPPKV	RES_CSFPPKV
CSFPSAV - Set attribute	Update the attributes of an object.	C_SetAttributeValue	Object management	FC_CSFPSAV	REQ_CSFPSAV	RES_CSFPSAV
CSFPSKD - Secret key decrypt	Decipher data by using a clear symmetric key.	C_DecryptInit C_Decrypt C_DecryptUpdate C_DecryptFinal	Encryption and decryption	FC_CSFPSKD	REQ_CSFPSKD	RES_CSFPSKD
CSFPSKE - Secret key encrypt	Encipher data by using a clear symmetric key.	C_EncryptInit C_Encrypt C_EncryptUpdate C_EncryptFinal	Encryption and decryption	FC_CSFPSKE	REQ_CSFPSKE	RES_CSFPSKE
CSFPTRC - Token record create	Initialize or reinitialize a z/OS PKCS #11 token, create, or copy a token object in the token data set, or create or copy a session object for the current PKCS #11 session.	C_CreateObject C_CopyObject	Object management	FC_CSFPTRC	REQ_CSFPTRC	RES_CSFPTRC
CSFPTRD - Token record delete	Delete a z/OS PKCS #11 token object, session object, or state object.	C_DestroyObject	Object management	FC_CSFPTRD	REQ_CSFPTRD	RES_CSFPTRD
CSFPTRL - Token record list	Obtain a list of z/OS PKCS #11 tokens or obtain a list of token and session objects for a token.	C_GetSlotList C_FindObjects	General purpose and object management	FC_CSFPTRL	REQ_CSFPTRL	RES_CSFPTRL
CSFPUWK - Unwrap key	Unwrap and create a key object by using another key.	C_UnwrapKey	Key management	FC_CSFPUWK	REQ_CSFPUWK	RES_CSFPUWK
CSFPWPK - Wrap key	Wrap a key with another key.	C_WrapKey	Key management	FC_CSFPWPK	REQ_CSFPWPK	RES_CSFPWPK
GLDTRD - Internal-only used to gain access to the chaining data	An internal-only extended operation request used to gain access to the chaining data returned by several ICSF callable services.	N/A	N/A	FC_GLDTRD	REQ_GLDTRD	RES_GLDTRD
CSFIQA - ICSF query algorithm	Obtain information about the cryptographic and hashing algorithms available	C_GetMechanismInfo	General purpose	FC_CSFIQA	REQ_CSFIQA	RES_CSFIQA

Because the **RemoteCryptoPKCS#11** extended operation is a generic extended operation, it supports all of the ICSF PKCS #11 callable services in Table 32.

Therefore, the extended operation has a common BER encoding for both the request and response values as many of the ICSF PKCS #11 callable service routines have common interfaces (inputs and outputs on the callable service routines).

The request and response values for the **RemoteCryptoPKCS#11** extended operation follows. For more information about ASN.1 (Abstract Syntax Notation One) and BER (Basic Encoding Rules), see:

`ftp://ftp.rsa.com/pub/pkcs/ascii/layman.asc`

- **Request values:** The following ASN.1 syntax describes the BER encoding of the request value for the **RemoteCryptoPKCS#11** extended operation request. Each ICSF PKCS #11 callable service has additional callable service-specific information that is encoded in the `requestData` field. The request OID is 1.3.18.0.2.12.83.

```
requestValue ::= SEQUENCE {
    version         INTEGER,
    exitData        OCTET STRING,
    handle          OCTET STRING,
    ruleArraySeq    RuleArraySeq,
    requestData     CSFPInput
}
```

  Where,

  `version`: Identifies which version of the interface is being used. Currently, the only value supported is 1. If the interface is extended in the future, then other values are supported.

  `exitData`: Identifies the data that is passed to the installation exit. See the appropriate ICSF callable service routine in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information. If the ICSF callable service routine does not need `exitData`, then specify a zero length octet string.

  `handle`: Identifies the input generic handle that is needed for the ICSF PKCS #11 callable service routine. If the ICSF callable service routine or helper function does not need a handle, then specify a zero length octet string. See Table 33 for information about what this handle represents for the ICSF PKCS #11 callable service or extended operation request. See the appropriate ICSF callable service routine in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the handle.

Table 33. requestValue handle descriptions. requestValue handle descriptions

ICSF callable service and helper function	Handle description
CSFIQF - ICSF query facility	Not used.
CSFPDMK - Derive multiple keys	The 44-byte handle of the base key object.
CSFPDVK - Derive key	The 44-byte handle of the source key object.
CSFPGAV - Get attribute value	The 44-byte handle of the object.
CSFPGKP - Generate key pair	The 44-byte handle of the token of the key objects.
CSFPGSK - Generate secret key	The 44-byte handle of the token.
CSFPHMG - Generate HMAC	The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests.
CSFPHMV - Verify HMAC	The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests.

Table 33. requestValue handle descriptions (continued). requestValue handle descriptions

ICSF callable service and helper function	Handle description
CSFPOWH - One-way hash, sign, or verify	For hash requests, this is the 44-byte name of the token to which this hash operation is related. The first 32 bytes of the handle are meaningful. The remaining 12 bytes are reserved. For sign and verify requests, this is the 44-byte handle to the key object that is to be used. For FIRST and MIDDLE chaining requests, only the first 32 bytes of the handle are meaningful, to identify the token.
CSFPPKS - Private key sign	The 44-byte handle of a private key object.
CSFPPKV - Public key verify	The 44-byte handle of public key object.
CSFPSAV - Set attribute	The 44-byte handle of the object.
CSFPSKD - Secret key decrypt	The 44-byte handle of secret key object.
CSFPSKE - Secret key encrypt	The 44-byte handle of secret key object.
CSFPTRC - Token record create	The 44-byte name of the z/OS PKCS #11 token to be initialized, or the token handle of the object to be created or copied. For the create or re-create functions, the first 32 bytes of the handle are meaningful on input. For the copy function, all 44 bytes of the handle are significant on input.
CSFPTRD - Token record delete	The 44-byte name of the token or object to be deleted.
CSFPTRL - Token record list	For tokens, an empty string (blanks) for the first call, or the 44-byte handle of the last token found for subsequent calls. For objects, the 44-byte handle of the token for the first call, or the 44-byte handle of the last object found for subsequent calls.
CSFPUWK - Unwrap key	The 44-byte handle of the private key or secret key object to unwrap the key.
CSFPWPK - Wrap key	The 44-byte handle of the secret key or private key object to be wrapped or the `source_key_handle` parameter on the callable service routine. **Note:** The `wrapping_key_handle` parameter in the callable service routine is specified in the `wrappingHandle` parameter of the `WPKInput`.
GLDTRD - Internal-only used to gain access to the chaining data.	Not used.
CSFIQA - ICSF query algorithm	Not used.

`ruleArraySeq`: Is a sequence that identifies an array of keywords that provide control information to the ICSF callable services. The ASN.1 description is:

```
RuleArraySeq ::= SEQUENCE {
   ruleArrayCount     INTEGER,
   ruleArray          OCTET STRING
}
```

See the appropriate ICSF callable service routine in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the rule array. The integer `ruleArrayCount` identifies the number of rules in the array and is referred to as `rule_array_count` in the ICSF callable service routine descriptions. The octet string `ruleArray` is the actual array of rules, and is referred to as `rule_array` in the ICSF callable service routine descriptions. The number `ruleArrayCount` must be consistent with the contents of the octet string

rule_array. The length of ruleArray octet string must be a multiple of 8 times the number that is specified in the ruleArrayCount field. If an individual rule specified in the ruleArray is fewer than 8 characters, it must be blank padded on the right to 8 full characters. You must always use the entire sequence. To specify an empty rule array, specify ruleArrayCount as zero, and ruleArray as a zero length octet string.

requestData: Identifies the extended operation request-specific data for the ICSF callable service. See Table 32 on page 120 for the appropriate request tag for CSFPInput. The values for these tags can be found in the ldap_rcrypto.h sample header file.

Where,

```
CSFPInput ::= CHOICE {
    IQF    [CSFIQF]           IQFInput,
    DMK    [CSFPDMK]          DMKInput,
    DVK    [CSFPDVK]          DVKInput,
    GAV    [CSFPGAV]          GAVInput,
    GKP    [CSFPGKP]          GKPInput,
    GSK    [CSFPGSK]          GSKInput,
    HMG    [CSFPHMG]          HMGInput,
    HMV    [CSFPHMV]          HMVInput,
    OWH    [CSFPOWH]          OWHInput,
    PKS    [CSFPPKS]          PKSInput,
    PKV    [CSFPPKV]          PKVInput,
    SAV    [CSFPSAV]          SAVInput,
    SKD    [CSFPSKD]          SKDInput,
    SKE    [CSFPSKE]          SKEInput,
    TRC    [CSFPTRC]          TRCInput,
    TRD    [CSFPTRD]          TRDInput,
    TRL    [CSFPTRL]          TRLInput,
    UWK    [CSFPUWK]          UWKInput,
    WPK    [CSFPWPK]          WPKInput,
    GLDTRD [GLDTRD]           GLDTRDInput,
    IQA    [CSFIQA]           IQAInput
}
```

- **Response values:** The following ASN.1 syntax describes the BER encoding of the response value for the **RemoteCryptoPKCS#11** extended operation response. The handle, the ICSF return code, and ICSF reason codes are returned from the underlying ICSF callable service routine. Each ICSF PKCS #11 callable service returns additional response-specific data in the responseData field. The response OID is 1.3.18.0.2.12.84.

```
responseValue ::= SEQUENCE {
   version        INTEGER,
   ICSFRc         INTEGER (0 .. MaxCSFPInteger),
   ICSFRsnCode    INTEGER (0 .. MaxCSFPInteger),
   exitData       OCTET STRING,
   handle         OCTET STRING,
   responseData   CSFPOutput
}
```

Where,

version: Identifies which version of the interface is being used. Currently the only value supported is 1. If the interface is extended in the future, then other values are supported.

ICSFRc: Identifies the ICSF return code. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the ICSF return code error.

ICSFRsnCode: Identifies the ICSF reason code. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the ICSF reason code error.

`exitData`: Identifies the data that is passed to the installation exit. See the appropriate ICSF callable service routine in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information. A zero length octet string indicates that no `exitData` is returned.

`handle`: Identifies the handle that is returned from the ICSF PKCS #11 callable service routine. In some instances, the handle that is returned in the `responseValue` is the same handle that was specified in the `requestValue`. See Table 34 for information about what this handle represents for the ICSF PKCS #11 callable service or extended operation response. See the appropriate ICSF callable service routine in *z/OS Cryptographic Services ICSF Application Programmer's Guide* or more information about the handle.

Table 34. responseValue handle descriptions. responseValue handle descriptions

ICSF callable service and helper function	Handle description
CSFIQF - ICSF query facility	Not used.
CSFPDMK - Derive multiple keys	The 44-byte handle of the base key object. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPDVK - Derive key	The 44-byte handle of the secret key object that was derived or the `target_key_object_handle` parameter of the callable service. If the callable service returns an error, this handle is encoded as a zero length octet string.
CSFPGAV - Get attribute value	The 44-byte handle of the object. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPGKP - Generate key pair	The 44-byte handle of the new public key object or the `public_key_object_handle` parameter of the callable service. If the callable service returns an error, this handle is encoded as a zero length octet string.
CSFPGSK - Generate secret key	The 44-byte handle of the new secret key object or the returned `handle` parameter of the callable service. If the callable service returns an error, this handle is the same as the `requestValue handle`.
CSFPHMG - Generate HMAC	The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPHMV - Verify HMAC	The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.

Table 34. responseValue handle descriptions (continued). responseValue handle descriptions

ICSF callable service and helper function	Handle description
CSFPOWH - One-way hash, sign, or verify	For hash requests, this is the 44-byte name of the token to which this hash operation is related. The first 32 bytes of the handle are meaningful. The remaining 12 bytes are reserved. For sign and verify requests, this is the 44-byte handle to the key object that is to be used. For FIRST and MIDDLE chaining requests, only the first 32 bytes of the handle are meaningful, to identify the token. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPPKS - Private key sign	The 44-byte handle of a private key object. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPPKV - Public key verify	The 44-byte handle of public key object. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPSAV - Set attribute	The 44-byte handle of the object. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPSKD - Secret key decrypt	The 44-byte handle of secret key object. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPSKE - Secret key encrypt	The 44-byte handle of secret key object. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPTRC - Token record create	The 44-byte handle of the z/OS PKCS #11 token or object created if the CREATE or RECREATE option is specified in the input `ruleArray`. This is the output `handle` parameter of the callable service.
CSFPTRD - Token record delete	The 44-byte name of the token or object to be deleted. **Note:** This handle value is the same as what was specified in the `requestValue handle`. It is not updated on the extended operation response.
CSFPTRL - Token record list	Not used. **Note:** The list of tokens is returned in the `OutputList` of the `TRLOutput`.
CSFPUWK - Unwrap key	The 44-byte handle of the secret key or private key object created for the unwrapped key. The object uses the token name of the unwrapping key object. This is the `target_key_handle` parameter of the callable service. If the callable service returns an error, this handle is encoded as a zero length octet string.

Table 34. responseValue handle descriptions (continued). responseValue handle descriptions

ICSF callable service and helper function	Handle description
CSFPWPK - Wrap key	The 44-byte handle of the secret key or private key object to be wrapped or the `source_key_handle` parameter on the callable service routine. **Note:** The `wrapping_key_handle` parameter in the callable service routine is specified in the `wrappingHandle` parameter of the `WPKInput`.
GLDTRD - Internal-only used to gain access to the chaining data.	Not used.
CSFIQA - ICSF query algorithm	Not used.

`responseData`: Identifies the ICSF PKCS #11 request-specific extended operation response data. See Table 32 on page 120 for the appropriate response tag for `CSFPOutput`. The values for these tags can be found in the ldap_rcypto.h sample header file.

Where,

```
CSFPOutput ::= CHOICE {
    IQF    [CSFIQF]           IQFOutput,
    DMK    [CSFPDMK]          DMKOutput,
    DVK    [CSFPDVK]          DVKOutput,
    GAV    [CSFPGAV]          GAVOutput,
    GKP    [CSFPGKP]          GKPOutput,
    GSK    [CSFPGSK]          GSKOutput,
    HMG    [CSFPHMG]          HMGOutput,
    HMV    [CSFPHMV]          HMVOutput,
    OWH    [CSFPOWH]          OWHOutput,
    PKS    [CSFPPKS]          PKSOutput,
    PKV    [CSFPPKV]          PKVOutput,
    SAV    [CSFPSAV]          SAVOutput,
    SKD    [CSFPSKD]          SKDOutput,
    SKE    [CSFPSKE]          SKEOutput,
    TRC    [CSFPTRC]          TRCOutput,
    TRD    [CSFPTRD]          TRDOutput,
    TRL    [CSFPTRL]          TRLOutput,
    UWK    [CSFPUWK]          UWKOutput,
    WPK    [CSFPWPK]          WPKOutput,
    GLDTRD [GLDTRD]           GLDTRDOutput,
    IQA    [CSFIQA]           IQAOutput
}
```

Common ASN.1 encodings used by the RemoteCryptoPKCS#11 extended operation

This section describes the common ASN.1 encodings that are used by the **RemoteCryptoPKCS#11** extended operation request and response. Table 35 on page 127 includes how PKCS #11 attribute types must be encoded using the `attributeValue`.

```
MaxCSFPInteger ::= 2^31 - 1

Attributes ::= SEQUENCE OF SEQUENCE {
   attrName          INTEGER,
   attrValue         AttributeValue
}

AttributeValue ::= CHOICE {
   charValue         [0] OCTET STRING,
   intValue          [1] INTEGER
}
```

Where,

`attrName`: Indicates an integer identifier for the PKCS #11 attribute name for the key or object that you are creating or searching for. The attribute name identifiers that are supported vary depending on the ICSF callable service that is invoked and the key or object type that is being created or searched for. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* and *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information.

`attrValue`: Specifies the value for the `attrName`. The `attrValue` can be an OCTET STRING or an integer that is depending on the `attrName`. The client application must encode this value appropriately for use with the specified `attrName`. See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for information about the supported data types for the specified `attrName`.

Table 35. Encoding PKCS #11 attribute types using the attributeValue. Encoding PKCS #11 attribute types using the attributeValue

PKCS #11 attribute data type	**ASN.1 encoding**
CKO_OBJECT_CLASS	OCTET STRING
CK_BOOL	OCTET STRING
"Printable EBCDIC string"	OCTET STRING (Encode in ASCII/UTF8 on the client and is automatically converted to EBCDIC on the server side)
"Byte array"	OCTET STRING
CK_DATE	OCTET STRING
CK_MECHANISM_TYPE	OCTET STRING
CK_KEY_TYPE	OCTET STRING
"Big integer"	OCTET STRING (big-endian formatting)
CK_CERTIFICATE_TYPE	INTEGER
CK_ULONG	INTEGER (00.. `MaxCSFPInteger`)
CK_KEY_TYPE	INTEGER

ICSF state cleanup ASN.1 syntaxes

The **RemoteCryptoPKCS#11** extended operation provides capability to interpret the chaining data without an object handle, and call the CSFPTRD ICSF callable service with the appropriate data to clean up ICSF resources that are used to handle the multi-part processing. This function allows users to avoid interpreting the chaining data to determine whether the ICSF resources must be cleaned up. The integer `ruleArrayCount` in the `requestValue` of this function identifies the number of rules in the array, and is referred to as `rule_array_count` in the CSFPTRD ICSF callable service routine descriptions. The octet string `ruleArray` in the `requestValue` of this function is the actual array of rules, and is referred to as `rule_array` in the ICSF callable service routine descriptions. Since this function always deletes an object, always set the `ruleArrayCount` to 1, and the `ruleArray` to "OBJECT ". See the CSFPTRD callable service in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the rule array or other parameters of this callable service.

```
GLDTRDInput ::= chainData
chainData ::= OCTET STRING
```

Where,

chainData: An octet string that specifies the chaining data to use to clean up ICSF resources.

```
GLDTRDOutput ::= NULL
```

General purpose-related ASN.1 syntaxes

ICSF provides a series of callable services that allow users to query the status of ICSF and the encryption and hashing algorithms that are supported by ICSF. The **RemoteCryptoPKCS#11** extended operation allows the CSFIQA and CSFIQF callable services to be available for remote invocation.

ICSF Query facility (CSFIQF) ASN.1 syntaxes

```
IQFInput ::= SEQUENCE {
   returnedDataMaxLen       INTEGER,
   reservedData             OCTET STRING
}
```

Where,

returnedDataMaxLen: An integer that specifies the length, in bytes, of the area that is provided by the caller that is receiving the returned data value from ICSF in the CSFIQF callable service.

reservedData: An octet string that identifies the data that is specified in the reserved data in the CSFIQF callable service. The reserved data is ignored by ICSF.

```
IQFOutput ::= SEQUENCE {
   returnedDataLen       INTEGER,
   returnedData          OCTET STRING
}
```

Where,

returnedDataLen: An integer that contains the length, in bytes, of the returned data. On a successful call, this is the length of the returnedData octet string. If the returned ICSF return code and reason code indicate the input returnedDataMaxLen is too small, this value indicates the size that is needed for returnedDataMaxLen.

returnedData: An octet string that specifies the returned data from the CSFIQF callable service.

ICSF Query algorithm (CSFIQA) ASN.1 syntaxes

```
IQAInput ::= SEQUENCE {
   returnedDataMaxLen         INTEGER,
   reservedData               OCTET STRING
}
```

Where,

returnedDataMaxLen: An integer that specifies the length, in bytes, of the area that is provided by the caller that is receiving the returned data value that is returned from ICSF in the CSFIQA callable service.

reservedData: An octet string that identifies the data that is specified in the reserved data in the CSFIQA callable service. The reserved data is ignored by ICSF.

```
IQAOutput ::= SEQUENCE {
   returnedDataLen       INTEGER,
   returnedData          OCTET STRING
}
```

Where,

`returnedDataLen`: An integer that contains the length, in bytes, of the returned data. On a successful call, this is the length of the `returnedData` octet string. If the returned ICSF return code and reason code indicate the input `returnedDataMaxLen` is too small, this value indicates the size that is needed for `returnedDataMaxLen`.

`returnedData`: An octet string that specifies the returned data from the CSFIQA callable service

Object management ASN.1 syntaxes

ICSF provides a series of callable services to allow users to create or delete PKCS #11 tokens or objects, retrieve information of PKCS #11 tokens or objects, and save or retrieve attributes values of a PKCS #11 object. The **RemoteCryptoPKCS#11** extended operation allows the CSFPTRC, CSFPTRD, CSFPTRL, CSFPGAV, and CSFPSAV ICSF callable services to be available for remote invocation.

Get attribute value (CSFPGAV) ASN.1 syntaxes

```
GAVInput ::= attrListLen
attrListLen ::= INTEGER (0 .. MaxCSFPInteger)
```

Where,

`attrListLen`: An integer that specifies the length of the buffer (in bytes) allocated to hold the attributes that are returned from ICSF in the CSFPGAV callable service.

```
GAVOutput ::= SEQUENCE {
   attrListLen         INTEGER (0 .. MaxCSFPInteger),
   attrList            Attributes
}
```

Where,

`attrListLen`: An integer that specifies the length (in bytes) of the `attrList` returned from ICSF in the CSFPGAV callable service. If the `attrListLen` specified on input is sufficient to hold all attributes, this is the same as `attrListLen` on input; otherwise, this is the minimum length needed.

`attrList`: A list of object attributes that are returned from ICSF in the CSFPGAV callable service

Set attribute value (CSFPSAV) ASN.1 syntaxes

```
SAVInput ::= attrList
attrList ::= Attributes
```

Where,

`attrList`: A list of attributes to be updated in the object in the CSFPSAV callable service.

```
SAVOutput ::= NULL
```

Token record create (CSFPTRC) ASN.1 syntaxes

```
TRCInput ::= SEQUENCE {
   trcAttrs ::= CHOICE {
       tokenAttrString     [0]   OCTET STRING,
       objectAttrList      [1]   Attributes
    }
}
```

Where,

trcAttrs: The token attributes string for the token that is being created or re-created, or the list of object attributes for the object that is being created.

tokenAttrString: When creating or re-creating a token ("TOKENbbb" specified in rule_array), this is a 68-byte string of the token attributes for the token that is being created or re-created.

objectAttrList: When creating or copying an object ("OBJECTbb" specified in rule_array), this is a list of object attributes for the object that is being created or copied. Note that for object copy ("COPYbbbb" specified in rule_array), this attributes list contains no attribute.

Note: b represents a blank character.

```
TRCOutput ::= NULL
```

Token record delete (CSFPTRD) ASN.1 syntaxes

```
TRDInput ::= NULL
TRDOutput ::= NULL
```

Token record list (CSFPTRL) ASN.1 syntaxes

```
TRLInput ::= SEQUENCE {
   inListLen                   INTEGER (0 .. MaxCSFPInteger),
   maxHandleCount              INTEGER (0 .. MaxCSFPInteger),
   searchTemplate          [0] Attributes OPTIONAL
}
```

Where,

inListLen: An integer that specifies the length, in bytes, of the buffer that is to hold the contents of the output list that is returned from ICSF in the CSFPTRL callable service.

maxHandleCount: An integer that specifies the maximum number of tokens or object handles that are to be returned in the output list from ICSF in the CSFPTRL callable service.

searchTemplate: A list of criteria (attribute values) that an object must meet to be added to the output list returned from ICSF in the CSFPTRL callable service. For requesting tokens ("TOKENbbb" specified in rule_array), do not complete this field; for requesting session objects ("OBJECTbb" specified in rule_array), this field is optional.

Note: b represents a blank character.

```
TRLOutput ::= SEQUENCE {
   outListLen             INTEGER (0 .. MaxCSFPInteger),
   outList                CHOICE {
```

```
      tokenList      [0]  OCTET STRING,
      handleList     [1]  OCTET STRING
   }
}
```

Where,

outListLen: Number of bytes used for the outList parameter. If the inListLen specified on input is insufficient to hold one record, it is set to the minimum length needed for one record.

tokenList: A string containing the list of z/OS PKCS #11 tokens that the user has SAF authorization to. Each token record is 116 bytes long.

handleList: A string containing a list of token handles or a list of session objects handles for a specific token that the user has SAF authorization to. Only the objects that meet the search criteria are returned. Each object handle is 44 bytes long.

Signing and verifying ASN.1 syntaxes

ICSF provides several callable services to support signing and verifying. The **RemoteCryptoPKCS#11** extended operation allows the CSFPHMG, CSFPHMV, CSFPPKS, and CSFPPKV ICSF callable services to be available for remote invocation.

Generate HMAC (CSFPHMG) ASN.1 syntaxes

```
HMGInput ::= SEQUENCE {
   text                    OCTET STRING,
   chainData               OCTET STRING,
   hmacLength              INTEGER (0 .. MaxCSFPInteger)
}
```

Where,

text: An octet string that identifies the data that is being used to generate an HMAC hash in the CSFPHMG callable service.

chainData: An octet string that specifies the chaining data that is maintained during multipart HMAC hashing in the CSFPHMG callable service.

hmacLength: Ignored by ICSF. The caller must provide an area large enough to receive the generated HMAC data as defined by the mechanism that is specified in the rule array.

```
HMGOutput ::= SEQUENCE {
   chainData               OCTET STRING,
   hmac                    OCTET STRING,
   hmacLength              INTEGER
}
```

Where,

chainData: An octet string that contains the updated chaining data for multipart HMAC generation that must be specified on the subsequent request in HMGInput.

hmac: An octet string that contains the HMAC hash that are generated by the CSFPHMG callable service on a LAST or ONLY request. For a FIRST or MIDDLE request, the extended operation returns this as a zero length octet string.

hmacLength: Ignored by ICSF.

Verify HMAC (CSFPHMV) ASN.1 syntaxes

```
HMVInput ::= SEQUENCE {
   text                    OCTET STRING,
   chainData               OCTET STRING,
   hmac                    OCTET STRING
}
```

Where,

text: An octet string that identifies the text value for which an HMAC is verified by using the CSFPHMV callable service.

chainData: An octet string that specifies the chaining data that is maintained during multipart HMAC hash verification in the CSFPHMV callable service.

hmac: An octet string that identifies the HMAC hash value that is being verified by the CSFPHMV callable service on a LAST or ONLY request. For a FIRST or MIDDLE request, this value is ignored.

```
HMVOutput ::= chainData
chainData ::= OCTET STRING
```

Where,

chainData: An octet string that contains the updated chaining data for multipart HMAC hash verification that must be specified on the subsequent call in HMVInput.

Public key sign (CSFPPKS) ASN.1 syntaxes

```
PKSInput ::= SEQUENCE {
   cipherValue             OCTET STRING,
   clearValueMaxLen        INTEGER (0 .. MaxCSFPInteger)
}
```

Where,

cipherValue: An octet string that specifies the value that is being signed or decrypted by the CSFPPKS callable service.

clearValueMaxLen: An integer that specifies the length, in bytes, of the area that is provided by the caller that is receiving the clear value that is returned from ICSF in the CSFPPKS callable service.

```
PKSOutput ::= SEQUENCE {
   clearValue              OCTET STRING,
   clearValueLen           INTEGER (0 .. MaxCSFPInteger)
}
```

Where,

clearValue: An octet string that contains the generated signature or decrypted data from the CSFPPKS callable service.

clearValueLen: An integer that contains the length, in bytes, of the generated signature or decrypted data. On a successful call, this is the length of the returned clearValue octet string. If the returned ICSF return code and reason code indicate

the input clearValueMaxLen is too small, the returned value of clearValueLen indicates the size that is needed for clearValueMaxLen.

Public key verify (CSFPPKV) ASN.1 syntaxes

```
PKVInput ::= SEQUENCE {
   clearValue               OCTET STRING,
   CHOICE {
      cipherValueMaxLen        [0] INTEGER (0 .. MaxCSFPInteger),
      cipherValue              [1] OCTET STRING
   }
}
```

Where,

clearValue: An octet string that specifies the data that is being encrypted on an ENCRYPT call, otherwise, the signature that is being verified by the CSFPPKV callable service.

cipherValueMaxLen: For ENCRYPT only, an integer that specifies the length, in bytes, of the area that is provided by the caller that is receiving the data that is returned in cipherValue by the CSFPPKV callable service.

cipherValue: For signature verification only, an octet string that specifies the data that is being verified by the CSFPPKV callable service.

```
PKVOutput ::= SEQUENCE {
   cipherValue               OCTET STRING OPTIONAL,
   cipherValueLen            INTEGER OPTIONAL
}
```

Where,

cipherValue: For ENCRYPT only, an octet string that contains the encrypted data that is returned by the CSFPPKV callable service.

cipherValueLen: For ENCRYPT only, an octet string that contains the length, in bytes, of the encrypted data. On a successful call, this is the length of the returned cipherValue octet string. If the returned ICSF return code and reason code indicate the input cipherValueMaxLen is too small, the returned value of cipherValueLen indicates the size that is needed for cipherValueMaxLen.

Message digesting ASN.1 syntaxes

ICSF provides the CSFPOWH callable service, which provides message digesting and hashing support. The **RemoteCryptoPKCS#11** extended operation allows the CSFPOWH callable service to be available for remote invocation.

One-way hash, sign, or verify (CSFPOWH) ASN.1 syntaxes

```
OWHInput ::= SEQUENCE {
   text                      OCTET STRING,
   chainData                 OCTET STRING,
   hash                      OCTET STRING
}
```

Where,

text: An octet string that identifies the input data that is being used for creating a hash in the CSFPOWH callable service.

chainData: An octet string that specifies the chaining data that is maintained during multipart hashing in the CSFPOWH callable service.

hash: An octet string that identifies the intermediate hash value in the CSFPOWH callable service.

```
OWHOutput ::= SEQUENCE {
   chainData                OCTET STRING,
   hash                     OCTET STRING.
   hashLen                  INTEGER
}
```

Where,

chainData: An optional octet string that contains the updated chaining data that is returned during multipart hashing that must be specified on the subsequent call in the CSFPOWH callable service. This data must be sent on the subsequent call in OWHInput.

hash: An octet string that specifies the hash that is generated from the CSKPOWH callable service from the input text data.

hashLen: An integer that contains the length, in bytes, of the hashed data. On a successful call, this is the length of the returned hash octet string. If the returned ICSF return code and reason code indicate the length of the input hash is too small, the returned value of hashLen indicates the buffer size that is needed for hash.

Secret key encrypt and secret key decrypt ASN.1 syntaxes

ICSF supports different types of secret key encryption and decryption algorithms. The **RemoteCryptoPKCS#11** extended operation allows the CSFPSKE and CSPFSKD ICSF callable services to be available for remote invocation.

Note: Special ASN.1 syntax must be used to handle GCM and GCMIVGEN mode AES encryption and decryption.

Secret key decrypt (CSFPSKD) ASN.1 syntaxes

```
SKDInput ::= SEQUENCE {
   initialValue             [0] OCTET STRING,
   gcmData                  [1] GCMData OPTIONAL,
   chainData                [2] OCTET STRING,
   cipherText               [3] OCTET STRING,
   clearTextMaxLen          [4] INTEGER (0 .. MaxCSFPInteger)
}

GCMData ::= SEQUENCE {
   gcmAuthenticationData    OCTET STRING,
   gcmTagLen                INTEGER
}
```

Where,

initialValue: An octet string that identifies the initialization vector for the decrypt operation in the CSFPSKD callable service.

gcmAuthenticationData: An optional octet string that identifies additional data that is provided for GCM mode AES decryption in the CSFPSKD callable service.

gcmTagLen: An optional integer that identifies the number of bytes for the tag that is generated during GCM mode AES decryption in the CSFPSKD callable service.

chainData: An octet string that specifies the chaining data that is maintained during multi-part decryption in the CSFPSKD callable service.

cipherText: An octet string that identifies the data to be decrypted in the CSFPSKD callable service.

clearTextMaxLen: An integer that specifies the maximum number of decrypted bytes to return in the CSKPSKD callable service.

```
SKDOutput ::= SEQUENCE {
   chainData                 OCTET STRING,
   clearText                 OCTET STRING,
   clearTextLength           INTEGER
}
```

Where,

chainData: An octet string that specifies the chaining data that is returned during multipart decryption in the CSFPSKD callable service. This data must be sent on subsequent SKDInput.

clearText: An octet string that returns the decrypted cipherText data that is returned from the CSFPSKD callable service.

clearTextLength: An integer that specifies the returned length of the clear text that is generated by the CSFPSKD callable service. On a successful call, this is the length of the returned clearText octet string. If the returned ICSF return code and reason code indicates the input clearTextMaxLen is too small, the returned value of clearTextLength indicates the size that is needed for the input clearTextMaxLen.

Secret key encrypt (CSFPSKE) ASN.1 syntaxes

```
SKEInput ::= SEQUENCE {
   initialValueChoice                InitialValue,
   chainData                         OCTET STRING,
   clearText                         OCTET STRING,
   cipherTextMaxLen                  INTEGER (0 .. MaxCSFPInteger)
}

InitialValue ::=  CHOICE {
   basicIV                           [0] OCTET STRING,
   gcmIV                             [1] GCMData,
   gcmIVGen                          [2] GCMIvGenData
}

GCMData ::= SEQUENCE {
   initialValue                      OCTET STRING,
   gcmAuthenticationData             OCTET STRING,
   gcmTagLen                         INTEGER
}

GCMIvGenData ::= SEQUENCE {
   4ByteNonce                        OCTET STRING,
   gcmAuthenticationData             OCTET STRING,
   gcmTagLen                         INTEGER,
   authenticationDataOffset          INTEGER OPTIONAL
}
```

Where,

initialValueChoice: Indicates the various options for specifying the initialization vector. GCM and GCMIVGEN modes of AES encryption must use special processing for the initialization vector. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information.

basicIV (initialization vector): An octet string that identifies the initialization vector for the encrypt operation in the CSFPSKE callable service.

initialValue: An octet string that identifies the initialization vector for the encrypt operation in the CSFPSKE callable service. For GCM mode AES encryption, this value requires special processing.

gcmAuthenticationData: An octet string that identifies additional authentication data that is needed for GCM and GCMIVGEN mode AES encryption in the CSFPSKE callable service.

gcmTagLen: An integer that identifies the number of bytes for the tag that is generated during GCM and GCMIVGEN mode AES encryption in the CSFPSKE callable service.

4ByteNonce: A 4-byte string that provides the nonce value used for initialization vector generation during GCMIVGEN mode AES encryption in the CSFPSKE callable service.

authenticationDataOffset: An optional integer that indicates the CSFPSKE callable service generates an initialization vector during GCMIVGEN mode AES encryption. The generated initialization vector must be stored within the gcmAuthenticationData octet string, beginning at the offset of this value.

chainData: An octet string that specifies the chaining data that is maintained during multipart encryption in the CSFPSKE callable service.

clearText: An octet string that specifies the clear text data that is being encrypted in the CSFPSKE callable service.

cipherTextMaxLen: An integer that specifies the maximum number of encrypted bytes to return in the CSFPSKE callable service.

Where,

```
SKEOutput ::= SEQUENCE {
   chainData                          OCTET STRING,
   cipherText                         OCTET STRING,
   cipherTextLength                   INTEGER,
   initialValue                       OCTET STRING,
   gcmAuthenticationData              OCTET STRING OPTIONAL
}
```

Where,

chainData: An octet string that specifies the chaining data that is returned during multipart encryption in the CSFPSKE callable service. This data must be sent on subsequent SKEInput.

cipherText: An octet string that specifies the encrypted clearText data that is returned from the CSFPSKE callable service.

`cipherTextLength`: An integer that specifies the returned length of the cipher text that is generated by the CSFPSKE callable service. On a successful call, this is the length of the returned `cipherText` octet string. If the returned ICSF return code and reason code indicates the input `cipherTextMaxLen` is too small, the returned value of `cipherTextLength` indicates the size that is needed for the input `cipherTextMaxLen`.

`initialValue`: An octet string that returns the initialization vector that is used for the encrypt operation in the CSFPSKE callable service. For GCMIVGEN mode AES encryption, this value returns the generated initialization vector.

`gcmAuthenticationData`: An octet string that returns the additional data that is used for GCM and GCMIVGEN mode AES encryption in the CSFPSKE callable service. If the `authenticationDataOffset` value for GCMIVGEN mode AES encryption is specified, this value is updated with the generated initialization vector.

CSFPSKD and CSFPSKE rule array reference

The CSFPSKE and CSFPSKD ICSF callable services support different algorithms and modes of encryption and decryption. The rule array sets the algorithm, mode, and multipart state of a CSFPSKE and CSFPSKD callable service invocation. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the different algorithms and modes that are supported by CSFPSKE and CSFPSKD.

Table 36. CSFPSKD and CSFPSKE supported mechanisms and rule arrays. CSFPSKD and CSFPSKE supported mechanisms and rule arrays

PKCS #11 mechanism	ICSF required values

CKM_CDMF_ECB
CKM_DES_ECB
	`rule_array = "DES ECB "`
	`initialization_vector_length = DES_BLOCK_SIZE=8`
	`key_handle`

CKM_CDMF_CBC
CKM_DES_CBC
	`rule_array ="DES CBC|CBC-PAD INITIAL|CONTINUE|FINAL|ONLY"`
	`initialization_vector_length = DES_BLOCK_SIZE=8`
	`key_handle`
	`chain_data`

CKM_DES_CBC_PAD
CKM_CDMF_CBC_PAD
	`rule_array ="DES CBC|CBC-PAD INITIAL|CONTINUE|FINAL|ONLY"`
	`initialization_vector_length = DES_BLOCK_SIZE=8`
	`key_handle`
	`chain_data`

CEKM_DS3_ECB	`rule_array ="DES3 ECB "`
	`initialization_vector_length = DES_BLOCK_SIZE=8`
	`key_handle`

CKM_DES3_CBC	`rule_array ="DES3 CBC|CBC-PAD INITIAL|CONTINUE|FINAL|ONLY"`
	`initialization_vector_length = DES_BLOCK_SIZE=8`
	`key_handle`
	`chain_data`

CKM_DES3_CBC_PAD	`rule_array ="DES3 CBC|CBC-PAD INITIAL|CONTINUE|FINAL|ONLY"`
	`initialization_vector_length = DES_BLOCK_SIZE=8`
	`key_handle`
	`chain_data`

CKM_RSA_PKCS	`rule_array ="RSA-PKCSENCRYPT "` `key_handle`
CKM_RSA_X_509	`rule_array ="RSA-ZEROENCRYPT "` `key_handle`

Table 36. CSFPSKD and CSFPSKE supported mechanisms and rule arrays (continued). CSFPSKD and CSFPSKE supported mechanisms and rule arrays

PKCS #11 mechanism	ICSF required values
CKM_AES_CBC	rule_array ="AES CBC|CBC-PAD INITIAL|CONTINUE|FINAL|ONLY" initialization_vector_length = DES_BLOCK_SIZE=8 key_handle chain_data
CKM_AES_ECB	rule_array ="AES CBC|CBC-PAD INITIAL|CONTINUE|FINAL|ONLY" initialization_vector_length = DES_BLOCK_SIZE=8 key_handle chain_data
CKM_AES_CBC_PAD	rule_array ="AES CBC|CBC-PAD INITIAL|CONTINUE|FINAL|ONLY" initialization_vector_length = DES_BLOCK_SIZE=8 key_handle chain_data
CKM_AES_GCM	rule_array = "AES GCM INITIAL|ONLY" initialization_vector_length key_handle chain_data
CKM_BLOWFISH_CBC	rule_array ="BLOWFISHCBC INITIAL|CONTINUE|FINAL|ONLY" initialization_vector_length initialization_vector key_handle chain_data
CKM_RC4	rule_array ="RC4 STREAM INITIAL|CONTINUE|FINAL|ONLY" key_handle chain_data

Key management ASN.1 syntaxes

ICSF supports different types of key management routines for creating secret, public, and private keys and for deriving single and multiple keys. The **RemoteCryptoPKCS#11** extended operation allows the CSFPGSK, CSFPGKP, CSFPDVK, CSFPDMK, CSFPWPK, and CSFPUWK ICSF callable services to be available for remote invocation.

Derive multiple keys (CSFPDMK) ASN.1 syntaxes

```
DMKInput ::= SEQUENCE {
   attrList             Attributes,
   parmsListChoice      DMKInputParmsList
}

DMKInputParmsList ::= CHOICE {
   SSL-KM_TLS-KM       [0] SSL_TLS_DMKInputParmsList
}

SSL_TLS_DMKInputParmsList ::= SEQUENCE {
   export              BOOLEAN,
   macSize             INTEGER,
   keySize             INTEGER,
   ivSize              INTEGER,
   clientRandomData    OCTET STRING,
   serverRandomData    OCTET STRING
}
```

Where,

attrList: An attribute list that identifies characteristics (attribute name and value pairs) that are used for deriving or creating multiple new keys in the CSFPDMK callable service.

parmsListChoice: A choice sequence that specifies portions of the parms_list for the CSFPDMK callable service. Only one parms_list format is supported by ICSF and it is for deriving keys for the SSL-KM and TLS-KM mechanisms. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the parms_list format for the CSFPDMK callable service.

export: A boolean that indicates that export processing must be used in the CSFPDMK callable service.

macSize: An integer indicating the size of the MAC that is being generated in bits where 8 <= size <= 384, in multiples of 8.

keySize: An integer indicating the size of key that is being generated in bits. Must match a supported size for the key type that is specified in the attribute list.

ivSize (initialization vector): An integer indicating the size of IV that is being generated in bits (v), where 0 <= size <= 128, in multiples of 8.

clientRandomData: An octet string that specifies the random data of the client in the parms_list for the CSFPDMK callable service.

serverRandomData: An octet string that specifies the random data of the server in the parms_list for the CSFPDMK callable service.

```
DMKOutput ::= SEQUENCE {
   parmsListChoice          DMKOutputParmsList
}

DMKOutputParmsList ::= CHOICE {
   SSL-KM_TLS-KM        [0] SSL_TLS_DMKOutputParmsList
}

SSL_TLS_DMKOutputParmsList ::= SEQUENCE {
   clientMACHandle      OCTET STRING,
   serverMACHandle      OCTET STRING,
   clientKeyHandle      OCTET STRING,
   serverKeyHandle      OCTET STRING,
   clientIV             OCTET STRING
}
```

Where,

parmsListChoice: A choice sequence that specifies portions of the parms_list for the CSFPDMK callable service. Only one parms_list format is supported by ICSF and it is for deriving keys for the SSL-KM and TLS-KM mechanisms. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the parms_list format for the CSFPDMK callable service.

clientMACHandle: An octet string that identifies the handle of the client MAC secret object that is generated by the CSFPDMK callable service. If the client MAC secret object is not successfully generated, this octet string consists of 44 bytes of x'00' data.

serverMACHandle: An octet string that identifies the handle of the server MAC secret object that is generated by the CSFPDMK callable service. If the server MAC secret object is not successfully generated, this octet string consists of 44 bytes of x'00' data.

clientKeyHandle: An octet string that identifies the handle of the client key object that is generated by the CSFPDMK callable service. If the client key object is not successfully generated, this octet string consists of 44 bytes of x'00' data.

serverKeyHandle: An octet string that identifies the handle of the server key object that is generated by the CSFPDMK callable service. If the server key object is not successfully generated, this octet string consists of 44 bytes of x'00' data.

clientIV (initialization vector): An octet string that identifies the IV of the client that is generated by the CSFPDMK callable service.

serverIV (initialization vector): An octet string that identifies the IV of the server that is generated by the CSFPDMK callable service.

Derive key (CSFPDVK) ASN.1 syntaxes

```
DVKInput ::= SEQUENCE {
   attrList                 Attributes,
   parmsListChoice          DVKInputParmsList
}

DVKInputParmsList ::= CHOICE {
   PKCS-DH_publicValue      [0] OCTET STRING,
   SSL-TLS                  [1] SSL-TLS_DVKInputParmsList,
   EC-DH                    [2] EC-DH_DVKInputParmsList
}

SSL_TLS_DVKInputParmsList ::= SEQUENCE {
   clientRandomData         OCTET STRING,
   serverRandomData         OCTET STRING
}

EC-DH_DVKInputParmsList ::= SEQUENCE {
   kdfCode                  OCTET STRING,
   sharedData               OCTET STRING,
   EC-DH_publicValue        OCTET STRING
}
```

Where,

attrList: An attribute list that identifies characteristics (attribute name and value pairs) that are used for deriving or creating a key in the CSFPDVK callable service.

parmsListChoice: A choice sequence that specifies portions of the parms_list for the CSFPDVK callable service. The sequence that is chosen depends on the type of key being derived. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the supported parms_list for CSFPDVK.

PKCS-DH_publicValue: An octet string specifying the binary value representing the public value of the other party when deriving a PKCS-DH key in the CSFPDVK callable service.

clientRandomData: An octet string specifying the random data of the client when deriving an SSL-MS, SSL-MSDH, TLS-MS, or TLS-MSDH key in the CSFPDVK callable service.

serverRandomData: An octet string specifying the random data of the server when deriving an SSL-MS, SSL-MSDH, TLS-MS, or TLS-MSDH key in the CSFPDVK callable service.

kdfCode: An octet string specifying the KDF function code that is used when deriving an EC-DH key in the CSFPDVK callable service.

sharedData: An octet string that identifies shared data that is needed for deriving an EC-DH key in the CSFPDVK callable service.

EC-DH_publicValue: An octet string that specifies the binary value representing the public value of the other party with or without DER encoding in the CSFPDVK callable service.

```
DVKOutput ::= SEQUENCE {
   parmsListChoice           DVKOutputParmsList
}

DVKOutputParmsList ::= CHOICE {
   PKCS-DH_Output            [0] NULL,
   SSL-TLS_Output            [1] OCTET STRING,
   EC-DH_Output              [2] NULL
}
```

Where,

parmsListChoice: A choice sequence that specifies portions of the parms_list for the CSFPDVK callable service. The sequence that is chosen depends on the type of key being derived. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* or more information about the supported parms_list for CSFPDVK.

SSL-TLS_Output: An octet string that represents the SSL or TLS protocol version of the derived SSL-MS or TLS-MS key. It is a hexadecimal representation of the protocol version. For example, 0301' for version 3.1. This value is set to 0000' when an SSL-MSDH or TLS-MSDH key is derived.

Generate key pair (CSFPGKP) ASN.1 syntaxes

```
GKPInput ::= SEQUENCE {
   publicKeyAttrList         Attributes,
   privateKeyAttrList        Attributes
}
```

Where,

publicKeyAttrList: An attribute list that identifies characteristics (attribute name and value pairs) that are used for creating the public key in the CSFPGKP callable service.

privateKeyAttrList: An attribute list that identifies characteristics (attribute name and value pairs) that are used for creating the private key in the CSFPGKP callable service.

```
GKPOutput ::= privateHandle
privateHandle ::= OCTET STRING
```

Where,

privateHandle: An octet string that identifies the handle of the private key that is generated by the CSFPGKP callable service. The generated public handle is

returned in the handle parameter of the responseValue encoding. If the callable service returns an error, privateHandle is encoded as a zero length octet string.

Generate secret key (CSFPGSK) ASN.1 syntaxes

```
GSKInput ::= SEQUENCE {
   attrList            Attributes,
   parmsList           OCTET STRING
}
```

Where,

attrList: An attribute list that identifies characteristics (attribute name and value pairs) that are used for creating the secret key in the CSFPGSK callable service. If an attrList is not needed for generating the secret key, specify a zero length sequence in the Attributes.

parmsList: An octet string that specifies the parameter list for the CSFPGSK callable service. If an parmsList is not needed for generating the secret key, specify a zero length octet string. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the parmsList format.

```
GSKOuput ::= NULL
```

Unwrap key (CSFPUWK) ASN.1 syntaxes

```
UWKInput ::= SEQUENCE {
   wrappedKey           OCTET STRING,
   initialValue         OCTET STRING,
   attrList             Attributes
}
```

Where,

wrappedKey: An octet string that contains key data that is to be unwrapped in the CSFPUWK callable service.

initialValue: An octet string that identifies the initialization vector for the unwrapping of the key in the CSFPUWK callable service.

attrList: An attribute list that identifies characteristics (attribute name and value pairs) that are used for creating a key after unwrapping the key data in the CSFPUWK callable service.

```
UWKOutput ::= NULL
```

Wrapped key (CSFPWPK) ASN.1 syntaxes

```
WPKInput ::= SEQUENCE {
   wrappingHandle              OCTET STRING,
   wrappedKeyMaxLen            INTEGER (0 .. MaxCSFPInteger),
   initialValue                OCTET STRING

}
```

Where,

wrappingHandle: An octet string that identifies the name of a public key or secret key object to wrap a secret key or the wrapping_key_handle parameter in the CSFPWPK callable service. The secret key that is wrapped is specified in the handle parameter of the requestValue encoding.

`wrappedKeyMaxLen`: An integer that specifies the maximum expected length of the wrapped key output that is generated by the CSFPWPK callable service.

`initialValue`: An octet string that identifies the initialization vector for the wrapping of the key in the CSFPWPK callable service. If an initialization vector is not needed, specify a zero length octet string. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the `initialValue` format.

```
WPKOutput ::= SEQUENCE {
   wrappedKey            OCTET STRING,
   wrappedKeyLen         INTEGER
}
```

Where,

`wrappedKey`: An octet string that specifies the returned wrapped or encrypted key data from the CSFPWPK callable service. If the callable service returns an error, this field is encoded as a zero length octet string.

`wrappedKeyLen`: An integer that specifies the returned length of the wrapped key that is generated by the CSPWPK callable service. On a successful call, this is the length of the returned `wrappedKey` octet string. If the returned ICSF return code and reason code indicates the input `wrappedKeyMaxLen` is too small, the returned value of `wrappedKeyLen` indicates the size that is needed for the input `wrappedKeyMaxLen`.

Common RemoteCryptoPKCS#11 extended operation error codes

Table 37 summarizes some different error scenarios and the **RemoteCryptoPKCS#11** extended operation response that is returned for such scenarios. ICSF callable service-specific return codes and reason codes are returned in the response. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about those errors

Table 37. Common RemoteCryptoPKCS#11 extended operation return codes. Common RemoteCryptoPKCS#11 extended operation return codes

Error scenario	RemoteCryptoPKCS#11 response
Out of memory	Returns an **LDAP_OTHER** return code
Internal server error	Returns an **LDAP_OPERATIONS_ERROR** return code
Unable to decode request	Returns an **LDAP_PROTOCOL_ERROR** return code
Data sent on request is not valid, for example, a -1 for a length field	Returns an **LDAP_PROTOCOL_ERROR** return code
Unable to encode response	Returns an **LDAP_OTHER** return code

ICSF callable services supported by the RemoteCryptoCCA extended operation

The **RemoteCryptoCCA** extended operation is a generic extended operation that allows an LDAP client application to specify the same data as if invoking the ICSF callable service locally. Table 38 on page 144 includes the ICSF CCA callable service routines that are supported by the **RemoteCryptoCCA** extended operation. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* or *Cryptographic Token Interface Standard* for more information about ICSF callable services.

Table 38. ICSF callable services supported by the RemoteCryptoCCA extended operation. ICSF callable services supported by the RemoteCryptoCCA extended operation

ICSF callable service and helper function	Description	Function code	Request tag	Response tag
CSNECKM - Multiple clear key import	Imports a single-length, double-length, or triple-length clear DATA key, enciphers it under the master key, and places the result into an internal key token. CSNBCKM converts the clear key into operational form as a DATA key.	FC_CSNECKM	REQ_CSNECKM	RES_CSNECKM
CSNEDEC - Decipher	Deciphers data using either the CDMF or the cipher block chaining mode of the DES. (The method depends on the token marking or keyword specification.)	FC_CSNEDEC	REQ_CSNEDEC	RES_CSNEDEC
CSNEENC - Encipher	Enciphers data using either the CDMF or the cipher block chaining mode of the DES. (The method depends on the token marking or keyword specification.)	FC_CSNEENC	REQ_CSNEENC	RES_CSNEENC
CSNEKGN - Key generate	Generates a 64-bit, 128-bit, or 192-bit odd parity key, or a pair of keys, and returns them in encrypted forms (operational, exportable, or importable).	FC_CSNEKGN	REQ_CSNEKGN	RES_CSNEKGN
CSNEKRC - CKDS key record create	Adds a key record containing a key token set to binary zeros to both the in-storage and DASD copies of the CKDS.	FC_CSNEKRC	REQ_CSNEKRC	RES_CSNEKRC
CSNEKRD - CKDS key record delete	Deletes a key record from both the in-storage and DASD copies of the CKDS.	FC_CSNEKRD	REQ_CSNEKRD	RES_CSNEKRD
CSNEKRR - CKDS key record read	Use the CKDS key record read callable service to read an internal AES or DES key token.	FC_CSNEKRR	REQ_CSNEKRR	RES_CSNEKRR
CSNEKRW - CKDS key record write	Writes an internal key token to the CKDS record specified in the key label parameter. Updates both the in-storage and DASD copies of the CKDS currently in use.	FC_CSNEKRW	REQ_CSNEKRW	RES_CSNEKRW
CSNEKTB - Key token build	Use the key token build callable service to build an external or internal key token from information which you supply.	FC_CSNEKTB	REQ_CSNEKTB	RES_CSNEKTB
CSNESAD - Symmetric algorithm decipher	Deciphers data using the AES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes.	FC_CSNESAD	REQ_CSNESAD	RES_CSNESAD

Table 38. ICSF callable services supported by the RemoteCryptoCCA extended operation (continued). ICSF callable services supported by the RemoteCryptoCCA extended operation

ICSF callable service and helper function	Description	Function code	Request tag	Response tag
CSNESAE - Symmetric algorithm encipher	Enciphers data using the AES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes.	FC_CSNESAE	REQ_CSNESAE	RES_CSNESAE
CSNESYD - Symmetric key decipher	Deciphers data using the AES or DES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes.	FC_CSNESYD	REQ_CSNESYD	RES_CSNESYD
CSNESYE - Symmetric key encipher	Enciphers data using the AES or DES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes.	FC_CSNESYE	REQ_CSNESYE	RES_CSNESYE
CSNFKRC - PKDS key record create	Writes a new record to the PKDS.	FC_CSNFKRC	REQ_CSNFKRC	RES_CSNFKRC
CSNFKRD - PKDS key record delete	Deletes a record from the PKDS.	FC_CSNFKRD	REQ_CSNFKRD	RES_CSNFKRD
CSNFKRR - PKDS key record read	Use the PKDS key record read callable service to read a record from the PKDS.	FC_CSNFKRR	REQ_CSNFKRR	RES_CSNFKRR
CSNFPKB - PKA key token build	Creates an external PKA key token containing a clear private RSA or DSS key. Using this token as input to the PKA key import callable service returns an operational internal token containing an enciphered private key. Using CSNDPKB on a clear public RSA or DSS key, returns the public key in a token format that other PKA services can directly use. CSNDPKB can also be used to create a skeleton token for input to the PKA key generate service for the generation of an internal DSS or RSA key token.	FC_CSNFPKB	REQ_CSNFPKB	RES_CSNFPKB
CSNFPKG - PKA key generate	Generates a DSS internal token for use in digital signature services, RSA keys (for use on the PCICC, PCIXCC, CEX2C, or CEX3C) and ECC keys (for use on the CEX3C).	FC_CSNFPKG	REQ_CSNFPKG	RES_CSNFPKG
CSNFPKI - PKA key import	Imports a PKA key token containing either a clear PKA key or a PKA key enciphered under a limited authority IMP-PKA KEK.	FC_CSNFPKI	REQ_CSNFPKI	RES_CSNFPKI
CSNFPKX - PKA public key extract	Extracts a PKA public key token from a supplied PKA internal or external private key token.	FC_CSNFPKX	REQ_CSNFPKX	RES_CSNFPKX

Table 38. ICSF callable services supported by the RemoteCryptoCCA extended operation (continued). ICSF callable services supported by the RemoteCryptoCCA extended operation

ICSF callable service and helper function	Description	Function code	Request tag	Response tag
CSNFSYI - Symmetric key import	Imports a symmetric key enciphered under an RSA public key into operational form enciphered under a host master key.	FC_CSNFSYI	REQ_CSNFSYI	RES_CSNFSYI
CSNFSYX - Symmetric key export	Transfers an application-supplied symmetric key from encryption under the host master key to encryption under an application-supplied RSA public key or AES EXPORTER key.	FC_CSNFSYX	REQ_CSNFSYX	RES_CSNFSYX

Because the **RemoteCryptoCCA** extended operation is a generic extended operation, it supports all of the ICSF CCA callable services in Table 38 on page 144. Therefore, the extended operation has a common BER encoding for both the request and response values as many of the ICSF CCA callable service routines have common interfaces (inputs and outputs on the callable service routines).

The request and response values for the **RemoteCryptoCCA** extended operation follows. For more information about ASN.1 (Abstract Syntax Notation One) and BER (Basic Encoding Rules), see:

`ftp://ftp.rsa.com/pub/pkcs/ascii/layman.asc`

- **Request values:** The following ASN.1 syntax describes the BER encoding of the request value for the **RemoteCryptoCCA** extended operation request. Each ICSF CCA callable service has additional callable service-specific information that is encoded in the `requestData` field. The request OID is 1.3.18.0.2.12.85.

```
requestValue ::= SEQUENCE {
    version         INTEGER,
    exitData        OCTET STRING,
    ruleArraySeq    RuleArraySeq,
    requestData     CCAInput
}
```

Where,

`version`: Identifies which version of the interface is being used. Currently the only value supported is 1. If the interface is extended in the future, then other values are supported.

`exitData`: Identifies the data that is passed to the installation exit. See the appropriate ICSF callable service routine in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information. If the ICSF callable service routine does not need `exitData`, then specify a zero length octet string.

`ruleArraySeq`: Is a sequence that identifies an array of keywords that provide control information to the ICSF callable services. The ASN.1 description is:

```
ASCII STRING ::= OCTET STRING
```

Note: `ASCII STRING` - Represents an OCTET STRING of ASCII bytes.

```
RuleArraySeq ::= SEQUENCE {
   ruleArrayCount     INTEGER,
   ruleArray          ASCII STRING
}
```

See the appropriate ICSF callable service routine in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the rule array. The integer `ruleArrayCount` identifies the number of rules in the array and is referred to as `rule_array_count` in the ICSF callable service routine descriptions. The octet string `ruleArray` is the actual array of rules, and is referred to as `rule_array` in the ICSF callable service routine descriptions. The number `ruleArrayCount` must be consistent with the contents of the octet string `rule_array`. The length of `ruleArray` octet string must be a multiple of 8 times the number that is specified in the `ruleArrayCount` field. If an individual rule specified in the `ruleArray` is fewer than 8 characters, it must be blank padded on the right to 8 full characters. You must always use the entire sequence. To specify an empty rule array, specify `ruleArrayCount` as zero, and `ruleArray` as a zero length octet string.

`requestData`: Identifies the extended operation request-specific data for the ICSF callable service. See Table 38 on page 144 for the appropriate request tag for `CCAInput`. The values for these tags can be found in the ldap_rcrypto.h sample header file.

Where,

```
CCAInput ::= CHOICE {
        ECKM    [CSNECKM]          ECKMInput,
        EDEC    [CSNEDEC]          EDECInput,
        EENC    [CSNEENC]          EENCInput,
        EKGN    [CSNEKGN]          EKGNInput,
        EKRC    [CSNEKRC]          EKRCInput,
        EKRD    [CSNEKRD]          EKRDInput,
        EKRR    [CSNEKRR]          EKRRInput,
        EKRW    [CSNEKRW]          EKRWInput,
        EKTB    [CSNEKTB]          EKTBInput,
        ESAD    [CSNESAD]          ESADInput,
        ESAE    [CSNESAE]          ESAEInput,
        ESYD    [CSNESYD]          ESYDInput,
        ESYE    [CSNESYE]          ESYEInput,
        FKRC    [CSNFKRC]          FKRCInput,
        FKRD    [CSNFKRD]          FKRDInput,
        FKRR    [CSNFKRR]          FKRRInput,
        FPKB    [CSNFPKB]          FPKBInput,
        FPKG    [CSNFPKG]          FPKGInput,
        FPKI    [CSNFPKI]          FPKIInput,
        FPKX    [CSNFPKX)          FPKXInput,
        FSYI    [CSNFSYI]          FSYIInput
        FSYX    [CSNFSYX]          FSYXInput
}
```

- **Response values:** The following ASN.1 syntax describes the BER encoding of the response value for the **RemoteCryptoCCA** extended operation response. The ICSF return code and ICSF reason codes are returned from the underlying ICSF callable service routine. Each ICSF CCA callable service returns additional response-specific data in the `responseData` field. The response OID is 1.3.18.0.2.12.86.

```
responseValue ::= SEQUENCE {
   version        INTEGER,
   ICSFRc         INTEGER (0 .. MaxCCAInteger),
   ICSFRsnCode    INTEGER (0 .. MaxCCAInteger),
   exitData       OCTET STRING,
   responseData   CCAOutput
}
```

Where,

`version`: Identifies which version of the interface is being used. Currently the only value supported is 1. If the interface is extended in the future, then other values are supported.

`ICSFRc`: Identifies the ICSF return code. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the ICSF return code error.

`ICSFRsnCode`: Identifies the ICSF reason code. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the ICSF reason code error.

`exitData`: Identifies the data that is passed to the installation exit. See the appropriate ICSF callable service routine in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information. A zero length octet string indicates that no `exitData` is returned.

`responseData`: Identifies the ICSF CCA request-specific extended operation response data. See Table 38 on page 144 for the appropriate response tag for `CCAOutput`. The values for these tags can be found in the ldap_rcypto.h sample header file.

Where,

```
CCAOutput ::= CHOICE {
    ECKM    [CSNECKM]          ECKMOutput,
        EDEC    [CSNEDEC]          EDECOutput,
        EENC    [CSNEENC]          EENCOutput,
        EKGN    [CSNEKGN]          EKGNOutput,
        EKRC    [CSNEKRC]          EKRCOutput,
        EKRD    [CSNEKRD]          EKRDOutput,
        EKRR    [CSNEKRR]          EKRROutput,
        EKRW    [CSNEKRW]          EKRWOutput,
        EKTB    [CSNEKTB]          EKTBOutput,
        ESAD    [CSNESAD]          ESADOutput,
        ESAE    [CSNESAE]          ESAEOutput,
        ESYD    [CSNESYD]          ESYDOutput,
        ESYE    [CSNESYE]          ESYEOutput,
        FKRC    [CSNFKRC]          FKRCOutput,
        FKRD    [CSNFKRD]          FKRDOutput,
        FKRR    [CSNFKRR]          FKRROutput,
        FPKB    [CSNFPKB]          FPKBOutput,
        FPKG    [CSNFPKG]          FPKGOutput,
        FPKI    [CSNFPKI]          FPKIOutput,
        FPKX    [CSNFPKX)          FPKXOutput,
        FSYI    [CSNFSYI]          FSYIOutput,
        FSYX    [CSNFSYX]          FSYXOutput
}
MaxCCAInteger ::= 2^31 - 1
keyIdentifier ::= OCTET STRING
```

Note: `keyIdentifier`: If the first byte of a key identifier is larger than 20, the LDAP server assumes that it is an ASCII key label. Otherwise, it assumes that it is a binary key token.

Common ASN.1 encodings used by the RemoteCryptoCCA extended operation

This section describes the common ASN.1 encodings that are used by the **RemoteCryptoCCA** extended operation request and response.

Symmetric key management ASN.1 syntaxes

ICSF supports a symmetric key management routine for importing a clear AES or DES key, enciphering the key under the corresponding master key and then returning the enciphered key in an internal key token. The **RemoteCryptoCCA** extended operation allows the CSNECKM ICSF callable service to be available for remote invocation.

Multiple clear key import (CSNECKM) ASN.1 syntaxes

```
ECKMInput ::= SEQUENCE {
   clearKey               OCTET STRING,
   keyIdentifier          keyIdentifier
}
```

Where,

`clearKey`: Specifies the clear key value to import in the CSNECKM callable service. The length of the value must be 8-bytes, 16-bytes, or 24-bytes for DES keys and 16-bytes, 24-bytes, or 32-bytes for AES keys.

`keyIdentifier`: A 64-byte string that is used to receive an internal AES or DES key token in the CSNECKM callable service.

```
ECKMOutput ::= SEQUENCE {
   keyIdentifier          keyIdentifier,
   keyIdLength            INTEGER
}
```

Where,

`keyIdentifier`: A 64-byte string containing an internal AES or DES key token where the enciphered key is located.

`keyIdLength`: The length in bytes of the output `keyIdentifier` parameter.

CKDS key record management ASN.1 syntaxes

ICSF supports different types of CKDS key management routines for creating and deleting a key record in the CKDS, writing a key token to the CKDS record, and generating DES keys in the CKDS. The **RemoteCryptoCCA** extended operation allows the CSNEKRC, CSNEKRD, CSNEKGN, CSNEKRR, CSNEKRW, and CSNEKTB ICSF callable services to be available for remote invocation.

CKDS key record create (CSNEKRC) ASN.1 syntaxes

```
EKRCInput ::= keyLabel
   keyLabel ::= ASCII STRING
```

Where,

`keyLabel`: The 64-byte label of the record that is added to the CKDS, and used to store AES and DES tokens in the CSNEKRC callable service.

```
EKRCOutput ::= NULL {
```

CKDS key record delete (CSNEKRD) ASN.1 syntaxes

```
EKRDInput ::= keyLabel
keyLabel ::= ASCII STRING
```

Where,

keyLabel: The 64-byte label of the key record containing an AES or DES token in CKDS that is deleted in the CSNEKRD callable service.

```
EKRDOutput ::= NULL {
```

CKDS key record read (CSNEKRR) ASN.1 syntaxes

```
EKRRInput ::= keyLabel
keyLabel ::= ASCII STRING
```

Where,

keyLabel: The 64-byte label of a record containing an AES or DES token in the in-storage CKDS. The internal key token in this record is returned to the caller.

```
EKRROutput ::= keyToken
keyToken ::= OCTET STRING
```

Where,

keyToken: The 64-byte internal key token that is retrieved from the in-storage CKDS.

CKDS key record write (CSNEKRW) ASN.1 syntaxes

```
EKRWInput ::= SEQUENCE {
    keyToken              OCTET STRING,
    keyLabel              ASCII STRING
}
```

Where,

keyToken: The 64-byte internal AES or DES key token that is written to the CKDS in the CSNEKRW callable service.

keyLabel: The 64-byte label of the record in the CKDS that key token is written to in the CSNEKRW callable service.

```
EKRWOutput ::= keyToken
keyToken ::= OCTET STRING
```

Where,

keyToken: The 64-byte internal AES or DES key token that is written to the CKDS in the CSNEKRW callable service.

CKDS key generate (CSNEKGN) ASN.1 syntaxes

```
EKGNInput ::= SEQUENCE {
       keyForm                ASCII STRING,
       keyLength              ASCII STRING,
       keyType1               ASCII STRING,
       keyType2               ASCII STRING,
       kekKeyIdentifer1       KeyIdentifier,
       kekKeyIdentifer2       KeyIdentifier,
       generatedKeyIdentifer1 KeyIdentifier,
       generatedKeyIdentifer2 KeyIdentifier,
}
```

Where,

keyForm: A 4-byte keyword that defines the type of key or keys you want to generate.

keyLength: An 8-byte string value that defines the length of the key.

keyType1: An 8-byte value that defines the type of generatedKeyIdentifer1.

keyType2: An 8-byte value that defines the type of generatedKeyIdentifer2. This can only be used if DES keys are being generated.

kekKeyIdentifer1: A 64-byte string of a DES internal key token that contains the importer or exporter key-encrypting key, or a key label.

kekKeyIdentifer2: A 64-byte string of a DES internal key token that contains the importer or exporter key-encrypting key, or a key label of an internal token.

generatedKeyIdentifer1: Specifies either a generated:

- Internal DES or AES key token for an operational key form, or
- External DES key tokens that contain a key that is enciphered by using KEK_key_identifier_1.

generatedKeyIdentifer2: Specifies a generated external key token that contains a key that is enciphered by using KEK_key_identifier_2.

CKDS Key token build (CSNEKTB) ASN.1 syntaxes

```
EKTBInput ::= SEQUENCE {
       keyToken               OCTET STRING,
       keyType                ASCII STRING,
       keyValue               OCTET STRING,
       mkKeyVersion           INTEGER,
       keyRegisterNumber      INTEGER,
       tokenData1             OCTET STRING,
       controlVector          OCTET STRING,
       initializationVector   OCTET STRING,
       padCharacter           INTEGER (0..255),
       crytpoPeriodStart      OCTET STRING,
       mkVerificationPattern  OCTET STRING
}
```

Where,

keyToken: A 64-byte string. If the keyType parameter is TOKEN, then this is a 64-byte internal token that is updated as specified in the rule_array. Otherwise, this field is an output-only field and must be 64 bytes long.

keyType: An 8-byte field that specifies the type of key you want to build or the keyword TOKEN for updating a supplied token.

keyValue: A 32-byte string representing the key value. See Key Token Build (CSNBKTB and CSNEKTB) in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about the content, and if needed, padding, requirements for this buffer.

mkKeyVersion: The master key version number. This field is examined only if the KEY keyword is specified, in which case, this field must be zero.

keyRegisterNumber: This value is ignored.

tokenData1: An 8-byte field containing the LRC value for AES keys.

controlVector: A 16-byte field containing the control vector. For AES keys, the 16 bytes are ignored.

initializationVector: This value is ignored.

padCharacter: Pad character.

crytpoPeriodStart: This value is ignored.

mkVerificationPattern: An 8-byte field containing the value that is inserted into the master key verification pattern field of the key token.

```
EKTBOutput ::= keyToken
keyToken ::= OCTET STRING
```

Where,

keyToken: The key token value.

Symmetric cryptography-related services

ICSF supports encipher and decipher callable services for protecting data or reading protected data. The **RemoteCryptoCCA** extended operation allows the CSNEDEC, CSNEENC, CSNESAD, CSNESAE, CSNESYD, and CSNESYE ICSF callable services to be available for remote invocation.

Symmetric algorithm encipher (CSNESAE) ASN.1 syntaxes

```
ESAEInput ::= SEQUENCE {
       keyIdentifier            KeyIdentifier,
       keyParms                 OCTET STRING,
       blockSize                INTEGER,
       initializationVector     OCTET STRING,
       chainData                OCTET STRING,
       clearText                OCTET STRING,
       cipherTextLength         INTEGER,
       optionalData             OCTET STRING,
}
```

Where,

keyIdentifier: Specifies an internal secure AES token or the label name of a secure AES token in the CKDS. Normal CKDS label name syntax is required.

keyParms: This value is ignored.

blockSize: Block size for the cryptographic algorithm.

initializationVector: Contains the initialization vector (IV) for CBC mode encryption, including the CBC mode invoked using the PKCS-PAD keyword.

chainData: A buffer that is used as a work area for sequences of chained symmetric algorithm encipher requests. When the keyword INITIAL is used, this is an output parameter and receives data that is needed when enciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output parameter; the value received as output from the previous call in the sequence is provided as input to this call, and this call returns new chainData that is used as input on the next call.

clearText: Text to be enciphered.

cipherTextLength: Specifies the maximum number of cipher bytes to return.

optionalData: This value is ignored.

```
ESAEOutput :: = SEQUENCE {
       chainData               OCTET STRING,
       cipherText              OCTET STRING,
       cipherTextLength        INTEGER,
       optionalData            OCTET STRING
}
```

Where,

chainData: Data that is needed when enciphering the next part of the input data.

cipherText: Contains the enciphered data.

cipherTextLength: Specifies the number of bytes returned in cipherText.

optionalData: Reserved.

Symmetric algorithm decipher (CSNESAD) ASN.1 syntaxes

```
ESADInput ::= SEQUENCE {
       keyIdentifier           keyIdentifier,
       keyParms                OCTET STRING,
       blockSize               INTEGER,
       initializationVector    OCTET STRING,
       chainData               OCTET STRING,
       cipherText              OCTET STRING,
       clearTextLength         INTEGER,
       optionalData            OCTET STRING,
}
```

Where,

keyIdentifier: Specifies an internal secure AES token or the label name of a secure AES token in the CKDS. Normal CKDS label name syntax is required.

keyParms: This value is ignored.

blockSize: Block size for the cryptographic algorithm.

initializationVector: Contains the initialization vector (IV) for CBC mode decryption, including the CBC mode that is invoked by using the PKCS-PAD keyword.

chainData: A buffer that is used as a work area for sequences of chained symmetric algorithm decipher requests. When the keyword INITIAL is used, this is an output parameter and receives data that is needed when deciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output parameter; the value received as output from the previous call in the sequence is provided as input to this call, and this call returns new chainData that is used as input on the next call.

cipherText: Text to be deciphered.

clearTextLength: Specifies the maximum number of clear bytes to return.

optionalData: This value is ignored.

```
ESADOutput :: = SEQUENCE {
       chainData                OCTET STRING,
       clearText                OCTET STRING,
       clearTextLength          INTEGER,
       optionalData             OCTET STRING
}
```

Where,

chainData: Data that is needed when deciphering the next part of the input data.

clearText: Contains the deciphered data.

clearTextLength: Specifies the number of bytes returned in clearText.

optionalData: Reserved.

Encipher (CSNEENC) ASN.1 syntaxes

```
EENCInput ::= SEQUENCE {
       keyIdentifier            KeyIdentifier,
       clearText                OCTET STRING,
       initializationVector     OCTET STRING,
       padCharacter             INTEGER (0..255),
       chainingVector           OCTET STRING,
}
```

Where,

keyIdentifier: A 64-byte string that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing the data-encrypting key, to be used for encrypting the data.

clearText: Text to be enciphered.

initializationVector: The 8-byte supplied string for the cipher block chaining.

padCharacter: An integer, 0 - 255, that is used as a padding character for the 4700-PAD process rule.

chainingVector: An 18-byte field that ICSF uses as a system work area.

```
EENCOutput :: = SEQUENCE {
       keyIdentifier            KeyIdentifier,
       chainingVector           OCTET STRING,
       cipherText               OCTET STRING,
       textLength               INTEGER
}
```

Where,

keyIdentifier: A 64-byte string that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing the data-encrypting key, to be used for encrypting the data.

chainingVector: An 18-byte buffer that is used as a work area for sequences of chained encipher requests. When the keyword INITIAL is used, this is an output parameter and receives data that is needed when enciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output

parameter; the value received as output from the previous call in the sequence is provided as input to this call, and this call returns new chainingVector that is used as input on the next call.

cipherText: Data to be enciphered.

textLength: On entry, you supply the length of the cleartext. See the MaxCCAInteger for the maximum text length value. A zero value for the text_length parameter is not valid. If the returned enciphered text (cipherText parameter) is a different length because of the addition of padding bytes, the value is updated to the length of the ciphertext.

Decipher (CSNEDEC) ASN.1 syntaxes

```
EDECInput ::= SEQUENCE {
    keyIdentifier               KeyIdentifier,
    cipherText                  OCTET STRING,
    initializationVector        OCTET STRING,
    chainingVector              OCTET STRING
}
```

Where,

keyIdentifier: A 64-byte string that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing a data-encypting key.

cipherText: Text to be deciphered.

initializationVector: The 8-byte supplied string for the cipher block chaining.

chainingVector: An 18-byte field that ICSF uses as a system work area.

```
EDECOutput :: = SEQUENCE {
    keyIdentifier               KeyIdentifier,
    chainingVector              OCTET STRING,
    clearText                   OCTET STRING,
    textLength                  INTEGER
}
```

Where,

keyIdentifier: A 64-byte string that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing a data-encypting key.

chainingVector: An 18-byte field that ICSF uses as a system work area.

clearText: Field where the callable service returns the deciphered text.

textLength: On entry, you supply the length of the ciphertext. See the MaxCCAInteger for the maximum length value. A zero value for the text_length parameter is not valid. If the returned deciphered text (clear_text parameter) is a different length because of the removal of padding bytes, the value is updated to the length of the plaintext.

Symmetric key encipher (CSNESYE) ASN.1 syntaxes

```
ESYEInput ::= SEQUENCE {
     keyIdentifier            KeyIdentifier,
     keyParms                 OCTET STRING,
```

```
     blockSize               INTEGER,
     initializationVector    OCTET STRING,
     chainData               OCTET STRING,
     clearText               OCTET STRING,
     cipherTextLength        INTEGER,
     optionalData            OCTET STRING
}
```

Where,

`keyIdentifier`: Specifies the cipher key, for the KEY-CLR keyword. The parameter must be left-aligned. Specifies an internal clear token, or the label name of a clear key or an encrypted key in the CKDS, for the KEYIDENT keyword. Normal CKDS label name syntax is required.

`keyParms`: Specifies the cipher key, for the KEY-CLR keyword. The parameter must be left-aligned. Specifies an internal clear token, or the label name of a clear key or an encrypted key in the CKDS, for the KEYIDENT keyword. Normal CKDS label name syntax is required.

`blockSize`: The block size for the cryptographic algorithm.

`initializationVector`: Contains the initialization chaining value. You must use the same ICV to decipher the data. This parameter is ignored for the ECB processing rule.

`chainData`: A buffer that is used as a work area for sequences of chained symmetric algorithm encipher requests. When the keyword INITIAL is used, this is an output parameter and receives data that is needed when enciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output parameter; the value received as output from the previous call in the sequence is provided as input to this call, and this call returns new `chainData` that is used as input on the next call.

`clearText`: Text to be enciphered.

`cipherTextLength`: Specifies the maximum number of cipher bytes to return.

`optionalData`: Optional data that is required by a specified algorithm.

```
ESYEOutput :: = SEQUENCE {
     keyParms                OCTET STRING,
     chainData               OCTET STRING,
     cipherText              OCTET STRING
     cipherTextLength        INTEGER
}
```

Where,

`keyParms`: Contains key-related parameters specific to the encryption algorithm and processing mode.

- For the CFB-LCFB processing rule, this 1-byte field specifies the segment size in bytes. Valid values are 1 to the blocksize, inclusive. The blocksize is eight for DES and 16 for AES.
- For the GCM processing rule, this contains the generated authentication tag for the provided plaintext (`plain_text` parameter) and additional authenticated data (`optional_data` parameter).
- For all other processing rules, this field is ignored.

- For the modes where key_parms is used, you must specify the same key_parms when deciphering the text by using the symmetric key decipher callable service.

chainData: Data that is needed when enciphering the next part of the input data.

cipherText: Contains the enciphered data.

cipherTextLength: Specifies the number of bytes returned in cipherText.

Symmetric key decipher (CSNESYD) ASN.1 syntaxes

```
ESYDInput ::= SEQUENCE {
     keyIdentifier            KeyIdentifier
     keyParms                 OCTET STRING,
     blockSize                INTEGER,
     initializationVector     OCTET STRING,
     chainData                OCTET STRING,
     cipherText               OCTET STRING,
     clearTextLength          INTEGER,
     optionalData             OCTET STRING
}
```

Where,

keyIdentifier: Specifies the cipher key, for the KEY-CLR keyword. The parameter must be left-aligned. Specifies an internal clear token, or the label name of a clear key or an encrypted key in the CKDS, for the KEYIDENT keyword. Normal CKDS label name syntax is required. KEYIDENT is valid with DES and AES.

keyParms: Contains key-related parameters specific to the encryption algorithm and processing mode.

- For the CFB-LCFB processing rule, this 1-byte field specifies the segment size in bytes. Valid values are 1 to the block size, inclusive. The block size is eight for DES and 16 for AES.
- For the GCM processing rule, this contains the authentication tag for the provided ciphertext (cipher_text parameter) and additional authenticated data (optional_data parameter).
- For all other processing rules, this field is ignored.
- For the modes where key_parms is used, you must specify the same key_parms used when enciphering the text by using the symmetric key encipher.

blockSize: Block size for the cryptographic algorithm.

initializationVector: Contains the initialization chaining value. You must use the same ICV that was used to encipher the data. This parameter is ignored for the ECB processing rule.

chainData: This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller. The direction is output if the ICV selection keyword is INITIAL. This parameter is ignored if the ICV selection keyword is ONLY. The mapping of the chain_data depends on the algorithm specified. For AES, the chain_data field must be at least 32 bytes in length. The OCV is in the first 16 bytes in the chain_data. For DES, chain_data field must be at least 16 bytes in length.

cipherText: Text to be deciphered.

clearTextLength: Specifies the maximum number of clear bytes to return.

optionalData: Optional data that is required by a specified algorithm or processing mode. For the GCM processing rule, this parameter contains the Additional Authenticated Data (AAD). For all other processing rules, this field is ignored. You must specify the same optionalData used when enciphering the text when using symmetric key encipher.

```
ESYDOutput :: = SEQUENCE {
     chainData               OCTET STRING,
     clearText               OCTET STRING
     clearTextLength         INTEGER,
}
```

Where,

chainData: Data that is needed when deciphering the next part of the input data.

cipherText: Contains the deciphered data.

cipherTextLength: Specifies the number of bytes returned in clearText.

Symmetric key management-related remote services

ICSF supports routines for exporting and importing symmetric keys. The **RemoteCryptoCCA** extended operation allows the CSNFSYX and CSNFSYI ICSF callable services to be available for remote invocation.

Symmetric key export (CSNFSYX) ASN.1 syntaxes

```
FSYXInput ::= SEQUENCE {
    sourceKeyIdentifier      KeyIdentifier,
    transportKeyIdentifier   KeyIdentifier,
    encipheredKeyLength      INTEGER
}
```

Where,

sourceKeyIdentifier: Specifies the application-supplied label or internal token of a secure AES DATA (version X'04'), DES DATA, or variable-length symmetric key token to encrypt under the supplied RSA public key or AES EXPORTER key.

transportKeyIdentifier: Specifies an RSA public key token, AES EXPORTER token, or label of the key to protect the exported symmetric key.

encipheredKeyLength: Specifies the maximum length of the returned encipheredKey that can be accepted by the application.

```
FSYXOutput :: = SEQUENCE {
    sourceKeyIdentifier      KeyIdentifier,
    encipheredKey            OCTET STRING
    encipheredKeyLength      Integer
}
```

Where,

sourceKeyIdentifier: Specifies the returned source key label or token, as updated by the service.

encipheredKey: Specifies the exported key, which is protected by the RSA public or AES EXPORTER key that is specified in the transporterKeyIdentifier on the request.

encipheredKeyLength: Specifies the length of the returned encipheredKey.

Symmetric key import (CSNFSYI) ASN.1 syntaxes

```
FSYIInput ::= SEQUENCE {
    RSAEncipheredKey             OCTET STRING,
    RSAPrivateKeyIdentifier      KeyIdentifier,
    targetKeyIdentifierLength    INTEGER
}
```

Where,

RSAEncipheredKey: Specifies the key to import, which is protected under an RSA public key.

RSAPrivateKeyIdentifier: Specifies an RSA private key token or label whose corresponding public key protects the symmetric key.

targetKeyIdentifierLength: Specifies the maximum length of the returned targetKeyIdentifier that can be accepted by the application.

```
FSYIOutput ::= SEQUENCE {
    targetKeyIdentifier          KeyIdentifier,
    targetKeyIdentifierLength    INTEGER
}
```

Where,

targetKeyIdentifier: Specifies the returned internal token of the imported symmetric key.

targetKeyIdentifierLength: Specifies the maximum length of the returned targetKeyIdentifier that can be accepted by the application.

Asymmetric key management services

ICSF supports different types of PKA key management routines for building, generating, importing, and extracting public keys. The **RemoteCryptoCCA** extended operation allows the CSNFPKB, CSNFPKG, CSNFPKI, and CSNFPKX ICSF callable services to be available for remote invocation.

PKA key token build (CSNFPKB) ASN.1 syntaxes

The ICSF asymmetric key management routine, CSNFPKB, builds an external PKA key token containing unenciphered private RSA, DSS, or ECC keys, or public RSA, DSS, or ECC keys.

```
KeyValue ::= CHOICE {
 [0] msbKeyValue    OCTET STRING
}
```

Note: When Choice 0 of KEYVALUE is used, all integer values must be in big-endian order. For example, most significant byte first.

```
FPKBInput ::= SEQUENCE {
    keyValue                        KeyValue,
    privateKeyName                  ASCII STRING,
    userDefinableAssociatedData     OCTET STRING,
    keyTokenLength                  INTEGER
}
```

Where,

keyValue: A segment of contiguous storage containing a variable number of input clear key values and the lengths of these values in bits or bytes, as specified. See PKA Key Token Build (CSNDPKB and CSNFPKB) in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for the CSNFPKB API description. Currently only choice 0 is supported. All integer values must be in big-endian order.

privateKeyName: Specifies a text string, which contains the name of a private key.

userDefinableAssociatedData: Specifies a string containing the associated data that is placed after the IBM associated data in the token.

keyTokenLength: Specifies the size of the caller allocated storage to hold the returned key.

```
FPKBOutput :: = SEQUENCE {
    keyToken                    OCTET STRING,
    keyTokenLength              INTEGER (0..3500)
}
```

Where,

keyToken: Specifies the returned key token.

keyTokenLength: Specifies the size of the returned key.

PKA key generate (CSNFPKG) ASN.1 syntaxes

The ICSF asymmetric key management routine CSNFPKG can generate a PKA key. The PKA key can be an internal token for use with the DSS algorithm in the digital signature services, or an RSA key for use on the Cryptographic Coprocessor Feature, PCI Cryptographic Coprocessor, PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor, or an ECC key for use on the Crypto Express3 Coprocessor.

```
FPKGInput ::= SEQUENCE {
    regenerationData                OCTET STRING,
    skeletonKeyIdentifier           KeyIdentifier,
    transportKeyIdentifier          KeyIdentifier,
    generatedKeyToken               KeyIdentifier
}
```

Where,

regenerationData: Specifies a string that is used as the basis for creating a particular public-private key pair in a repeatable manner.

skeletonKeyIdentifier: Specifies either a key token or key label.

transportKeyIdentifier: Specifies a 64-byte label of a CKDS record that contains the transport key, 64-byte DES internal key token containing the transport key, or a variable-length AES internal key token containing the transport key.

generatedKeyToken: Specifies the internal token or label of the generated DSS, ECC, or RSA key. The label can be like a retained key for most RSA key tokens.

```
FPKGOutput ::= SEQUENCE {
    generatedKeyToken            KeyIdentifier,
    generatedKeyTokenLength      INTEGER
}
```

Where,

generatedKeyToken: Specifies the internal token or label of the generated DSS, ECC, or RSA key. The label can be like a retained key for most RSA key tokens.

generatedKeyTokenLength: Specifies the length of the returned generatedKeyToken.

PKA key import (CSNFPKI) ASN.1 syntaxes

The ICSF asymmetric key management routine, CSNFPKI, can be used to import an external PKA private key token, a clear PKA key, or an external trusted block token. Output of this service is an ICSF internal token of the RSA, DSS, or ECC private key or trusted block.

```
FPKIInput ::= SEQUENCE {
    sourceKeyIdentifier         KeyIdentifier,
    importerKeyIdentifier       KeyIdentifier,
    targetKeyIdentifier         KeyIdentifier
}
```

Where,

sourceKeyIdentifier: Contains an external token or label of a PKA private key, without section identifier 0x14 (Trusted Block Information), or the trusted block in external form as produced by the Trusted Block Create (CSNDTBC and CSNETBC) service with the ACTIVATE keyword.

importerKeyIdentifier: Specifies a variable-length field containing an AES or DES key identifier that is used to wrap the imported key.

targetKeyIdentifier: Specifies the internal token or label of the imported PKA private key or a Trusted Block.

```
FPKIOutput ::= SEQUENCE {
    importerKeyIdentifier         KeyIdentifier
    targetKeyIdentifier           KeyIdentifier,
    targetKeyIdentifierLength     INTEGER
}
```

Where,

importerKeyIdentifier: Specifies a variable-length field containing an AES or DES key identifier that is used to wrap the imported key.

targetKeyIdentifier: Specifies the internal token or label of the imported PKA private key or a Trusted Block.

targetKeyIdentifierLength: Specifies the length of the returned targetKeyIdentifier.

PKA key extract (CSNFPKX) ASN.1 syntaxes

The ICSF asymmetric key management routine, CSNFPKX, can be used to extract a PKA public key token from a supplied PKA internal or external private key token.

```
FPKXInput ::= SEQUENCE {
    sourceKeyIdentifier                KeyIdentifier,
    targetPublicKeyTokenLength         INTEGER
}
```

Where,

sourceKeyIdentifier: Specifies the internal or external token of a PKA private key or the label of a PKA private key.

targetPublicKeyTokenLength: Specifies the length of the allocated storage to hold the returned targetPublicKeyToken.

```
FPKXOutput ::= SEQUENCE {
    sourceKeyIdentifier                 KeyIdentifier,
    targetPublicKeyToken                OCTET STRING,
    targetPublicKeyTokenLength          INTEGER
}
```

Where,

sourceKeyIdentifier: Specifies

targetPublicKeyToken: Specifies the token of the extracted PKA public key.

targetPublicKeyTokenLength: Specifies the length of the returned targetPublicKeyToken.

PKDS key record management-related remote services

ICSF supports routines for creating, deleting, and reading records in the Public Key Data Set (PKDS). The **RemoteCryptoCCA** extended operation allows the CSNFKRC, CSNFKRD, and CSNFKRR ICSF callable services to be available for remote invocation.

PKDS key record create (CSNFKRC) ASN.1 syntaxes

```
FKRCInput ::= SEQUENCE {
    label         ASCII STRING,
    token         OCTET STRING
}
```

Where,

label: Specifies the 64-byte label of the record to be added to the PKDS.

token: Specifies an RSA, DSS, or ECC private token in either external or internal format, or an RSA, DSS, or ECC public token, or a null token. To store a null token, encode a zero length octet string in the request.

```
FKRCOutput ::= NULL
```

PKDS key record delete (CSNFKRD) ASN.1 syntaxes

```
FKRDInput ::= label ASCII STRING
```

Where,

label: Specifies the 64-byte label of the record to be deleted from the PKDS.

```
FKRDOutput ::= NULL
```

PKDS key record read (CSNFKRR) ASN.1 syntaxes

```
FKRRInput ::= SEQUENCE {
    label                  ASCII STRING,
    tokenLength            INTEGER
}
```

Where,

`label`: Specifies the 64-byte label of the record to be read from the PKDS.

`tokenLength`: Specifies the maximum length of bytes to return for the output token.

```
FKRRInput ::= SEQUENCE {
    token           OCTET STRING,
    tokenLength     INTEGER
}
```

Where,

`token`: Specifies the record that is returned from the PKDS.

`tokenLength`: Specifies the length of the record that is returned from the PKDS.

Part 3. Appendixes

Appendix A. Plug-in sample

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

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

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

Steps for building and running a sample plug-in

How to build and run a sample plug-in:

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

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

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

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

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

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

    Note: For this 64-bit example, it is assumed **PLUGSAMP** is the name that is used when the 64-bit DLL was built, as shown above. The name **PLUGSM31** is a place holder name for the **plugin** configuration option. It can be any name and no DLL with that name must exist.
    See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS, Customizing the LDAP server configuration* chapter, for a complete description of the **plugin** configuration option and its parameters.
11. Restart the LDAP server.

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

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

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

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

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

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

See Chapter 2, “Building an LDAP server plug-in,” on page 5 for more information about building and writing a z/OS LDAP server plug-in.

Appendix B. Accessibility

Accessible publications for this product are offered through the z/OS Information Center, which is available at www.ibm.com/systems/z/os/zos/bkserv/.

If you experience difficulty with the accessibility of any z/OS information, please send a detailed message to mhvrcfs@us.ibm.com or to the following mailing address:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features

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

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

Using assistive technologies

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

Keyboard navigation of the user interface

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

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the z/OS Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually

exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1!

(KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

  Note:

  1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
  2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
  3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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

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

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

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

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

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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

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

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.

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

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

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

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming interface information

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

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

Programming interface information

End Programming interface information

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml (http://www.ibm.com/legal/copytrade.shtml).

Index

K

M

N

O

P

R

S

T

U

Product Number: 5650-ZOS

Printed in USA

SA76-0169-00