Embedded LDIF for C

By TeraCortex

Table of Contents

Independent Submission
INTERNET-DRAFT
Intended Status: Proposed Standard
Expires 2015/08/26
draft-hollstein-embedded-ldif-c-03.txt

Christian Hollstein
TeraCortex
February 2015

The Embedded LDAP Data Interchange Format for C (EMLDIF-C)

Status of This Memo

This document is not an Internet Standards Track specification.
It is published for examination, experimental implementation, and evaluation. Distribution of this memo is unlimited.

Abstract

EMLDIF specifies how LDAP operations can be encoded in EXLDIF and embedded in high level programming languages. EMLDIF is is the common basis for language - dependant specifications. This document specifies Embedded LDIF for the programming language C. In order to make EMLDIF-C source files interoperable across vendor specific implementations the specification goes into detail about data structures and API.

Copyright Notice

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

1. Overview

This document specfies how EXLDIF can be embedded and used in the high level programming language C. Based on [EMLDIF] this handles the following issues:

- Use of static functions

- Export of global functions for dynamic attachment

- Multi Threading

- Application Programmer Interface (API)

- Result Handling

- Examples

The chapters 2 through 5 give a formal specification of the language elements.

1.1. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2. Implementation Types

As specified in [EMLDIF] there are two types of implementations:

Type I uses generates shared libraries from EMLDIF-C source files and uses dynamic linking to attach them to the running process.

Type II generates executable binaries from EMLDIF-C source files and executes them on the fly or leaves them for later execution.

Any implemantation SHOULD be able to handle more than one EMLDIF-C source file upon each invocation.

2. The EMLDIF-C File

2.1. General Layout

An EMLDIF-C file is a text file with source code of the programming language "C". In this file EXLDIF records MAY appear inside of blocks. A block is opened with "{" and closed with "}". EXLDIF records MUST NOT appear outside of blocks.

2.2. Header File Inclusion

Each EMLDIF-C file must contain the following inclusion statement near the top of the file:

```
#include <eldc_exec.h>

The content of this file is specified as follows:


#define ELDC_EXPORT(fname, farg)  void *fname(void *farg)
#define ELDC_FUNCTION(fname)      fname
#define ELDC_INSTANCE(iname)      int iname = 0;
#define ELDC_ARGUMENT(farg)       void *farg


typedef struct EldcAva    {          /* Attribute - Value assertions
                                        "value" is a null terminated
                                        list of character pointers.
                                        It is NULL, if there is only
                                        an attribute name
                                    */
  char                 *aname;       /* Attribute name             */
  char                 **value;      /* List of NULL terminated
```

```
                                        value(s)
                                     */
} EldcAva;


typedef struct EldcObject {          /* Describes a searched object
                                        "ava" is a null terminaed
                                        list of EldcAva pointers
                                     */
  char                *dn;           /* Distinguished name         */
  EldcAva             **ava;         /* List of AVAs               */
} EldcObject;

typedef struct EldcResult {          /* Describes an operation result
                                        "object" is a null terminated
                                        list of EldcObject pointers
                                     */
  unsigned long long   sent;         /* Bytes sent by client       */
  unsigned long long   recd;         /* Bytes received by client   */
  unsigned int         operations;   /* Number of operations       */
  unsigned int         errcnt;       /* Number of errors           */
  unsigned int         entries;      /* Number of received search
                                        entries
                                     */

  int                  rcode;        /* Response code received from
                                        server
                                     */
  int                  rescnt;       /* Number of results in array */
  int                  msgid;        /* Server message ID          */
  int                  restype;      /* Result message type        */
  char                 *rmessage;    /* NULL terminated server result
                                        message
                                     */
  EldcObject           **object;     /* List of searched objects   */
} EldcResult;


typedef struct EldcThread {          /* Describes the current thread
```

```
                                        executing the toplevel
                                        function
                                    */
    int                 fileno;     /* Input file number 0 ... N  */
    int                 threadno;   /* Thread number 0 ... N      */
    unsigned int        *random;    /* Global random array   */
    int                 randcnt;    /* Number of values in random */
    int                 randsd;     /* File descriptor random file*/

  } EldcThread;
```

int eldc_result_free(EldcResult ***result);

2.3. Exported Functions

This chapter is only relevant for type I implementations.

A collection of EMLDIF-C source code files MUST contain at least one exported function. "Exported" means: It is recognized by the parser, added to a "jump table" and is explicitely looked up in the jump table when the generated library is attached to the running process. During exection the function will be called from the executive binary as entry point of Embedded LDIF operations. Exported functions MUST be defined as follows:

```
eldc-export = FILL "ELDC_EXPORT" FILL "(" FILL
                export-name
                FILL ","
               eldc-thread
               FILL ")" FILL
```

```
export-name = 1*( ALPHA / DIGIT / "-" / "_" )
```

This is the function name by which the exported function will be looked up during dynamic linking.

```
eldc-thread = 1*( ALPHA / DIGIT / "-" / "_" )
```

This is a variable name. The variable gives access to the data structures representing the Embedded LDIF records.

Exported functions are of type "void *", thus must return a pointer.

2.4. Internal Functions

EMLDIF-C MAY contain arbitrary definitions of C functions. They MAY be local (defined static) to the respective source code file or global because they are called from within other source files. In any case they are not exported in the sense of chapter 2.3.

If a function contains Embedded LDIF records the function header MUST be defined as follows:

```
eldc-function = data-type 1*SPACE "ELDC_FUNCTION"
                        FILL "(" FILL
                        function-name
                        FILL ")" FILL "(" FILL
                        "ELDC_ARGUMENT"
                        FILL "(" FILL
                        eldc-thread
                        FILL ")" FILL
                        *parameter-list FILL ")"
```

These properties are at the choice of the programmer:

data-type The data type the function shall return

function-name The name by which the function will be called

eldc-thread = 1*(ALPHA / DIGIT / "-" / "_")

This is a variable name. The variable gives access to the data structures representing the Embedded LDIF records.

parameter-list List of function parameters according to the syntax

of C function parameters. If present it MUST begin with a comma ",".

2.5. Result Structures

Results of LDAP operations are returned as NULL terminated array of pointers to structures of type EldcResult. Each such structure contains:

- A result code

- A message ID

- A result message type

- An optional NULL terminated result message if received from the server

- An optional NULL terminated array list of pointers to structures of type EldcObject if returned from a SEARCH operation.

Implementations MAY add other fields to this structure, for example statistics related ones like:

- The number of bytes sent and received so far

- The number of executed operations

- The number of errors

- The number of received entries from search requests

If the array of objects is present each object contains:

- A NULL terminated distinguished name. The distinguished name MAY have zero length if the SEARCH operation targeted the LDAP server top level object.

- A NULL terminated array of pointers to structures of type EldcAva.

Each EldcAva element contains:

- A NULL terminated attribute name

- An optional NULL terminated array of pointers to "char *". Each element of this array points to NULL terminated value string.

2.6. Result Handling API

Results are delivered in lists of EldcResults. They are allocated in heap memory. The programmer of the EMLDIF-C source file is responsible to free the used memory using an API call. The API function is declared here:

```
int eldc_result_free(EldcResult ***result);
```

The parameter has the following meaning:

- result This is the name of a variable that was defined with in the EMLDIF-C source file using the RESULT directive (see chapter 4.3 of [EMLDIF].

3. Examples

The examples below assume a LDAP server running on the localhost at port 389. The bind DN is:

cn=Manager,dc=my-domain,dc=com

The password is:

secret

The following object exists:

dc=my-domain,dc=com

All examples are self - cleaning. At the end they remove any objects created to leave the directory in the same state as before.

3.1. Basic Operations

Use ADD to create an object, MODIFY to replace a value, SEARCH the result, COMPARE the result with your expectation, RENAME the object and DELETE the renamed object. The example shows also the handling of comments in the MODIFY record and indentation in the ADD record.

```
#include <eldc_exec.h>

ELDC_EXPORT(test_01, ct)
{

  /* Connect to the server
  */
  dn:
  changetype: connect
  connection: ldap://localhost:389

  /* Authenticate
  */
  dn: cn=Manager,dc=my-domain,dc=com
  changetype: bind connectionId(test_01.eldc:0:test_01:0:0:0)
  passwd: secret

  /* Add an object
  */
  dn: commonName=0000,dc=my-domain,dc=com
  changetype: add
  objectClass: inetOrgPerson
  commonName: 0000
  sn: Valcourt-Zywiel
  mail: N.Valcourt.Zywiel@test.co.uk
  userPassword: some
   thing
```

```
/* Modify the object
*/
dn: commonName=0000,dc=my-domain,dc=com
changetype: modify
replace: mail
mail: myaddress@my-domain.com
#-
#replace: userPassword
 userPassword: otherthing


  /* Search for results
*/
dn: commonName=0000,dc=my-domain,dc=com
changetype: search
scope: base
deref: never
sizelimit: 0
timelimit: 0
attrsonly: 0
filter: (objectClass=*)

/* Compare an attribute
*/
dn: commonName=0000,dc=my-domain,dc=com
changetype: compare
mail: myaddress@my-domain.com

/* Rename the entry
*/
dn: commonName=0000,dc=my-domain,dc=com
changetype: moddn
newrdn: commonName=0001
deleteoldrdn: true

/* Delete the renamed entry
*/
dn: commonName=0001,dc=my-domain,dc=com
changetype: delete
```

```
    dn:
    changetype: unbind

    return NULL;
  }
```

3.2. Transactions

The example below shows a LDAP transaction according to [RFC5805]:

```
#include <eldc_exec.h>


ELDC_EXPORT(test_02, ct)
{

  /* Connect to the server
  */
  dn:
  changetype: connect
  connection: ldap://localhost:389

  /* Authenticate
  */
  dn: cn=Manager,dc=my-domain,dc=com
  changetype: bind connectionId(test_02.eldc:0:test_02:0:0:0)
  passwd: secret

  /* Start a transaction
  */
  dn:
  changetype: extended
  oid: 1.3.6.1.1.21.1
  value:
  responses: 1

  /* Add an object
  */
  dn: commonName=0000,dc=my-domain,dc=com
  control: 1.3.6.1.1.21.2 true:transactionId
   (test_02.eldc::test_02:2:0:0)
  changetype: add
  objectClass: inetOrgPerson
```

```
commonName: 0000
sn: Valcourt-Zywiel
mail: N.Valcourt.Zywiel@test.co.uk

/* Modify the object
*/
dn: commonName=0000,dc=my-domain,dc=com
control: 1.3.6.1.1.21.2 true:transactionId
 (test_02.eldc::test_02:2:0:0)
changetype: modify
replace: mail
mail: myaddress@my-domain.com

/* Rename the object
*/
dn: commonName=0000,dc=my-domain,dc=com
control: 1.3.6.1.1.21.2 true:transactionId
 (test_02.eldc::test_02:2:0:0)
changetype: moddn
newrdn: commonName=0001
deleteoldrdn: true     /* Delete the renamed object
*/
dn: commonName=0001,dc=my-domain,dc=com
control: 1.3.6.1.1.21.2 true:transactionId
 (test_02.eldc::test_02:2:0:0)
changetype: delete


/* Stop the transaction
*/
dn:
control: 1.3.6.1.1.21.2 true:transactionId
 (test_02.eldc::test_02:2:0:0)
changetype: extended
oid: 1.3.6.1.1.21.3
value: transactionId(test_02.eldc::test_02:2:0:0)
responses: 1

dn:
```

```
    changetype: unbind

    return NULL;
  }
```

3.3. Source Code Parametrisation with Static Value Replacement

This example shows how source code can be parameterized by use of the following environment variables:

LDAP_HOST Target LDAP server host name

LDAP_HOST Target LDAP server port number

LDAP_ADMIN Target LDAP server bind DN

The variables must have been set and exported on operating system level.

```
#include <eldc_exec.h>

ELDC_EXPORT(test_03, ct)
{

  /* Connect to the server
  */
  dn:
  changetype: connect
  connection: ldap://$LDAP_HOST:$LDAP_PORT


  /* Authenticate
  */
  dn: $LDAP_ADMIN
  changetype: bind connectionId(test_03.eldc:0:test_03:0:0:0)
  passwd: secret


  /* Search an object
  */
```

```
    dn: dc=my-domain,dc=com
    changetype: search
    scope: base
    deref: never
    sizelimit: 0
    timelimit: 0
    attrsonly: 0
    filter: (objectClass=*)

    dn:
    changetype: unbind

    return NULL;
  }
```

3.4. Mass Data Creation with Dynamic Value Replacement

This example shows how values inside of EXLDIF records can be replaced with local variables of the embedding program. The code employs also the definition of internal functions and uses an environment variable to get the number of operations to execute.

```
#include <eldc_exec.h>

static int ELDC_FUNCTION(test_04a)(ELDC_ARGUMENT(ct), int cnt)
{
  int i;

  /* The loop below adds the objects

  commonName=0000,dc=my-domain,dc=com
  commonName=0001,dc=my-domain,dc=com
  commonName=0002,dc=my-domain,dc=com
  ...
  */
  for ( i = 0; i < cnt; i++ ) {

    /* Add an object
    */
    dn: commonName=%04d:i%,dc=my-domain,dc=com
    changetype: add connectionId(test_04.eldc:0:test_04:0:0:0)
    objectClass: inetOrgPerson
    sn: Valcourt-Zywiel
    mail: N.Valcourt.Zywiel@test.co.uk

  }

  return 0;
}


static int ELDC_FUNCTION(test_04b)(ELDC_ARGUMENT(ct), int cnt)
```

```
  {
    int i;

    /* The loop below deletes the objects

    commonName=0000,dc=my-domain,dc=com
    commonName=0001,dc=my-domain,dc=com
    commonName=0002,dc=my-domain,dc=com
    ...
    */
    for ( i = 0; i < cnt; i++ ) {

      /* Delete an object
      */
      dn: commonName=%04d:i%,dc=my-domain,dc=com
      changetype: delete connectionId(test_04.eldc:0:test_04:0:0:0)

    }

    return 0;
  }


  ELDC_EXPORT(test_04, ct)
  {
    int cnt;


    /* Connect to the server
    */
    dn:
    changetype: connect
    connection: ldap://localhost:389
```

```
  /* Authenticate
  */
  dn: cn=Manager,dc=my-domain,dc=com
  changetype: bind connectionId(test_04.eldc:0:test_04:0:0:0)
  passwd: secret

  /* Get the number of operations to execute
  */
  cnt = $OPCNT;

  test_04a(ct, cnt);
  test_04b(ct, cnt);

  dn:
  changetype: unbind

  return NULL;
}
```

3.5. Search Result Handling

```
#include <eldc_exec.h>

ELDC_EXPORT(test_06, ct)
{
  /* Connect to the server
  */
  dn:
  changetype: connect
  connection: ldap://localhost:389

  /* Authenticate
  */
  dn: cn=Manager,dc=my-domain,dc=com
  changetype: bind connectionId(test_06.eldc:0:test_06:0:0:0)
  passwd: secret


  /* Add an object
  */
  dn: commonName=0000,dc=my-domain,dc=com
  changetype: add
  objectClass: inetOrgPerson
  commonName: 0000
  sn: Valcourt-Zywiel
  mail: N.Valcourt.Zywiel@test.co.uk


  /* Search the sub tree. Store results in the variable
     "myresult"
  */
  dn: dc=my-domain,dc=com
  changetype: search result(myresult:3:0)
  scope: sub
  deref: find
  sizelimit: 0
```

```
    timelimit: 0
    attrsonly: 0
    filter: (objectClass=*)

    /*
      Do something useful with the results
      ....
    */

    /* Free the memory allocated for the results
    */
    eldc_result_free(&myresult);

    return NULL;
  }
  <CODE ENDS>
```

3.6. Asynchronous Mode

```
  <CODE BEGINS>
  #include <eldc_exec.h>

  ELDC_EXPORT(test_06, ct)
  {
    /* Connect to the server
    */
    dn:
    changetype: connect
    connection: ldap://localhost:389

    /* Authenticate
    */
    dn: cn=Manager,dc=my-domain,dc=com
    changetype: bind connectionId(test_06.eldc:0:test_06:0:0:0)
    passwd: secret
```

```
/* Add an object
*/
dn: commonName=0000,dc=my-domain,dc=com
changetype: add
objectClass: inetOrgPerson
commonName: 0000
sn: Valcourt-Zywiel
mail: N.Valcourt.Zywiel@test.co.uk

/* Search the sub tree. Send three requests in asynchronous
   mode.
*/
dn: dc=my-domain,dc=com
control: 1.3.6.1.4.1.30275.7.6 true:3
changetype: search
scope: sub
deref: find
sizelimit: 0
timelimit: 0
attrsonly: 0
filter: (objectClass=*)

/* Compare the added object
*/
dn: commonName=0000,dc=my-domain,dc=com
changetype: compare
sn: Valcourt-Zywiel

/* Abandon the search request */
dn:
changetype: abandon
messageId: messageId(test_06.eldc::test_06:3:0:0)

/* Expect just one response for the compare request.
   The search request is abandoned and the abandon
```

```
      request has no response
   */
   dn:
   changetype: response
   responses: 1

   /* Cleanup */
   dn: commonName=0000,dc=my-domain,dc=com
   changetype: delete

   dn:
   changetype: unbind
   return NULL;
}
```

4. Security Considerations

In addition to the security issues of LDIF files [RFC2849] Extended LDIF may contain authentication information used for BIND operations. This sensitive data MUST NOT be displayed to unauthorized people.

In Embedded LDIF it is pretty easy to create a program firing millions of requests to a LDAP server in short time frame. Such denial of service attacks are illegal. Their prevention is not in scope of this specification.

General security considerations [RFC4510], especially those associated with update operations [RFC4511], apply to this extension.

5. IANA Considerations

There are no new object identifiers associated with this specification.

6. Acknowledgments

The author gratefully acknowledges the contributions made by Internet Engineering Task Force participants.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC2849] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", RFC 2849, June 2000.

[RFC4510] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

[RFC4511] Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

7.2. Informative References

[RFC5805] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP) Transactions", RFC 5805, March 2010.

[EMLDIF] Hollstein, C., "The Embedded LDAP Data Interchange Format (EMLDIF)", draft-hollstein-embedded-ldif-03.txt, February 2015.

Author's address

Christian Hollstein
TeraCortex
Hopfenbrede 2
D-49179 Ostercappeln

E-Mail: chollstein@teracortex.com
Phone: 0049 / 5473 / 9933
Mobile: 0049 / 160 / 96220958