# Paged LDAP Search Results

These routines are used to control the number of entries displayed in one page for the search result returned from the server following an LDAP search operation.

- ldap_create_page_control() -- Create a paged results control.
- ldap_parse_page_control() -- Retrieve values in a paged results control.

The ldap_create_page_control() function uses the page size and the cookie to build the paged results control. The control output from ldap_create_page_control() function includes the criticality set based on the value of the isCritical flag. This control is added to the list of client controls sent to the server on the LDAP search request.

When a paged results control is returned by the server, the ldap_parse_page_control() function can be used to retrieve the values from the control. The function takes as input the server controls returned by the server, and returns a cookie to be used on the next paged results request for this search operation.

**Note:**

If the page size is greater than or equal to the search sizeLimit value , the server ignores the paged results control because the request can be satisfied in a single page. No paged results control value is returned by the server in this case. In all other cases, error or not, the server returns a paged results control to the client.

### Simple paged results of search results

Simple Paged Results provides paging capabilities for LDAP clients that want to receive just a subset of search results (page) instead of the entire list. The next page of entries is returned to the client application for each subsequent paged results search request submitted by the client until the operation is canceled or the last result is returned. The server ignores a simple paged results request if the page size is greater than or equal to the sizeLimit value for the server because the request can be satisfied in a single operation.

The ldap_create_page_control() API takes as input a page size and a cookie, and outputs an LDAPControl structure that can be added to the list of client controls sent to the server on the LDAP search request. The page size specifies how many search results must be returned for this request, and the cookie is an opaque structure returned by the server. (On the initial paged results search request, the cookie must be a zero-length string). No assumptions must be made about the internal organization or value of the cookie. The cookie is used on subsequent paged results search requests when more entries are to be retrieved from the results set. The cookie must be the value of the cookie returned on the last response returned from the server on all subsequent paged results search requests. The cookie is empty when there are no more entries to be returned by the server, or when the client application abandons the paged results request by sending in a zero page size. After the paged results search request has been completed, the cookie must not be used because it is no longer valid.

The LDAPControl structure returned by ldap_create_page_control() can be used as input to ldap_search_ext() or ldap_search_ext_s(), which are used to make the actual search request.

**Note:**

Server side simple paged results is an optional extension of the LDAP v3 protocol, so the server you have bound to prior to the ldap_search_ext() or ldap_search_ext_s() call might not support this function.

Upon completion of the search request you submitted using ldap_search_ext() or ldap_search_ext_s(), the server returns an LDAP result message that includes a paged results control. The client application can parse this control using ldap_parse_page_control(), which takes the returned server response controls (a null terminated array of pointers to LDAPControl structures) as input. ldap_parse_page_control() outputs a cookie and the total number of entries in the entire search result set. Servers that cannot provide an estimate for the total number of entries might set this value to zero. Use ldap_controls_free() to free the memory used by the client application to hold the server controls when you are finished processing all controls returned by the server for this search request.

The server might limit the number of outstanding paged results operations from a given client or for all clients. A server with a limit on the number of outstanding paged results requests might return either LDAP_UNWILLING_TO_PERFORM in the sortResultsDone message or age out an older paged results request. There is no guarantee to the client application that the results of a search query

have remained unchanged throughout the life of a set of paged results request/response sequences. If the result set for that query has changed since the initial search request specifying paged results, the client application might not receive all the entries matching the given search criteria. When chasing referrals, the client application must send in an initial paged results request, with the cookie set to null, to each of the referral servers. It is up to the application using the client's services to decide whether or not to set the criticality as to the support of paged results, and to handle a lack of support of this control on referral servers as appropriate, based on the application. Additionally, the LDAP server does not ensure that the referral server supports the paged results control. Multiple lists can be returned to the client application, some not paged. It is the client application's decision as to how best to present this information to the end user. Possible solutions include:

- Combine all referral results before presenting to the end user
- Show multiple lists and the corresponding referral server host name
- Take no extra steps and show all results to the end user as they are returned from the server

The client application must turn off referrals to get one truly paged list; otherwise, when chasing referrals with the paged results search control specified, unpredictable results might occur.

More information about the simple paged results search control, with control OID of 1.2.840.113556.1.4.319, can be found in RFC 2686 - LDAP Control Extension for Simple Paged Results Manipulation.

### Related Information

- ldap_create_page_control() -- Create a paged results control.
- ldap_parse_page_control() -- Retrieve values in a paged results control.
- ldap_create_sort_keylist() -- Create a structure with sort key values.
- ldap_free_sort_keylist() -- Free all memory used by the sort key list.
- ldap_create_sort_control() -- Create a sorted results control.
- ldap_parse_sort_control() -- Retrieve values in a sorted results control.
- ldap_search() -- Asynchronously search the directory.
- ldap_search_ext() -- Asynchronously search the directory with controls.
- ldap_search_ext_s() -- Synchronously search the directory with controls.
- ldap_parse_result() -- Extract information from results.
- ldap_control_free() -- Free a single LDAPControl structure.
- ldap_controls_free() -- Free an array of LDAPControl structures.

## Example

The following example shows how to use the LDAP paged search APIs.

**Note:** By using the code examples, you agree to the terms of the Code license and disclaimer information.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <ldap.h>
#include <lber.h>

static char ibmid[] = "Copyright IBM Corporation 2003 LICENSED MATERIAL - PROGRAM PROPERTY OF IBM";
#define         BIND_DN "cn=administrator"
#define         BIND_PW "adminpwd"

char             *server, *base, *filter, *scopes[] = { "BASE", "ONELEVEL", "SUBTREE" };
int              scope;
```

```
LDAP             *ld;

int main (int argc, char *argv[])
{
    int            l_rc, l_entries, l_port, l_entry_count=0, morePages, l_errcode=0, page_nbr;
    unsigned long  pageSize;
    struct berval  *cookie=NULL;
    char           pagingCriticality = 'T', *l_dn;
    unsigned long  totalCount;
    LDAPControl    *pageControl=NULL, *M_controls[2] = { NULL, NULL }, **returnedControls = NULL;
    LDAPMessage    *l_result, *l_entry;

   /*****************************************************************/
   /* Check input parameters                                        */
   /*                                                               */
   if (argc < 5)
   {
      printf("The input parameters are as follows:\n");
      printf("     1. Search base\n");
      printf("     2. Filter\n");
      printf("     3. Search Scope (0=base, 1=onelevel, OR 2=subtree)\n");
      printf("     4. Page size\n");
   return 0;
   }
   /*                                                               */
   /*****************************************************************/

   /*****************************************************************/
   /* Set default values: Server and Port. And then parse           */
   /* input parameters into program variables                       */
   /*                                                               */
   server = NULL;
   l_port   = LDAP_PORT;

   base   = argv[1];
   filter = argv[2];
   scope  = atoi(argv[3]);
   pageSize = atoi(argv[4]);
   /*                                                               */
   /*****************************************************************/

   /*****************************************************************/
   /* Initialize an LDAP session                                    */
   /*                                                               */
   ld = ldap_init(server, l_port);
   /* Check if connection is OK                                     */
   if (ld == NULL)
   {
      printf("==Error==");
      printf("  Init of server %s at port %d failed.\n", server, l_port);
      return 0;
   }
   /*                                                               */
   /*****************************************************************/

   /*****************************************************************/
```

```
/* Bind as the ldap administrator                                  */
/*                                                                 */
l_rc = ldap_simple_bind_s (ld, BIND_DN , BIND_PW);
if ( l_rc != LDAP_SUCCESS)
{
   printf("==Error== %s");
   printf("  Unable to Bind to the LDAP server.  Return code is %d.\n", l_rc);
   return 0;
}
/*                                                                 */
/*****************************************************************/

printf("  The search parms were:\n");
printf("        base: %s\n",base);
printf("       scope: %s\n",scopes[scope]);
printf("      filter: %s\n",filter);
printf("   page size: %d\n",pageSize);
printf("  The entries returned were:\n");
page_nbr = 1;

/*****************************************************************/
/* Get one page of the returned results each time                  */
/* through the loop                                                */
do
{
   l_rc = ldap_create_page_control(ld, pageSize, cookie, pagingCriticality, &pageControl);

   /* Insert the control into a list to be passed to the search.     */
   M_controls[0] = pageControl;

   /* Search for entries in the directory using the parmeters.       */
   l_rc = ldap_search_ext_s(ld, base, scope, filter, NULL, 0, M_controls, NULL, NULL, 0, &l_result);
   if ((l_rc != LDAP_SUCCESS) & (l_rc != LDAP_PARTIAL_RESULTS))
   {
      printf("==Error==");
      printf("  Failure during a search.  Return code is %d.\n",l_rc);
      ldap_unbind(ld);
      break;
   }

   /* Parse the results to retrieve the contols being returned.      */
   l_rc = ldap_parse_result(ld, l_result, &l_errcode, NULL, NULL, NULL, &returnedControls, LDAP_FALSE);

   if (cookie != NULL)
   {
      ber_bvfree(cookie);
      cookie = NULL;
   }

   /* Parse the page control returned to get the cookie and          */
   /* determine whether there are more pages.                        */
   l_rc = ldap_parse_page_control(ld, returnedControls, &totalCount, &cookie);

   /* Determine if the cookie is not empty, indicating there are more pages */
   /* for these search parameters. */
   if (cookie && cookie->bv_val != NULL && (strlen(cookie->bv_val) > 0))
   {
```

```
      {
         morePages = LDAP_TRUE;
      }
      else
      {
         morePages = LDAP_FALSE;
      }

      /* Cleanup the controls used. */
      if (returnedControls != NULL)
      {
         ldap_controls_free(returnedControls);
         returnedControls = NULL;
      }
      M_controls[0] = NULL;
      ldap_control_free(pageControl);
      pageControl = NULL;

      /*****************************************************************/
      /* Disply the returned result                                    */
      /*                                                               */
      /* Determine how many entries have been found.                   */
      if (morePages == LDAP_TRUE)
         printf("===== Page : %d =====\n", page_nbr);
      l_entries = ldap_count_entries(ld, l_result);

      if (l_entries > 0)
      {
         l_entry_count = l_entry_count + l_entries;
      }

      for ( l_entry = ldap_first_entry(ld, l_result);
            l_entry != NULL;
            l_entry = ldap_next_entry(ld, l_entry) )
      {
         l_dn = ldap_get_dn(ld, l_entry);
         printf("    %s\n",l_dn);
      }

      /* Free the search results.                                      */
      ldap_msgfree(l_result);
      page_nbr = page_nbr + 1;

   } while (morePages == LDAP_TRUE);

   printf("\n  %d entries found during the search",l_entry_count);
   /* Free the cookie since all the pages for these search parameters   */
   /* have been retrieved.                                          */
   ber_bvfree(cookie);
   cookie = NULL;

   /* Close the LDAP session.                                       */
   ldap_unbind(ld);

   return 0;
}
```