```
**********************************************************
    6624 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/Makefile
5910 libnisdb won't build with modern GCC
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2015 Gary Mills
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#

LIBRARY= libnisdb.a
VERS=   .2
PROTOCOL_DIR= $(ROOT)/include/rpcsvc
YPTOL_DIR= ./yptol

SED=    sed

OBJECTS = \
        db_entry.o db_entry_c_xdr.o \
        db_item.o db_item_c_xdr.o \
        db_vers.o db_vers_c_xdr.o \
        db_pickle.o \
        db_table.o db_table_c_xdr.o \
        db_index_entry.o db_index_entry_c.o \
        db_index.o db_index_c_xdr.o \
        db_scheme.o db_scheme_c_xdr.o \
        db_query.o db_query_c_xdr.o \
        db_mindex.o db_mindex2.o db_mindex3.o db_mindex_c_xdr.o \
        db_log_entry.o db_log_entry_c_xdr.o \
        db_log.o \
        db.o \
        xdr_nullptr.o \
        db_dictionary.o db_dictionary_c_xdr.o \
        db_dictxdr.o db_dictlog.o db_dictlog_c_xdr.o \
        nis_db.o \
        nisdb_mt.o nisdb_rw.o \
        nis_hashitem.o \
        ldap_attr.o ldap_cto.o ldap_glob.o ldap_ldap.o \
        ldap_map.o ldap_nisdbquery.o \
        ldap_op.o ldap_parse.o ldap_print.o ldap_ruleval.o \
        ldap_util.o ldap_val.o ldap_xdr.o ldap_scheme.o \
        nis_ldap.o \
        nis_parse_ldap_attr.o nis_parse_ldap_conf.o \
        nis_parse_ldap_map.o nis_parse_ldap_util.o \
```



```
        nis_parse_ldap_yp_util.o        \
        dit_access.o \
        dit_access_utils.o \
        lock_update.o \
        map_conv.o \
        map_ctrl.o \
        yptol_utils.o \
        shim.o \
        shim_ancil.o \
        shim_lockmap.o \
        ttl_utils.o \
        update.o \
        shim_changepasswd.o \
        stubs.o

X_FILES= \
    db_c.x             db_dictionary_c.x db_entry_c.x  db_index_c.x     \
    db_index_entry_c.x db_item_c.x       db_log_c.x    db_log_entry_c.x \
    db_mindex_c.x      db_query_c.x      db_scheme_c.x db_table_c.x     \
    db_vers_c.x        db_dictlog_c.x

DERIVED_HEADERS= $(X_FILES:%_c.x=%_c.h) $(X_FILES:%_c.x=%.h)
DERIVED_SOURCES= $(X_FILES:%_c.x=%_c_xdr.c)
DERIVED_FILES= $(DERIVED_HEADERS) $(DERIVED_SOURCES)

# delete the derived files when cleaning up
CLEANFILES += $(DERIVED_FILES)

# include library definitions
include ../Makefile.lib

MAPFILES =      mapfile-vers

SRCS= db_dictionary_c_xdr.c db_dictlog_c_xdr.c db_dictxdr.c \
db_entry_c_xdr.c db_index_c_xdr.c db_index_entry_c_xdr.c db_item_c_xdr.c \
db_log_c_xdr.c db_log_entry_c_xdr.c db_mindex_c_xdr.c db_query_c_xdr.c \
db_scheme_c_xdr.c db_table_c_xdr.c db_vers_c_xdr.c \
$(YPTOL_DIR)/dit_access.c \
$(YPTOL_DIR)/dit_access_utils.c \
$(YPTOL_DIR)/lock_update.c \
$(YPTOL_DIR)/map_conv.c \
$(YPTOL_DIR)/map_ctrl.c \
$(YPTOL_DIR)/yptol_utils.c \
$(YPTOL_DIR)/shim.c \
$(YPTOL_DIR)/shim_ancil.c \
$(YPTOL_DIR)/shim_lockmap.c \
$(YPTOL_DIR)/ttl_utils.c \
$(YPTOL_DIR)/update.c \
$(YPTOL_DIR)/shim_changepasswd.c \
$(YPTOL_DIR)/stubs.c \
#db_c_xdr.c xdr_nullptr.c

# Libnisdb has grown large enough that there may be too many symbols
# wanting 'small' PIC references. Hence, compile some files with
# 'large' PIC references; the parser files are good candidates, since
# they're only used at startup, and performance isn't too important.
LARGESYMS=      nis_parse_ldap_conf.o \
                nis_parse_ldap_attr.o \
                nis_parse_ldap_yp_util.o \
                nis_parse_ldap_map.o \
                ldap_parse.o \
                dit_access.o \
                dit_access_utilis.o \
                map_conv.o \
                map_ctrl.o \
                shim.o \
```

```
128                 shim_ancil.o \
129                 shim_changepasswd.o \
130                 update.o \
131                 stubs.o \
132                 yptol_utils.o \
133                 nis_parse_ldap_util.o
134 LARGEPICS=      $(LARGESYMS:%=pics/%)
135 $(LARGEPICS) := sparc_C_PICFLAGS = $(C_BIGPICFLAGS)
136 $(LARGEPICS) := i386_C_PICFLAGS = $(C_BIGPICFLAGS)

138 # More and stricter semantic checks and lint-like checks
139 CFLAGS  +=      $(CCVERBOSE)

141 CPPFLAGS +=     -D_REENTRANT

143 # Have an unused variable that can't be removed
144 UVAR_PICS = \
145 pics/db_dictionary_c_xdr.o \
146 pics/db_dictlog_c_xdr.o \
147 pics/db_entry_c_xdr.o \
148 pics/db_index_c_xdr.o \
149 pics/db_item_c_xdr.o \
150 pics/db_log_entry_c_xdr.o \
151 pics/db_mindex_c_xdr.o \
152 pics/db_query_c_xdr.o \
153 pics/db_scheme_c_xdr.o \
154 pics/db_table_c_xdr.o \
155 pics/db_vers_c_xdr.o \
156 pics/ldap_xdr.o \
157 pics/dit_access.o \
158 pics/update.o \
159 pics/yptol_utils.o \
160 pics/map_ctrl.o

162 $(UVAR_PICS) := CERRWARN += -_gcc=-Wno-unused-variable

164 # Correcting these might alter logic
139 CERRWARN +=     -_gcc=-Wno-unused-variable
140 CERRWARN +=     -_gcc=-Wno-unused-value
165 CERRWARN +=     -_gcc=-Wno-uninitialized
142 CERRWARN +=     -_gcc=-Wno-implicit-function-declaration
166 CERRWARN +=     -_gcc=-Wno-switch
144 CERRWARN +=     -_gcc=-Wno-parentheses

168 # Extra includes, from yp, for yptol files.
169 CPPFLAGS += -I$(SRC)/cmd/ypcmd

171 # Need the path to nis_clnt.h
172 CLNT_PICS = pics/db_entry_c_xdr.o pics/ldap_xdr.o pics/db_mindex_c_xdr.o

174 $(CLNT_PICS) := CPPFLAGS += -I../libnsl/nis/gen

176 LIBS = $(DYNLIB)
177 ZDEFS=
178 LDLIBS += -lnsl -lldap -lc

180 # A number of interfaces are interposed by numerous applications, therefore
181 # prevent direct binding to anything in libnisdb.  Disable libnisdb from
182 # directly binding to itself, but allow libnisdb to directly bind to its
183 # dependencies (ie. map -Bdirect -> -zdirect).  Ensure lazy loading is
184 # established (which is enabled automatically with -Bdirect).  In addition,
185 # libnisdb interposes on socket(), so tag this library as an interposer.
186 # dependencies (lazyload).
187 BDIRECT =
188 DYNFLAGS +=     $(BNODIRECT) $(ZINTERPOSE) $(ZDIRECT) $(ZLAZYLOAD)
```



```
190 all :=          TARGET= all
191 clean :=        TARGET= clean
192 clobber :=      TARGET= clobber
193 install :=      TARGET= install
194 lint :=         TARGET= lint

196 .KEEP_STATE:

198 all:    $(DERIVED_FILES) $(LIBS)

200 install: all $(ROOTLIBS) $(ROOTLINKS)

202 lint: $(DERIVED_FILES) .WAIT lintcheck

204 pics/%.o: %.c
205         $(COMPILE.c) -o $@ $<
206         $(POST_PROCESS_O)

208 pics/%.o: %.cc
209         $(COMPILE.cc) -o $@ $<
210         $(POST_PROCESS_O)

212 objs/%.o pics/%.o profs/%.o: $(YPTOL_DIR)/%.c
213         $(COMPILE.c) -o $@ $<
214         $(POST_PROCESS_O)

216 #
217 # Pattern matching rules that define how to build the derived files.
218 #
219 %_c.h: %_c.x
220         $(RM) $@
221         $(RPCGEN) -DUSINGC -h -o $@ $<

223 %.h: %_c.x
224         $(RM) $@
225         $(RPCGEN) -h -o $@ $<

227 %_c_xdr.c: %_c.x
228         $(RM) $@
229         $(RPCGEN) -DUSINGC -c -o $@ $<

231 # rename the xdr_db_free_entry() in the db_table_c_xdr.c.  So the one
232 # in db_index_entry_c is used instead.
233 DB_TABLE=db_table_c

235 $(DB_TABLE)_xdr.c:      $(DB_TABLE).x
236         $(RM) $@
237         $(RPCGEN) -DUSINGC -c $(DB_TABLE).x |\
238         $(SED) -e 's/^xdr_db_free_entry(/__OBSOLETED_xdr_db_free_entry(/' > $@

240 # include library targets
241 include ../Makefile.targ
```

```
**********************************************************
    9450 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/db_dictionary_c.x
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License, Version 1.0 only
   6  * (the "License").  You may not use this file except in compliance
   7  * with the License.
   8  *
   9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10  * or http://www.opensolaris.org/os/licensing.
  11  * See the License for the specific language governing permissions
  12  * and limitations under the License.
  13  *
  14  * When distributing Covered Code, include this CDDL HEADER in each
  15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16  * If applicable, add the following below this CDDL HEADER, with the
  17  * fields enclosed by brackets "[]" replaced with your own identifying
  18  * information: Portions Copyright [yyyy] [name of copyright owner]
  19  *
  20  * CDDL HEADER END
  21  */
  22 /*
  23  *      db_dictionary_c.x
  24  *
  25  * Copyright 2015 Gary Mills
  26  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  27  * Use is subject to license terms.
  28  */
  29
  30 #if RPC_XDR
  31 %#include "xdr_nullptr.h"
  32 #endif /* RPC_XDR */
  29 %#pragma ident  "%Z%%M% %I%     %E% SMI"
  33
  34 #if RPC_HDR
  35 %#ifndef _DB_DICTIONARY_H
  36 %#define _DB_DICTIONARY_H

  38 #ifdef USINGC
  39 %#include "nisdb_rw.h"
  40 %#include "nisdb_ldap.h"
  41 %#include "db_entry_c.h"
  42 %#include "db_scheme_c.h"
  43 %#include "db_vers_c.h"
  44 %#include "ldap_xdr.h"
  45 %typedef void *nullptr;
  46 %typedef u_int db_dict_version;
  47 #else
  48 %#include "nisdb_rw.h"
  49 %#include "nisdb_ldap.h"
  50 %#include "db_entry.h"
  51 %#include "db_scheme.h"
  52 %#include "db.h"
  53 %#include "db_vers.h"
  54 %#include "db_dictlog.h"
  55 %#include "ldap_xdr.h"
  56 #endif /* USINGC */
  57 #endif /* RPC_HDR */

  59 struct db_table_desc {
  60   string table_name<NIS_MAXNAMELEN>;
```



```
  61   u_long hashval;
  62   db_scheme * scheme;
  63 #ifdef USINGC
  64   nullptr database;   /* for XDR, keep database from descriptor */
  65 #else
  66   db *database;        /* for program use in c++ code */
  67 #endif /* USINGC */
  68   db_table_desc *next;
  69 };
_____unchanged_portion_omitted_
 110 typedef struct __nisdb_dictionary_defer_struct  __nisdb_dictionary_defer_t;
 111 #ifdef  USINGC
 112 %bool_t xdr___nisdb_dictionary_defer_t();
 113 #endif  /* USINGC */
 114 #endif

 116 #ifndef USINGC
 117 #ifdef RPC_HDR
 118 %class db_dictionary {
 119 %  db_dict_desc_p dictionary;
 120 %  bool_t initialized;
 121 %  char* filename;
 122 %  char* tmpfilename;
 123 %  char* logfilename;
 124 %  db_dictlog *logfile;
 125 %  bool_t logfile_opened;
 126 %  bool_t changed;
 127 %  __nisdb_dictionary_defer_t deferred;
 128 %  __nisdb_flag_t noWriteThrough;
 129 %  STRUCTRWLOCK(dict);
 130 %
 131 %/* Dump contents of this dictionary (minus the database representation)
 132 %     to its file. Returns 0 if operation succeeds, -1 otherwise. */
 133 %  int dump();
 134 %
 135 %/* Delete old log file and descriptor */
 136 %  int reset_log();
 137 %
 138 %/* Open log file (and creates descriptor) if it has not been opened */
 139 %  int open_log();
 140 %
 141 %/* Incorporate updates in log to dictionary already loaded.
 142 %   Does not affect "logfile" */
 143 %  int incorporate_log( char * );
 144 %
 145 %  /* closes log file if opened */
 146 %  int close_log();
 147 %
 148 %/*  Log the given action and execute it.
 149 %    The minor version of the dictionary is updated after the action has
 150 %    been executed and the dictionary is flagged as being changed.
 151 %    Return the structure db_result, or NULL if the loggin failed or the
 152 %    action is unknown. */
 153 %  db_status log_action(int, char* table, table_obj* tobj =0);
 154 %
 155 %  db_status create_table_desc(char* table_name, table_obj* table_desc,
 156 %                              db_table_desc**);
 157 %
 158 %  db_dict_desc_p db_copy_dictionary(void);
 159 %
 160 % public:
 161 %/* Constructor:  creates an empty, uninitialized dictionary. */
 162 %  db_dictionary();
 163 %
 164 %/* Destructor: noop. Use db_shutdown if you really want to clean up. */
 165 %  ~db_dictionary() {}
```

```
166 %
167 %  db_status merge_dict (db_dictionary&, char *, char *);
168 %
169 %  db_status massage_dict (char *, char *, char *);
170 %  int       db_clone_bucket (db_table_desc *, db_table_desc_p *);
171 %  int       change_table_name (db_table_desc *, char *, char *);
172 %  bool_t    extract_entries (db_dictionary&, char **, int );
173 %
174 %/* Real destructor: deletes filename and table descriptors */
175 %  db_status db_shutdown();
176 %
177 %/*  Initialize dictionary from contents in 'file'.
178 %    If there is already information in this dictionary, it is removed.
179 %    Therefore, regardless of whether the load from the file succeeds,
180 %    the contents of this dictionary will be altered.  Returns
181 %    whether table has been initialized successfully. */
182 %  bool_t init( char* fname );
183 %  bool_t inittemp( char* fname, db_dictionary&);
184 %
185 %/* closes any open log files for all tables in dictionary or 'tab'.
186 %   "tab" is an optional argument.
187 % */
188 %   db_status db_standby( char* tab = 0 );
189 %
190 %/* Write out in-memory copy of dictionary to file.
191 %   1.  Update major version.
192 %   2.  Dump contents to temporary file.
193 %   3.  Rename temporary file to real dictionary file.
194 %   4.  Remove log file.
195 %   A checkpoint is done only if it has changed since the previous checkpoint.
196 %   Returns TRUE if checkpoint was successful; FALSE otherwise. */
197 %  db_status checkpoint();
198 %
199 %/*  Checkpoints table specified by 'tab', or all tables if 'tab' is 0. */
200 %   db_status db_checkpoint( char* tab = 0 );

202 %/* Add table with given name 'tab' and description 'zdesc' to dictionary.
203 %   Returns error code if table already exists, or if no memory can be found
204 %   to store the descriptor, or if dictionary has not been intialized.
205 %   Dictionary is updated to stable store before addition.
206 %   Fatal error occurs if dictionary cannot be saved.
207 %   Returns DB_SUCCESS if dictionary has been updated successfully. */
208 %  db_status add_table_aux(char* table_name, table_obj* table_desc, int mode);
209 %
210 %/* Delete table with given name 'tab' from dictionary.
211 %   Returns error code if table does not exist or if dictionary has not been
212 %   initialized.   Dictionary is updated to stable store if deletion is
213 %   successful.  Fatal error occurs if dictionary cannot be saved.
214 %   Returns DB_SUCCESS if dictionary has been updated successfully.
215 %   Note that the files associated with the table are also removed.  */
216 %  db_status delete_table_aux( char* table_name, int mode );
217 %
218 %  db_status add_table( char* table_name, table_obj* table_desc );
219 %  int copyfile( char* infile, char *outfile);
220 %
221 %  db_status delete_table( char* table_name );
222 %
223 %/* Return database structure of table named by 'table_name'.
224 %   If 'where' is set, set it to the table_desc of 'table_name.'
225 %   The database is loaded in from stable store if it has not been loaded.
226 %   If it cannot be loaded, it is initialized using the scheme stored in
227 %   the table_desc.  NULL is returned if the initialization fails.   */
228 %  db* find_table( char* table_name, db_table_desc ** where = NULL );
229 %
230 %  db *find_table(char *table_name, db_table_desc **where,
231 %                       bool_t searchDeferred);
```



```
232 %  db *find_table(char *table_name, db_table_desc **where,
233 %                       bool_t searchDeferred, bool_t doLDAP,
234 %                       bool_t doLoad);
235 %
236 %  db *find_table_noLDAP(char *table_name, db_table_desc **where,
237 %                       bool_t searchDeferred, bool_t doLoad);
238 %
239 %/* Returns db_table_desc of table name 'tab'.
240 %   Use this if you do not want table to be loaded. */
241 %  db_table_desc * find_table_desc( char* table_name );
242 %
243 %  db_table_desc * find_table_desc(char *table_name, bool_t searchDeferred);
244 %
245 %/* Translate given nis attribute list to a db_query structure.
246 %   Return FALSE if dictionary has not been initialized, or
247 %   table does not have a scheme (which should be a fatal error?). */
248 %  db_query * translate_to_query( db_table_desc*, int, nis_attr * );
249 %
250 %/* Return an array of strings of table names of all tables in dictionary. */
251 %   db_table_names * get_table_names();
252 %
253 %/* Set/clear no-write-through flag */
254 %  void setNoWriteThrough(void);
255 %  void clearNoWriteThrough(void);
256 %
257 %/* Locking */
258 %  int acqexcl(void) {
259 %       return(WLOCK(dict));
260 %  }
261 %
262 %  int relexcl(void) {
263 %       return (WULOCK(dict));
264 %  }
265 %
266 %  int acqnonexcl(void) {
267 %       return (RLOCK(dict));
268 %  }
269 %
270 %  int relnonexcl(void) {
271 %       return (RULOCK(dict));
272 %  }
273 %
274 %/* Set deferred commit mode; intended for replica resync */
275 %  db_status defer(char *table);
276 %
277 %/* Commit deferred changes; intended for replica resync */
278 %  db_status commit(char *table);
279 %
280 %/* Roll back deferred changes; intended for replica resync */
281 %  db_status rollback(char *table);
282 %};
283 %#ifdef __STDC__
284 %extern "C" bool_t xdr_db_table_desc_p(XDR *, db_table_desc_p *);
285 %extern "C" bool_t xdr_db_table_desc(XDR *, db_table_desc *);
286 %extern "C" bool_t xdr_db_dict_desc_p(XDR *, db_dict_desc_p *);
287 %extern "C" bool_t xdr_db_table_namep(XDR *, db_table_namep *);
288 %extern "C" bool_t xdr_db_table_names(XDR *, db_table_names *);
289 %#endif

291 #endif /* RPC_HDR */
292 #endif /* USINGC */

294 #if RPC_HDR
295 %#endif /* _DB_DICTIONARY_H */
296 #endif /* RPC_HDR */
```

****
1964 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/db_dictxdr.c
5910 libnisdb won't build with modern GCC
****

```
 1 /*
 2  * CDDL HEADER START
 3  *
 4  * The contents of this file are subject to the terms of the
 5  * Common Development and Distribution License, Version 1.0 only
 6  * (the "License").  You may not use this file except in compliance
 7  * with the License.
 8  *
 9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10  * or http://www.opensolaris.org/os/licensing.
11  * See the License for the specific language governing permissions
12  * and limitations under the License.
13  *
14  * When distributing Covered Code, include this CDDL HEADER in each
15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16  * If applicable, add the following below this CDDL HEADER, with the
17  * fields enclosed by brackets "[]" replaced with your own identifying
18  * information: Portions Copyright [yyyy] [name of copyright owner]
19  *
20  * CDDL HEADER END
21  */
22 /*
23  *      db_dictxdr.c
24  *
25  * Copyright 2015 Gary Mills
26  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
27  * Use is subject to license terms.
28  */

29 #pragma ident   "%Z%%M% %I%     %E% SMI"

30 #include "db_dictionary_c.h"
31 #include "db_dictionary.h"
32 #include "db_vers_c.h"

34 extern vers db_update_version;

36 extern void make_zero(vers*);

38 /* Special xdr_db_dict_desc that understands optional version number at end. */
39 bool_t
40 xdr_db_dict_desc(XDR *xdrs, db_dict_desc *objp)
41 {

43         if (!xdr_db_dict_version(xdrs, &objp->impl_vers))
44                 return (FALSE);
45         if (!xdr_array(xdrs, (char **)&objp->tables.tables_val,
46                 (uint_t *)&objp->tables.tables_len, ~0,
47                 sizeof (db_table_desc_p), (xdrproc_t)xdr_db_table_desc_p))
48                 return (FALSE);
49         if (!xdr_int(xdrs, &objp->count))
50                 return (FALSE);

52         if (xdrs->x_op == XDR_DECODE) {
53                 /* If no version was found, set version to 0. */
54                 if (!xdr_vers(xdrs, (void**) &db_update_version))
55                         make_zero(&db_update_version);
56                 return (TRUE);
57         } else if (xdrs->x_op == XDR_ENCODE) {
58                 /* Always write out version */
59                 if (!xdr_vers(xdrs, (void**) &db_update_version))
```



```
60                         return (FALSE);
61         } /* else XDR_FREE: do nothing */

63         return (TRUE);
64 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    3227 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/db_entry_c.x
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 *      db_entry_c.x
 *
 * Copyright 2015 Gary Mills
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

%#pragma ident  "%Z%%M% %I%     %E% SMI"

/*
 * Some manifest constants, chosen to maximize flexibility without
 * plugging the wire full of data.
 */

#if RPC_HDR
%#ifndef _DB_NIS_H
%#define _DB_NIS_H

%#include <rpcsvc/nis.h>
#endif /* RPC_HDR */

#if RPC_XDR
%#include "nis_clnt.h"
#endif /* RPC_XDR */

#ifdef USINGC
enum db_status {DB_SUCCESS, DB_NOTFOUND, DB_NOTUNIQUE,
                DB_BADTABLE, DB_BADQUERY, DB_BADOBJECT,
                DB_MEMORY_LIMIT, DB_STORAGE_LIMIT, DB_INTERNAL_ERROR};

enum db_action {DB_LOOKUP, DB_REMOVE, DB_ADD, DB_FIRST, DB_NEXT,
                        DB_ALL, DB_RESET_NEXT, DB_ADD_NOLOG,
                        DB_ADD_NOSYNC, DB_REMOVE_NOSYNC };
#endif /* USINGC */

/* Make alias to NIS definition */

typedef entry_obj entry_object;
typedef entry_object * entry_object_p;
```



```
typedef nis_name db_stringname;
typedef nis_attr db_attrname;          /* What the database knows it as */


/*  nis_dba.x ----------------------------- */

/*
 * Structure definitions for the parameters and results of the actual
 * NIS DBA calls
 *
 * This is the standard result (in the protocol) of most of the nis
 * requests.
 */

/*typedef long db_next_desc;*/

typedef opaque db_next_desc<>;            /* opaque string */

struct db_result {
        db_status       status;         /* The status itself     */
        db_next_desc    nextinfo;       /* for first/next sequence */
        entry_object_p  objects<>;      /* And the objects found */
        long            ticks;          /* for statistics        */
};
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4659 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/db_index_c.x
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 *      db_index_c.x
 *
 *  Copyright 2015 Gary Mills
 *  Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 *  Use is subject to license terms.
 */

#if RPC_XDR
%#include "ldap_xdr.h"
#endif /* RPC_XDR */

#if RPC_HDR
%#ifndef _DB_INDEX_H
%#define _DB_INDEX_H

%
%/* db_index is a hash table with separate overflow buckets. */
%


#ifdef USINGC
%#include "db_item_c.h"
%#include "db_index_entry_c.h"
%#include "db_table_c.h"
%#include "db_scheme_c.h"
#else
%#include "db_item.h"
%#include "db_index_entry.h"
%#include "db_table.h"
%#include "db_scheme.h"
#endif /* USINGC */
#endif /* RPC_HDR */
%
%#include "nisdb_rw.h"
%
#if RPC_HDR || RPC_XDR
#ifdef USINGC
struct db_index {
  db_index_entry_p tab<>;
  int count;
```



```
  bool case_insens;
  __nisdb_rwlock_t index_rwlock;
};
______unchanged_portion_omitted_
```

```
**********************************************************
    4762 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/db_index_entry_c.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2015 Gary Mills
 */

/*
 * This is a non-recursive version of XDR routine used for db_index_entry
 * type.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/syslog.h>
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#include <memory.h>
#include "db_index_entry_c.h"
#include "db_table_c.h"
#include "xdr_nullptr.h"

bool_t
xdr_db_index_entry(xdrs, objp)
        register XDR *xdrs;
        db_index_entry *objp;
{
        bool_t  more_data;
        register db_index_entry *ep = objp;
        register db_index_entry *loc;
        register db_index_entry *freeptr = NULL;

        for (;;) {
                if (!xdr_u_long(xdrs, &ep->hashval))
                        return (FALSE);
                if (!xdr_pointer(xdrs, (char **)&ep->key, sizeof (item),
                        (xdrproc_t) xdr_item))
                        return (FALSE);
                if (!xdr_entryp(xdrs, &ep->location))
                        return (FALSE);
```



```
                if (!xdr_nullptr(xdrs, &ep->next_result))
                        return (FALSE);

                /*
                 * The following code replaces the call to
                 * xdr_pointer(
                 *      xdrs,
                 *      (char **)&ep->next,
                 *      sizeof (db_index_entry),
                 *      (xdrproc_t) xdr_db_index_entry))
                 *
                 * It's a modified version of xdr_refer.c from the rpc library:
                 *      @(#)xdr_refer.c         1.8     92/07/20 SMI
                 */


                /*
                 * the following assignment to more_data is only useful when
                 * encoding and freeing.  When decoding, more_data will be
                 * filled by the xdr_bool() routine.
                 */
                more_data = (ep->next != NULL);
                if (! xdr_bool(xdrs, &more_data))
                        return (FALSE);
                if (! more_data) {
                        ep->next = NULL;
                        break;
                }

                loc = ep->next;


                switch (xdrs->x_op) {
                case XDR_DECODE:
                        if (loc == NULL) {
                                ep->next = loc = (db_index_entry *)
                                        mem_alloc(sizeof (db_index_entry));
                                if (loc == NULL) {
                                        syslog(LOG_ERR,
                                "xdr_db_index_entry: mem_alloc failed");
                                        return (FALSE);
                                }
                                memset(loc, 0, sizeof (db_index_entry));
                        }
                        break;
                case XDR_FREE:
                        if (freeptr != NULL) {
                                mem_free(freeptr, sizeof (db_index_entry));
                        } else
                                ep->next = NULL;
                        freeptr = loc;
                        break;
                }

                if (loc == NULL)
                        break;
                ep = loc;
        }       /* for loop */

        if ((freeptr != NULL) && (xdrs->x_op == XDR_FREE)) {
                mem_free(freeptr, sizeof (db_index_entry));
        }

        return (TRUE);
}
```

_____***unchanged_portion_omitted_***

****
25635 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/db_mindex3.cc
5910 libnisdb won't build with modern GCC
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2015 RackTop Systems.
 */

#include <sys/types.h>
#include <time.h>
#include <sys/time.h>
#include <lber.h>
#include <ldap.h>
#include <signal.h>
#include <pthread.h>
#include "db_headers.h"
#include "db.h"
#include "db_mindex.h"
#include "db_dictionary.h"
#include "nisdb_mt.h"
#include "ldap_map.h"
#include "ldap_glob.h"
#include "ldap_util.h"


extern db_dictionary    *InUseDictionary;


extern "C" {

typedef struct {
        db_mindex               *mindex;
        __nis_table_mapping_t   *t;
        db_query                *qin;
        db_query                *q;
        char                    *dbId;
        nis_object              *dirObj;
        int                     isDeferred;
        char                    *tableName;
} __entries_from_ldap_arg_t;
```

*_____unchanged_portion_omitted_*


```
#endif  /* SET_ENTRY_FLAGS */

static void                     setOid(nis_object *obj);

/*
 * Retrieve container entries from LDAP per 't' and 'qin'/'q'.
 * This is a helper function for db_mindex::queryLDAP(); see
 * that function for details of the parameters (except doAsynch).
 *
 * If 'doAsynch' is set, and the retrieval is an enumeration
 * (qin == NULL), the retrieval is performed in a detached
 * thread. In this case, the return code just reflects the
 * setup and launch of the detached thread. Retrieval will
 * complete asynchronously.
 */
int
db_mindex::entriesFromLDAP(__nis_table_mapping_t *t, db_query *qin, db_query *q,
                        char *dbId, nis_object *dirObj, int doAsynch) {
        __entries_from_ldap_arg_t       *arg;
        int                             stat;
        db_status                       dstat;
        const char                      *myself = "db_mindex::entriesFromLDAP";

        arg = (__entries_from_ldap_arg_t *)am(myself, sizeof (*arg));
        if (arg == 0) {
                freeQuery(q);
                if (dirObj != 0)
                        nis_destroy_object(dirObj);
                return (LDAP_NO_MEMORY);
        }

        arg->mindex = this;
        arg->t = t;
        arg->qin = qin;
        arg->q = q;
        arg->dbId = dbId;
        arg->dirObj = dirObj;
        arg->tableName = t->objName;

        /*
         * Check if an enumeration thread is running; if so, then regardless
         * of whether or not the current operation is an enumeration, we
         * just return success, and let our caller get the data from the
         * existing (deferred) DB.
         */
        (void) mutex_lock(&table->mapping.enumLock);
        if (table->mapping.enumTid != 0) {
                int     doReturn = 0;

                stat = pthread_kill(table->mapping.enumTid, 0);
                if (stat == ESRCH) {
                        logmsg(MSG_NOTIMECHECK, LOG_WARNING,
        "%s: Enumeration thread %d not found for \"%s\"; exit status = %d (%s)",
                                myself, table->mapping.enumTid,
                                NIL(t->objName), table->mapping.enumStat,
                                ldap_err2string(table->mapping.enumStat));
                        /* Reflect the fact that no enum thread is running */
                        table->mapping.enumTid = 0;
                        table->mapping.enumStat = -1;
                        /* Cleanup deferred mode */
                        if (table->mapping.enumDeferred) {
                                dstat = InUseDictionary->commit(t->objPath);
                                if (dstat == DB_SUCCESS) {
                                        table->mapping.enumDeferred = 0;
                                } else {
                                        logmsg(MSG_NOTIMECHECK, LOG_ERR,
```

```
150                                     "%s: DB error %d committing \"%s\"",
151                                             myself, dstat, NIL(t->objName));
152                             }
153                     }
154             } else if (stat == 0) {
155                     logmsg(MSG_NOTIMECHECK, LOG_INFO,
156                     "%s: Enumeration thread %d already running for \"%s\"",
157                             myself, table->mapping.enumTid,
158                             NIL(t->objName));
159                     stat = LDAP_SUCCESS;
160                     doReturn = 1;
161             } else {
162                     logmsg(MSG_NOTIMECHECK, LOG_INFO,
163             "%s: Error %d looking for enumeration thread %d for \"%s\"",
164                             myself, stat, table->mapping.enumTid,
165                             NIL(t->objName));
166                     doReturn = 1;
167                     stat = LDAP_OPERATIONS_ERROR;
168             }
169             if (doReturn) {
170                     (void) mutex_unlock(&table->mapping.enumLock);
171                     sfree(arg);
172                     freeQuery(q);
173                     if (dirObj != 0)
174                             nis_destroy_object(dirObj);
175                     return (stat);
176             }
177     }

179     /*
180      * If we're enumerating (and hence expect that retrieving all data,
181      * and updating the local DB, might take a while), create a deferred-
182      * update table that clients can use while we are updating the real
183      * one.
184      */
185     if (doAsynch && qin == 0) {
186             if ((dstat = InUseDictionary->defer(t->objPath)) ==
187                             DB_SUCCESS) {
188                     arg->isDeferred = 1;
189                     table->mapping.enumDeferred = 1;
190             } else {
191                     logmsg(MSG_NOTIMECHECK, LOG_WARNING,
192                     "%s: Unable to defer updates for \"%s\" (status=%d);"
193                                     " updating in place",
194                             myself, NIL(t->objName), dstat);
195                     arg->isDeferred = 0;
196                     table->mapping.enumDeferred = 0;
197             }
198     } else {
199             arg->isDeferred = 0;
200             table->mapping.enumDeferred = 0;
201     }

203     /* If enumerating, perform the operation in a separate thread */
204     if (doAsynch && qin == 0) {
205             pthread_t       tid;
206             pthread_attr_t  attr;

208             (void) pthread_attr_init(&attr);
209 #ifdef  FORCE_SYNCHRONOUS
210 #else
211             (void) pthread_attr_setdetachstate(&attr,
212                                                PTHREAD_CREATE_DETACHED);
213 #endif  /* FORCE_SYNCHRONOUS */
214             stat = pthread_create(&tid, &attr, entriesFromLDAPthread, arg);
215             if (stat != 0) {
```



```
216                     (void) mutex_unlock(&table->mapping.enumLock);
217                     logmsg(MSG_NOTIMECHECK, LOG_WARNING,
218                     "%s: Error %d creating new thread; using current one",
219                             myself, stat);
220                     stat = entriesFromLDAPreal(arg);
217                     stat = (int)entriesFromLDAPthread(arg);
221                     return (stat);
222             }

224             table->mapping.enumTid = tid;
225             table->mapping.enumStat = -1;

227             /*
228              * We're now returning to the caller, who will get data
229              * from:
230              *
231              *      The deferred DB, if an enumeration thread already
232              *      was running, and deferred mode was on, or
233              *
234              *      The original DB, if we just started an enumeration
235              *      thread. In this case, our caller (several levels up)
236              *      is holding a lock on the db_mindex/db_table, which
237              *      means that the enum thread will have to wait for
238              *      our caller once it's done the LDAP retrieval, and
239              *      wants to update the DB.
240              */
241             (void) mutex_unlock(&table->mapping.enumLock);
242             stat = LDAP_SUCCESS;
243 #ifdef  FORCE_SYNCHRONOUS
244             {
245                     int     tstat;

247                     stat = pthread_join(tid, (void **)&tstat);
248                     if (stat == 0) {
249                             stat = tstat;
250                             logmsg(MSG_NOTIMECHECK, LOG_WARNING,
251                                     "%s: thread %d => %d",
252                                     myself, tid, tstat);
253                     } else {
254                             logmsg(MSG_NOTIMECHECK, LOG_ERR,
255                                     "%s: pthread_join(%d) => %d",
256                                     myself, tid, stat);
257                             stat = LDAP_OPERATIONS_ERROR;
258                     }
259             }
260 #endif  /* FORCE_SYNCHRONOUS */
261     } else {
262             (void) mutex_unlock(&table->mapping.enumLock);
263             stat = entriesFromLDAPreal(arg);
260             stat = (int)entriesFromLDAPthread(arg);
264     }

266     return (stat);
267 }

269 extern "C" {

271 /*
272  * We use this 'extern "C"' function in order to make sure that
273  * pthread_create() doesn't have any problems trying to invoke a
274  * C++ function.
275  */
276 static void *
277 entriesFromLDAPthread(void *voidarg) {
278         __entries_from_ldap_arg_t       *arg;
276         int                             stat;
```

```
279         db                              *dbase;
280         db_table_desc                   *tbl = 0;
281         char                            *tableName;

283         arg = (__entries_from_ldap_arg_t *)voidarg;

285         /* Lock to prevent removal */
286         (void) __nis_lock_db_table(arg->tableName, 1, 0,
287                                         "entriesFromLDAPthread");

289         /*
290          * It's possible that the db_mindex for the table has changed,
291          * or disappeared, between now and the time when our parent
292          * thread released its lock on the table. Hence, we search the
293          * dictionary to re-acquire the 'db', and the db_mindex.
294          */
295         tableName = internalTableName(arg->tableName);
296         if (tableName != 0) {
297 #ifdef  NISDB_LDAP_DEBUG
298                 db_mindex       *oldMindex = arg->mindex;
299 #endif  /* NISDB_LDAP_DEBUG */

301                 dbase = InUseDictionary->find_table(tableName, &tbl, FALSE);
302                 if (dbase != 0)
303                         arg->mindex = dbase->mindex();
304                 else
305                         arg->mindex = 0;
306 #ifdef  NISDB_LDAP_DEBUG
307                 logmsg(MSG_NOTIMECHECK, LOG_WARNING,
308                         "entriesFromLDAPthread: %s -> %s -> 0x%x (0x%x)",
309                         NIL(arg->tableName), NIL(tableName),
310                         arg->mindex, oldMindex);
311 #endif  /* NISDB_LDAP_DEBUG */
312                 sfree(tableName);
313                 tableName = 0;
314         }

316         (void) entriesFromLDAPreal(arg);
314         stat = entriesFromLDAPreal(arg);

318         (void) __nis_ulock_db_table(arg->tableName, 1, 0,
319                                         "entriesFromLDAPthread");

321         freeQuery(arg->q);
322         if (arg->dirObj != 0)
323                 nis_destroy_object(arg->dirObj);
324         sfree(arg);
325         return (NULL);
323         return ((void *)stat);
326 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   12550 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/db_mindex_c.x
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  *      db_mindex_c.x
  23  *
  24  * Copyright 2015 Gary Mills
  25  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  26  * Use is subject to license terms.
  27  */

  29 #if RPC_XDR
  30 %#include "ldap_xdr.h"
  31 %#include "nis_clnt.h"
  32 #endif /* RPC_XDR */

  34 #if RPC_HDR
  35 %#ifndef _DB_MINDEX_H
  36 %#define _DB_MINDEX_H

  38 #ifdef USINGC
  39 %#include "db_vers_c.h"
  40 %#include "db_table_c.h"
  41 %#include "db_index_entry_c.h"
  42 %#include "db_index_c.h"
  43 %#include "db_scheme_c.h"
  44 %#include "db_query_c.h"
  45 #else
  46 %#include "db_vers.h"
  47 %#include "db_table.h"
  48 %#include "db_index_entry.h"
  49 %#include "db_index.h"
  50 %#include "db_scheme.h"
  51 %#include "db_query.h"
  52 #endif /* USINGC */
  53 %#include "ldap_parse.h"
  54 %#include "nisdb_rw.h"
  55 %#include "ldap_xdr.h"
  56 #endif /* RPC_HDR */

  58 #if RPC_HDR
  59 %struct db_next_index_desc {
  60 %  entryp location;
  61 %  struct db_next_index_desc *next;
```



```
  63 #ifndef USINGC
  64 %  db_next_index_desc( entryp loc, struct db_next_index_desc *n )
  65 %    { location = loc; next = n; }
  66 #endif /* USINGC */

  68 %};
  69 #endif /* RPC_HDR */


  72 #if RPC_HDR || RPC_XDR
  73 #ifdef USINGC

  75 struct db_mindex {
  76   vers rversion;
  77   db_index indices<>;                /* indices[num_indices] */
  78   db_table *table;
  79   db_scheme *scheme;
  80   __nisdb_ptr_t objPath;
  81   __nisdb_flag_t noWriteThrough;
  82   __nisdb_flag_t noLDAPquery;
  83   __nisdb_flag_t initialLoad;
  84   __nisdb_ptr_t dbptr;
  85   __nisdb_rwlock_t mindex_rwlock;
  86 };
_____unchanged_portion_omitted_
  96 typedef struct xdr_nis_object_s xdr_nis_object_t;

  92 %extern bool_t  xdr_nis_object();
  98 #endif /* USINGC */
  99 #endif /* RPC_HDR */

 101 #ifndef USINGC
 102 #ifdef RPC_HDR
 103 %
 104 %struct xdr_nis_object_s {
 105 %       int                             version;
 106 %       nis_object                      *obj;
 107 %       struct {
 108 %               uint_t  dirEntry_len;
 109 %               char    **dirEntry_val;
 110 %       }                               dirEntry;
 111 %};
 112 %typedef struct xdr_nis_object_s        xdr_nis_object_t;
 113 %
 109 %extern bool_t  xdr_nis_object();
 110 %
 114 %class db_mindex {
 115 %  vers rversion;
 116 %//  int num_indices;
 117 %//  db_index * indices;               /* indices[num_indices] */
 118 %  struct {
 119 %   int indices_len;
 120 %   db_index *indices_val;
 121 %  } indices;
 122 %  db_table *table;
 123 %  db_scheme *scheme;
 124 %  __nisdb_ptr_t objPath;
 125 %  __nisdb_flag_t noWriteThrough;
 126 %  __nisdb_flag_t noLDAPquery;
 127 %  __nisdb_flag_t initialLoad;
 128 %  __nisdb_ptr_t dbptr;
 129 %  STRUCTRWLOCK(mindex);
 130 %
 131 %/* Return a list of index_entries that satsify the given query 'q'.
 132 %   Return the size of the list in 'count'. Return NULL if list is empty.
```

```
133 %   Return in 'valid' FALSE if query is not well formed. */
134 %  db_index_entry_p satisfy_query(db_query *, long *, bool_t *valid,
135 %                                 bool_t fromLDAP = FALSE);
136 %  db_index_entry_p satisfy_query(db_query *, long *, bool_t *valid = NULL);
137 %
138 %/* Returns a newly db_query containing the index values as
139 %   obtained from the given object.  The object itself,
140 %   along with information on the scheme given, will determine
141 %   which values are extracted from the object and placed into the query.
142 %   Returns an empty query if 'obj' is not a valid entry.
143 %   Note that space is allocated for the query and the index values
144 %   (i.e. do not share pointers with strings in 'obj'.) */
145 %  db_query * extract_index_values_from_object( entry_object * );
146 %
147 %/* Returns a newly created db_query structure containing the index values
148 %   as obtained from the record named by 'recnum'.  The record itself, along
149 %   with information on the schema definition of this table, will determine
150 %   which values are extracted from the record and placed into the result.
151 %   Returns NULL if recnum is not a valid entry.
152 %   Note that space is allocated for the query and the index values
153 %   (i.e. do not share pointers with strings in 'obj'.) */
154 %  db_query * extract_index_values_from_record( entryp );
155 %
156 %/* Returns an array of size 'count' of 'entry_object_p's, pointing to
157 %   copies of entry_objects named by the result list of db_index_entries 'res'.
158 %*/
159 %  entry_object_p * prepare_results( int, db_index_entry_p, db_status* );
160 %
161 %/* Remove the entry identified by 'recloc' from:
162 %   1.  all indices, as obtained by extracting the index values from the entry
163 %   2.  table where entry is stored. */
164 %  db_status remove_aux( entryp );
165 %
166 %/*  entry_object * get_record( entryp );*/
167 % public:
168 %
169 %/* Constructor:  Create empty table (no scheme, no table or indices). */
170 %  db_mindex();
171 %
172 %/* Constructor:  Create new table using scheme defintion supplied.
173 %   (Make copy of scheme and keep it with table.) */
174 %  db_mindex(db_scheme *, char *tablePath);
175 %
176 %/* destructor */
177 %  ~db_mindex();
178 %
179 %  db_index_entry_p satisfy_query_dbonly(db_query *, long *,
180 %                                        bool_t checkExpire,
181 %                                        bool_t *valid = NULL);
182 %
183 %/* Returns whether there table is valid (i.e. has scheme). */
184 %  bool_t good() { return scheme != NULL && table != NULL; }
185 %
186 %/* Change the version of the table to the one given. */
187 %  void change_version( vers *v ) {  rversion.assign( v );}
188 %
189 %/* Return the current version of the table. */
190 %  vers *get_version()  { return( &rversion ); }
191 %
192 %/* Reset contents of tables by: deleting indice entries, table entries */
193 %  void reset_tables();
194 %
195 %/* Reset the table by: deleting all the indices, table of entries, and its
196 %   scheme. Reset version to 0 */
197 %  void reset();
198 %
```



```
199 %/* Initialize table using information from specified file.
200 %   The table is first 'reset', then the attempt to load from the file
201 %   is made.  If the load failed, the table is again reset.
202 %   Therefore, the table will be modified regardless of the success of the
203 %   load.  Returns TRUE if successful, FALSE otherwise. */
204 %  int load( char * );
205 %
206 %/* Initialize table using information given in scheme 'how'.
207 %   Record the scheme for later use (make copy of it);
208 %   create the required number of indices; and create table for storing
209 %   entries.
210 %   The 'tablePath' is passed on to db_table in order to obtain the
211 %   NIS+/LDAP mapping information (if any). */
212 %  void init( db_scheme *);
213 %
214 %/* Write this structure (table, indices, scheme) into the specified file. */
215 %  int dump( char *);
216 %
217 %/* Removes the entry in the table named by given query 'q'.
218 %   If a NULL query is supplied, all entries in table are removed.
219 %   Returns DB_NOTFOUND if no entry is found.
220 %   Returns DB_SUCCESS if one entry is found; this entry is removed from
221 %   its record storage, and it is also removed from all the indices of the
222 %   table. If more than one entry satisfying 'q' is found, all are removed. */
223 %  db_status remove( db_query *);
224 %
225 %/* Add copy of given entry to table.  Entry is identified by query 'q'.
226 %   The entry (if any) satisfying the query is first deleted, then
227 %   added to the indices (using index values extracted form the given entry)
228 %   and the table.
229 %   Returns DB_NOTUNIQUE if more than one entry satisfies the query.
230 %   Returns DB_NOTFOUND if query is not well-formed.
231 %   Returns DB_SUCCESS if entry can be added.  */
232 %  db_status add( db_query *, entry_object* );
233 %
234 %
235 %/* Finds entry that satisfy the query 'q'.  Returns the answer by
236 %   setting the pointer 'rp' to point to the list of answers.
237 %   Note that the answers are pointers to copies of the entries.
238 %   Returns the number of answers find in 'count'.
239 %   Returns DB_SUCCESS if search found at least one answer;
240 %   returns DB_NOTFOUND if none is found. */
241 %  db_status lookup( db_query *, long *, entry_object_p ** );
242 %
243 %/* Returns the next entry in the table after 'previous' by setting 'answer' to
244 %   point to a copy of the entry_object.  Returns DB_SUCCESS if 'previous'
245 %   is valid and next entry is found; DB_NOTFOUND otherwise.  Sets 'where'
246 %   to location of where entry is found for input as subsequent 'next'
247 %   operation. */
248 %  db_status next( entryp, entryp *, entry_object ** );
249 %
250 %/* Returns the next entry in the table after 'previous' by setting 'answer' to
251 %   point to a copy of the entry_object.  Returns DB_SUCCESS if 'previous'
252 %   is valid and next entry is found; DB_NOTFOUND otherwise.  Sets 'where'
253 %   to location of where entry is found for input as subsequent 'next'
254 %   operation. */
255 %  db_status next( db_next_index_desc*, db_next_index_desc **, entry_object ** )
256 %
257 %/* Returns the first entry found in the table by setting 'answer' to
258 %   a copy of the entry_object.  Returns DB_SUCCESS if found;
259 %   DB_NOTFOUND otherwise.  */
260 %  db_status first( entryp*, entry_object ** );
261 %
262 %/* Returns the first entry that satisfies query by setting 'answer' to
263 %   a copy of the entry_object.  Returns DB_SUCCESS if found;
264 %   DB_NOTFOUND otherwise.  */
```

```
265 %  db_status first( db_query *, db_next_index_desc **, entry_object ** );
266 %
267 % /* Delete the given list of results; used when no longer interested in
268 %    the results of the first/next query that returned this list.     */
269 %  db_status reset_next( db_next_index_desc *orig );
270 %
271 %/* Return all entries within table.  Returns the answer by
272 %   setting the pointer 'rp' to point to the list of answers.
273 %   Note that the answers are pointers to copies of the entries.
274 %   Returns the number of answers find in 'count'.
275 %   Returns DB_SUCCESS if search found at least one answer;
276 %   returns DB_NOTFOUND if none is found. */
277 %  db_status all( long *, entry_object_p ** );
278 %
279 %  /* for debugging */
280 %/* Prints statistics of the table.  This includes the size of the table,
281 %   the number of entries, and the index sizes. */
282 %  void print_stats();
283 %
284 %/* Prints statistics about all indices of table. */
285 %  void print_all_indices();
286 %
287 %
288 %/* Prints statistics about indices identified by 'n'. */
289 %  void print_index( int n );
290 %
291 %/* Configure LDAP mapping */
292 %  bool_t configure (char *objName);
293 %
294 %/* Mark this instance deferred */
295 %  void markDeferred(void) {
296 %       if (table != NULL) table->markDeferred();
297 %  }
298 %/* Remove deferred mark */
299 %  void unmarkDeferred(void) {
300 %       if (table != NULL) table->unmarkDeferred();
301 %  }
302 %
303 %/* Retrieve, remove, or store data from/in/to LDAP */
304 %  int queryLDAP(db_query *, char *, int);
305 %  int entriesFromLDAP(__nis_table_mapping_t *, db_query *, db_query *,
306 %                      char *, nis_object *, int);
307 %
308 %  int removeLDAP(db_query *, nis_object *o);
309 %
310 %  int storeObjLDAP(__nis_table_mapping_t *t, nis_object *o);
311 %  int storeLDAP(db_query *, entry_obj *, nis_object *, entry_obj *,
312 %               char *dbId);
313 %
314 %/* Set/clear no-write-through flag */
315 %  void setNoWriteThrough(void);
316 %  void clearNoWriteThrough(void);
317 %
318 %/* Set/clear no-LDAP-query flag */
319 %  void setNoLDAPquery(void);
320 %  void clearNoLDAPquery(void);
321 %
322 %/* Set/clear initialLoad flag */
323 %  void setInitialLoad(void);
324 %  void clearInitialLoad(void);
325 %
326 %/* Store/retrieve pointer to parent 'db' class instance */
327 %  void setDbPtr(void *ptr);
328 %  void *getDbPtr(void);
329 %
330 %/* Get pointer to private 'table' field */
```



```
331 %  db_table *getTable(void);
332 %
333 %/*
334 % * Update table entry per the (entry_object *). If 'replace' is set,
335 % * the entry is replaced or added; otherwise, it is removed.
336 % */
337 %  int updateTableEntry(entry_object *e, int replace, char *tableName,
338 %                       nis_object *obj, nis_object *tobj, uint32_t ttime,
339 %                       int *xid);
340 %
341 %/* Touch the indicated entry */
342 %  bool_t touchEntry(entry_object *e);
343 %  bool_t touchEntry(db_query *q);
344 %
345 %/* Return the 'scheme' pointer */
346 %  db_scheme *getScheme(void) {return (scheme);}
347 %
348 %/* RW lock functions */
349 %
350 %  int tryacqexcl(void) {
351 %       return (TRYWLOCK(mindex));
352 %  }
353 %
354 %  int acqexcl(void) {
355 %       return (WLOCK(mindex));
356 %  }
357 %
358 %  int relexcl(void) {
359 %       return (WULOCK(mindex));
360 %  }
361 %
362 %  int acqnonexcl(void) {
363 %       return (RLOCK(mindex));
364 %  }
365 %
366 %  int relnonexcl(void) {
367 %       return (RULOCK(mindex));
368 %  }
369 %};
370 %#ifdef __cplusplus
371 %extern "C" bool_t xdr_db_mindex(XDR*, db_mindex*);
372 %#elif __STDC__
373 %extern bool_t xdr_db_mindex(XDR*, db_mindex*);
374 %#endif
375 %typedef class db_mindex * db_mindex_p;
376 #endif /* RPC_HDR */
377 #endif /* USINGC */

379 #if RPC_HDR
380 %#endif /* _DB_MINDEX_H */
381 #endif /* RPC_HDR */
```

```
**********************************************************
    4253 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/db_scheme_c.x
5910 libnisdb won't build with modern GCC
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 *      db_scheme_c.x
 *
 * Copyright 2015 Gary Mills
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#if RPC_XDR
%#include "ldap_xdr.h"
#endif /* RPC_XDR */
%#pragma ident  "%Z%%M% %I%     %E% SMI"

#if RPC_HDR
%#ifndef _DB_SCHEMA_H
%#define _DB_SCHEMA_H

#ifdef USINGC
%#include "db_item_c.h"
%#include "db_entry_c.h"
#else
%#include "db_item.h"
%#include "db_entry.h"
#endif /* USINGC */

const DB_KEY_CASE = TA_CASE;

#endif /* RPC_HDR */
%
%#include "nisdb_rw.h"
%
%/* Positional information of where field starts within record
%   and its maximum length in terms of bytes. */
struct db_posn_info {
  short int start_column;
  short int max_len;
};
_____unchanged_portion_omitted_
```

```
**********************************************************
   29213 Fri Jul 24 12:28:10 2015
new/usr/src/lib/libnisdb/db_table.cc
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 *      db_table.cc
 *
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2015 RackTop Systems.
 */

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <stdlib.h>             /* srand48() */
#include <lber.h>
#include <ldap.h>
#include "db_headers.h"
#include "db_table.h"
#include "db_pickle.h"    /* for dump and load */
#include "db_entry.h"
#include "nisdb_mt.h"

#include "ldap_parse.h"
#include "ldap_util.h"
#include "ldap_map.h"
#include "ldap_xdr.h"
#include "nis_hashitem.h"
#include "nisdb_ldap.h"
#include "nis_parse_ldap_conf.h"

static time_t   maxTimeT;

/*
 * Find the largest (positive) value of time_t.
 *
 * If time_t is unsigned, the largest possible value is just ~0.
 * However, if it's signed, then ~0 is negative. Since lint (for
 * sure), and perhaps the compiler too, dislike comparing an
 * unsigned quantity to see if it's less than zero, we compare
 * to one instead. If negative, the largest possible value is
 * th inverse of 2**(N-1), where N is the number of bits in a
```



```
 * time_t.
 */
extern "C" {
static void
__setMaxTimeT(void)
{
        unsigned char   b[sizeof (time_t)];
        int             i;

        /* Compute ~0 for an unknown length integer */
        for (i = 0; i < sizeof (time_t); i++) {
                b[i] = 0xff;
        }
        /* Set maxTimeT to ~0 of appropriate length */
        (void) memcpy(&maxTimeT, b, sizeof (time_t));

        if (maxTimeT < 1)
                maxTimeT = ~(1L<<((8*sizeof (maxTimeT))-1));
                maxTimeT = ~(1<<((8*sizeof (maxTimeT))-1));
}
_____unchanged_portion_omitted_
```

****
3266 Fri Jul 24 12:28:11 2015
new/usr/src/lib/libnisdb/db_vers_c.x
5910 libnisdb won't build with modern GCC
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 *      db_vers_c.x
 *
 * Copyright 2015 Gary Mills
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#if RPC_XDR
%#include "ldap_xdr.h"
#endif /* RPC_XDR */
%#pragma ident  "%Z%%M% %I%     %E% SMI"

#if RPC_HDR
%#ifndef _DB_VERS_H
%#define _DB_VERS_H
#endif /* RPC_HDR */

%/* 'vers' is the version identifier.  */

%
%#include "nisdb_rw.h"
%
#if RPC_HDR || RPC_XDR
#ifdef USINGC
struct vers {
        u_int vers_high;
        u_int vers_low;
        u_int time_sec;
        u_int time_usec;
        __nisdb_rwlock_t vers_rwlock;
};
```

*unchanged_portion_omitted_*

```
**********************************************************
    9580 Fri Jul 24 12:28:11 2015
new/usr/src/lib/libnisdb/ldap_attr.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2001-2003 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/systeminfo.h>
#include <strings.h>
#include <rpcsvc/nis.h>

#include "nis_parse_ldap_conf.h"

#include "ldap_attr.h"
#include "ldap_util.h"
#include "ldap_structs.h"


/*
 * If 'name' doesn't end in a trailing dot, return a copy with the
 * value of "nisplusLDAPbaseDomain" appended. Otherwise, return a
 * copy of 'name'. If deallocate!=0, free 'name'.
 */
char *
fullObjName(int deallocate, char *name) {
        int     l;
        char    *full;
        char    *myself = "fullObjName";

        if (name == 0)
                return (sdup(myself, T, proxyInfo.default_nis_domain));

        l = strlen(name);
        if (name[l-1] == '.') {
                full = sdup(myself, T, name);
        } else {
                full = scat(myself, T, scat(myself, F, name, "."),
                        sdup(myself, T, proxyInfo.default_nis_domain));
        }
```



```
        if (deallocate)
                free(name);

        return (full);
}
```

*unchanged_portion_omitted_*

```
/*
 * Despite its general-sounding name, this function only knows how to
 * turn a list of attributes ("a,b,c") into an AND filter ("(&(a)(b)(c))").
 */
char *
makeFilter(char *attr) {
        int     len, s, e, c;
        char    *str, *filter, *tmp;
        char    *myself = "makeFilter";

        if (attr == 0 || (len = strlen(attr)) == 0)
                return (0);

        /* Assume already of appropriate form if first char is '(' */
        if (len > 1 && attr[0] == '(' && attr[len-1] == ')')
                return (sdup(myself, T, attr));

        str = sdup(myself, T, attr);
        if (str == 0)
                return (0);
        filter = sdup(myself, T, "(&");
        if (filter == 0) {
                free(str);
                return (0);
        }
        for (s = c = 0; s < len; s = e+1) {
                /* Skip blank space, if any */
                for (; str[s] == ' ' || str[s] == '\t'; s++);
                /* Find delimiter (comma) or end of string */
                for (e = s; str[e] != '\0' && str[e] != ','; e++);
                str[e] = '\0';
                tmp = scat(myself, T, sdup(myself, T, "("),
                        scat(myself, F, &str[s], ")"));
                if (tmp == 0) {
                        sfree(filter);
                        return (0);
                }
                c++;
                filter = scat(myself, T, filter, tmp);
        }

        /*
         * If there's just one component, we return it as is. This
         * means we avoid turning "objectClass=posixAccount" into
         * "(&(objectClass=posixAccount))".
         */
        if (c == 1) {
                sfree(filter);
                return (str);
        }

        /* Add the closing ')' */
        tmp = filter;
        filter = scat(myself, F, tmp, ")");
        sfree(tmp);

        free(str);
```

```
248         return (filter);
249 }

251 /*
252  * Split an AND-filter string into components.
253  */
254 char **
255 makeFilterComp(char *filter, int *numComps) {
256         int     nc = 0, s, e, i;
257         char    **comp = 0, **new, *str;
258         int     len;
259         char    *myself = "makeFilterComp";

261         if ((len = slen(filter)) <= 0)
262                 return (0);

264         /* Is it just a plain "attr=val" string ? If so, return a copy */
265         if (len <= 2 || filter[0] != '(') {
266                 comp = am(myself, 2 * sizeof (comp[0]));
267                 if (comp == 0)
268                         return (0);
269                 comp[0] = sdup(myself, T, filter);
270                 if (comp[0] == 0) {
271                         sfree(comp);
272                         return (0);
273                 }
274                 if (numComps != 0)
275                         *numComps = 1;
276                 return (comp);
277         }

279         if (filter != 0 && (len = strlen(filter)) != 0 && len > 2 &&
280                         filter[0] == '(' && filter[1] == '&' &&
281                         filter[len-1] == ')') {
282                 str = sdup(myself, T, filter);
283                 if (str == 0)
284                         return (0);
285                 for (s = 2; s < len; s = e+1) {
286                         /* Skip past the '(' */
287                         for (; s < len && str[s] != '('; s++);
288                         for (0; s < len && str[s] != '('; s++);
288                         s++;
289                         if (s >= len)
290                                 break;
291                         for (e = s; str[e] != '\0' && str[e] != ')'; e++);
292                         str[e] = '\0';
293                         new = realloc(comp, (nc+1) * sizeof (comp[nc]));
294                         if (new == 0) {
295                                 if (comp != 0) {
296                                         for (i = 0; i < nc; i++)
297                                                 sfree(comp[i]);
298                                         free(comp);
299                                         comp = 0;
300                                 }
301                                 nc = 0;
302                                 break;
303                         }
304                         comp = new;
305                         comp[nc] = sdup(myself, T, &str[s]);
306                         if (comp[nc] == 0) {
307                                 for (i = 0; i < nc; i++)
308                                         sfree(comp[i]);
309                                 sfree(comp);
310                                 comp = 0;
311                                 nc = 0;
312                                 break;
```



```
313                         }
314                         nc++;
315                 }
316                 sfree(str);
317         }

319         if (numComps != 0)
320                 *numComps = nc;

322         return (comp);
323 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   42061 Fri Jul 24 12:28:11 2015
new/usr/src/lib/libnisdb/ldap_map.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2015 Gary Mills
  23  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */

  27 #include <strings.h>
  28 #include <sys/types.h>
  29 #include <sys/stat.h>
  30 #include <errno.h>
  31 #include <stdio.h>
  32 #include <rpcsvc/nis.h>
  33 #include <rpc/xdr.h>

  35 #include "ldap_util.h"
  36 #include "ldap_attr.h"
  37 #include "ldap_ruleval.h"
  38 #include "ldap_op.h"
  39 #include "ldap_map.h"
  40 #include "ldap_glob.h"
  41 #include "ldap_xdr.h"
  42 #include "ldap_val.h"

  44 /* From yptol/dit_access_utils.h */
  45 #define N2LKEY          "rf_key"
  46 #define N2LIPKEY        "rf_ipkey"

  48 __nis_hash_table_mt     ldapMappingList = NIS_HASH_TABLE_MT_INIT;
  49 extern  int yp2ldap;


  52 int
  53 setColumnNames(__nis_table_mapping_t *t) {
  54         int     i, j, nic, noc;
  55         char    **col;
  56         zotypes type;
  57         char    *myself = "setColumnNames";

  59         if (t == 0)
  60                 return (0);
```



```
  62         type = t->objType;
  63         col = t->column;
  64         nic = (col != 0) ? t->numColumns : -1;

  66         t->objType = NIS_BOGUS_OBJ;
  67         t->obj = 0;

  69         /*
  70          * If it's a table object, but there are no translation rules,
  71          * this mapping is for the table object itself. In that case,
  72          * we throw away the column names (if any).
  73          */
  74         if (t->objType == NIS_TABLE_OBJ && t->numRulesFromLDAP == 0 &&
  75                         t->numRulesToLDAP == 0) {
  76                 for (i = 0; i < t->numColumns; i++)
  77                         sfree(t->column[i]);
  78                 sfree(t->column);
  79                 t->column = 0;
  80                 t->numColumns = 0;
  81                 noc = 0;
  82         }

  84         /*
  85          * Verify that all column names found by the parser
  86          * are present in the actual column list.
  87          */
  88         if (verbose) {
  89                 for (i = 0, noc = 0; i < nic; i++) {
  90                         int     found = 0;

  92                         if (col[i] == 0)
  93                                 continue;
  94                         /* Skip the 'zo_*' special column names */
  95                         if (isObjAttrString(col[i]))
  96                                 continue;
  97                         for (j = 0; j < t->numColumns; j++) {
  98                                 if (strcmp(col[i], t->column[j]) == 0) {
  99                                         noc++;
 100                                         found = 1;
 101                                         break;
 102                                 }
 103                         }
 104                         if (!found) {
 105                                 logmsg(MSG_NOTIMECHECK, LOG_WARNING,
 106                                         "%s: No column \"%s\" in \"%s\"",
 107                                         myself, NIL(col[i]), NIL(t->objName));
 108                         }
 109                 }
 110         }

 112         /* Remove any setup by the parser */
 113         for (i = 0; i < nic; i++) {
 114                 sfree(col[i]);
 115         }
 116         sfree(col);

 118         return (0);
 119 }
```

_____***unchanged_portion_omitted_***

```
1226 /*
1227  * Return all table mappings that match the column values in 'q'.
1228  * If there's no match, return those alternative mappings that don't
1229  * have an index; if no such mapping exists, return NULL.
1230  *
1231  * If 'wantWrite' is set, we want mappings for writing (i.e., data
```

```
1232  * to LDAP); otherwise, we want mappings for reading.
1233  *
1234  * If 'wantObj' is set, we want object mappings only (i.e., _not_
1235  * those used to map entries in tables).
1236  *
1237  * If 'dbId' is non-NULL, we select mappings with a matching dbId field.
1238  */
1239 __nis_table_mapping_t **
1240 selectTableMapping(__nis_table_mapping_t *t, db_query *q,
1241                         int wantWrite, int wantObj, char *dbId,
1242                         int *numMatches) {
1243         __nis_table_mapping_t   *r, *x, **tp;
1244         int                     i, nm, numap;
1243         int                     i, j, k, nm, numap;
1245         char                    *myself = "selectTableMapping";

1247         if (numMatches == 0)
1248                 numMatches = &nm;

1250         /*
1251          * Count the number of possible mappings, so that we can
1252          * allocate the 'tp' array up front.
1253          */
1254         for (numap = 0, x = t; x != 0; numap++, x = x->next);

1256         if (numap == 0) {
1257                 *numMatches = 0;
1258                 return (0);
1259         }

1261         tp = am(myself, numap * sizeof (tp[0]));
1262         if (tp == 0) {
1263                 *numMatches = -1;
1264                 return (0);
1265         }

1267         /*
1268          * Special cases:
1269          *
1270          *      q == 0 trivially matches any 't' of the correct object type
1271          *
1272          *      wantObj != 0 means we ignore 'q'
1273          */
1274         if (q == 0 || wantObj) {
1275                 for (i = 0, x = t, nm = 0; i < numap; i++, x = x->next) {
1276                         if (x->objectDN == 0)
1277                                 continue;
1278                         if (wantWrite) {
1279                                 if (x->objectDN->write.scope ==
1280                                                 LDAP_SCOPE_UNKNOWN)
1281                                         continue;
1282                         } else {
1283                                 if (x->objectDN->read.scope ==
1284                                                 LDAP_SCOPE_UNKNOWN)
1285                                         continue;
1286                         }
1287                         if (wantObj) {
1288                                 if (x->numColumns > 0)
1289                                         continue;
1290                         } else {
1291                                 if (x->numColumns <= 0)
1292                                         continue;
1293                         }
1294                         if (dbId != 0 && x->dbId != 0 &&
1295                                         strcmp(dbId, x->dbId) != 0)
1296                                 continue;
```



```
1297                         tp[nm] = x;
1298                         nm++;
1299                 }
1300                 *numMatches = nm;
1301                 if (nm == 0) {
1302                         sfree(tp);
1303                         tp = 0;
1304                 }
1305                 return (tp);
1306         }

1308         /* Scan all mappings, and collect candidates */
1309         for (nm = 0, r = 0, x = t; x != 0; x = x->next) {
1310                 if (x->objectDN == 0)
1311                         continue;
1312                 if (wantWrite) {
1313                         if (x->objectDN->write.scope == LDAP_SCOPE_UNKNOWN)
1314                                 continue;
1315                 } else {
1316                         if (x->objectDN->read.scope == LDAP_SCOPE_UNKNOWN)
1317                                 continue;
1318                 }
1319                 /* Only want table/entry mappings */
1320                 if (x->numColumns <= 0)
1321                         continue;
1322                 if (dbId != 0 && x->dbId != 0 &&
1323                                 strcmp(dbId, x->dbId) != 0)
1324                         continue;
1325                 /*
1326                  * It's a match if: there are no indexes, or we actually
1327                  * match the query with the indexes.
1328                  */
1329                 if (x->index.numIndexes <= 0 ||
1330                                         verifyIndexMatch(x, q, 0, 0, 0)) {
1331                         tp[nm] = x;
1332                         nm++;
1333                 }
1334         }

1336         if (nm == 0) {
1337                 free(tp);
1338                 tp = 0;
1339         }

1341         *numMatches = nm;

1343         return (tp);
1344 }
______unchanged_portion_omitted_

1413 extern bool_t   xdr_nis_object(register XDR *xdrs, nis_object *objp);

1415 /*
1416  * Copy an XDR:ed version of the NIS+ object 'o' (or the one indicated
1417  * by 't->objName' if 'o' is NULL) to the place indicated by
1418  * 't->objectDN->write'. Return an appropriate LDAP status code.
1419  */
1420 int
1421 objToLDAP(__nis_table_mapping_t *t, nis_object *o, entry_obj **ea, int numEa) {
1422         __nis_table_mapping_t   **tp;
1422         XDR                     xdr;
1423         char                    *objName;
1423         int                     stat, osize, n, numMatches = 0;
1424         void                    *buf;
1425         __nis_rule_value_t      *rv;
1426         __nis_value_t           *val;
```

```
1427         __nis_single_value_t    *sv;
1428         char                    **attrName, *dn;
1429         char                    *myself = "objToLDAP";

1431         if (t == 0)
1432                 return (LDAP_PARAM_ERROR);

1434         logmsg(MSG_NOTIMECHECK,
1435 #ifdef  NISDB_LDAP_DEBUG
1436                 LOG_WARNING,
1437 #else
1438                 LOG_INFO,
1439 #endif  /* NISDB_LDAP_DEBUG */
1440                 "%s: %s", myself, NIL(t->objName));

1442         tp = selectTableMapping(t, 0, 1, 1, 0, &numMatches);
1443         if (tp == 0 || numMatches <= 0) {
1444                 sfree(tp);
1445                 logmsg(MSG_NOTIMECHECK,
1446 #ifdef  NISDB_LDAP_DEBUG
1447                         LOG_WARNING,
1448 #else
1449                         LOG_INFO,
1450 #endif  /* NISDB_LDAP_DEBUG */
1451                         "%s: %s (no mapping)", myself, NIL(t->objName));
1452                 return (LDAP_SUCCESS);
1453         }

1455         for (n = 0; n < numMatches; n++) {

1457                 t = tp[n];

1459                 if (o == 0) {
1460                         sfree(tp);
1461                         return (LDAP_OPERATIONS_ERROR);
1462                 }

1464                 buf = (char *)xdrNisObject(o, ea, numEa, &osize);
1465                 if (buf == 0) {
1466                         sfree(tp);
1467                         return (LDAP_OPERATIONS_ERROR);
1468                 }

1470                 /*
1471                  * Prepare to build a rule-value containing the XDR:ed
1472                  * object
1473                  */
1474                 rv = am(myself, sizeof (*rv));
1475                 sv = am(myself, sizeof (*sv));
1476                 val = am(myself, sizeof (*val));
1477                 attrName = am(myself, sizeof (attrName[0]));
1478                 if (attrName != 0)
1479                         attrName[0] = attrVal(myself, "nisplusObject",
1480                                               "nisplusObject",
1481                                               t->objectDN->write.attrs);
1482                 if (rv == 0 || sv == 0 || val == 0 || attrName == 0 ||
1483                                 attrName[0] == 0) {
1484                         sfree(tp);
1485                         sfree(buf);
1486                         sfree(rv);
1487                         sfree(sv);
1488                         sfree(val);
1489                         sfree(attrName);
1490                         return (LDAP_NO_MEMORY);
1491                 }
```



```
1493                 sv->length = osize;
1494                 sv->value = buf;

1496                 /* 'vt_ber' just means "not a NUL-terminated string" */
1497                 val->type = vt_ber;
1498                 val->repeat = 0;
1499                 val->numVals = 1;
1500                 val->val = sv;

1502                 rv->numAttrs = 1;
1503                 rv->attrName = attrName;
1504                 rv->attrVal = val;

1506                 /*
1507                  * The 'write.base' is the actual DN of the entry (and the
1508                  * scope had better be 'base', but we don't check that).
1509                  */
1510                 dn = t->objectDN->write.base;

1512                 stat = ldapModify(dn, rv, t->objectDN->write.attrs, 1);

1514                 freeRuleValue(rv, 1);

1516         logmsg(MSG_NOTIMECHECK,
1517 #ifdef  NISDB_LDAP_DEBUG
1518                 LOG_WARNING,
1519 #else
1520                 LOG_INFO,
1521 #endif  /* NISDB_LDAP_DEBUG */
1522                 "%s: %s (%s)", myself, NIL(t->objName), ldap_err2string(stat));

1524                 if (stat != LDAP_SUCCESS)
1525                         break;

1527         }

1529         sfree(tp);

1531         return (stat);
1532 }

1534 /*
1535  * Retrieve a copy of the 't->objName' object from LDAP, where it's
1536  * stored in XDR:ed form in the place indicated by 't->objectDN->read'.
1537  * Un-XDR the object, and return a pointer to it in '*obj'; it's the
1538  * responsibility of the caller to free the object when it's no
1539  * longer needed.
1540  *
1541  * Returns an appropriate LDAP status.
1542  */
1543 int
1544 objFromLDAP(__nis_table_mapping_t *t, nis_object **obj,
1545                 entry_obj ***eaP, int *numEaP) {
1546         __nis_table_mapping_t   **tp;
1548         XDR                     xdr;
1547         nis_object              *o;
1548         __nis_rule_value_t      *rv;
1549         __nis_ldap_search_t     *ls;
1550         char                    *attrs[2], *filter, **fc = 0;
1551         void                    *buf;
1552         int                     i, j, nfc, nrv, blen, stat = LDAP_SUCCESS;
1553         int                     n, numMatches;
1554         char                    *myself = "objFromLDAP";

1556         if (t == 0)
1557                 return (LDAP_PARAM_ERROR);
```

```
1559         /*
1560          * If there's nowhere to store the result, we might as
1561          * well pretend all went well, and return right away.
1562          */
1563         if (obj == 0)
1564                 return (LDAP_SUCCESS);

1566         /* Prepare for the worst */
1567         *obj = 0;

1569         logmsg(MSG_NOTIMECHECK,
1570 #ifdef  NISDB_LDAP_DEBUG
1571                 LOG_WARNING,
1572 #else
1573                 LOG_INFO,
1574 #endif  /* NISDB_LDAP_DEBUG */
1575                 "%s: %s", myself, NIL(t->objName));

1577         tp = selectTableMapping(t, 0, 0, 1, 0, &numMatches);
1578         if (tp == 0 || numMatches <= 0) {
1579                 sfree(tp);
1580                 logmsg(MSG_NOTIMECHECK,
1581 #ifdef  NISDB_LDAP_DEBUG
1582                         LOG_WARNING,
1583 #else
1584                         LOG_INFO,
1585 #endif  /* NISDB_LDAP_DEBUG */
1586                         "%s: %s (no mapping)", myself, NIL(t->objName));
1587                 return (LDAP_SUCCESS);
1588         }

1590         for (n = 0; n < numMatches; n++) {

1592                 t = tp[n];

1594                 filter = makeFilter(t->objectDN->read.attrs);
1595                 if (filter == 0 || (fc = makeFilterComp(filter, &nfc)) == 0 ||
1596                                 nfc <= 0) {
1597                         sfree(tp);
1598                         sfree(filter);
1599                         freeFilterComp(fc, nfc);
1600                         return ((t->objectDN->read.attrs != 0) ?
1601                                 LDAP_NO_MEMORY : LDAP_PARAM_ERROR);
1602                 }
1603                 /* Don't need the filter, just the components */
1604                 sfree(filter);

1606                 /*
1607                  * Look for a "nisplusObject" attribute, and (if found) copy
1608                  * the value to attrs[0]. Also remove the "nisplusObject"
1609                  * attribute and value from the filter components.
1610                  */
1611                 attrs[0] = sdup(myself, T, "nisplusObject");
1612                 if (attrs[0] == 0) {
1613                         sfree(tp);
1614                         freeFilterComp(fc, nfc);
1615                         return (LDAP_NO_MEMORY);
1616                 }
1617                 attrs[1] = 0;
1618                 for (i = 0; i < nfc; i++) {
1619                         char    *name, *value;
1620                         int     compare;

1622                         name = fc[i];
1623                         /* Skip if not of attr=value form */
```



```
1624                         if ((value = strchr(name, '=')) == 0)
1625                                 continue;

1627                         /* Temporarily overWrite the '=' with a '\0' */
1628                         *value = '\0';

1630                         /* Compare with our target attribute name */
1631                         compare = strcasecmp("nisplusObject", name);

1633                         /* Put back the '=' */
1634                         *value = '=';

1636                         /* Is it the name we're looking for ? */
1637                         if (compare == 0) {
1638                                 sfree(attrs[0]);
1639                                 attrs[0] = sdup(myself, T, value+1);
1640                                 if (attrs[0] == 0) {
1641                                         sfree(tp);
1642                                         freeFilterComp(fc, nfc);
1643                                         return (LDAP_NO_MEMORY);
1644                                 }
1645                                 sfree(fc[i]);
1646                                 if (i < nfc-1)
1647                                         (void) memmove(&fc[i], &fc[i+1],
1648                                                 (nfc-1-i) * sizeof (fc[i]));
1649                                 nfc--;
1650                                 break;
1651                         }
1652                 }

1654                 ls = buildLdapSearch(t->objectDN->read.base,
1655                                         t->objectDN->read.scope,
1656                                         nfc, fc, 0, attrs, 0, 1);
1657                 sfree(attrs[0]);
1658                 freeFilterComp(fc, nfc);
1659                 if (ls == 0) {
1660                         sfree(tp);
1661                         return (LDAP_OPERATIONS_ERROR);
1662                 }

1664                 nrv = 0;
1665                 rv = ldapSearch(ls, &nrv, 0, &stat);
1666                 if (rv == 0) {
1667                         sfree(tp);
1668                         freeLdapSearch(ls);
1669                         return (stat);
1670                 }

1672                 for (i = 0, buf = 0; i < nrv && buf == 0; i++) {
1673                         for (j = 0; j < rv[i].numAttrs; j++) {
1674                                 if (strcasecmp(ls->attrs[0],
1675                                         rv[i].attrName[j]) == 0) {
1676                                         if (rv[i].attrVal[j].numVals <= 0)
1677                                                 continue;
1678                                         buf = rv[i].attrVal[j].val[0].value;
1679                                         blen = rv[i].attrVal[j].val[0].length;
1680                                         break;
1681                                 }
1682                         }
1683                 }

1685                 if (buf != 0) {
1686                         o = unXdrNisObject(buf, blen, eaP, numEaP);
1687                         if (o == 0) {
1688                                 sfree(tp);
1689                                 freeLdapSearch(ls);
```

```
1690                                    freeRuleValue(rv, nrv);
1691                                    return (LDAP_OPERATIONS_ERROR);
1692                            }
1693                            stat = LDAP_SUCCESS;
1694                            *obj = o;
1695                    } else {
1696                            stat = LDAP_NO_SUCH_OBJECT;
1697                    }

1699                    freeLdapSearch(ls);
1700                    freeRuleValue(rv, nrv);

1702            logmsg(MSG_NOTIMECHECK,
1703 #ifdef  NISDB_LDAP_DEBUG
1704                    LOG_WARNING,
1705 #else
1706                    LOG_INFO,
1707 #endif  /* NISDB_LDAP_DEBUG */
1708                    "%s: %s (%s)", myself, NIL(t->objName), ldap_err2string(stat));

1710                    if (stat != LDAP_SUCCESS)
1711                            break;

1713            }

1715            sfree(tp);

1717            return (stat);
1718 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   29273 Fri Jul 24 12:28:11 2015
new/usr/src/lib/libnisdb/ldap_nisdbquery.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


#include <strings.h>
#include <string.h>
#include <lber.h>
#include <ldap.h>

#include "db_item_c.h"

#include "nisdb_mt.h"

#include "ldap_util.h"
#include "ldap_structs.h"
#include "ldap_val.h"
#include "ldap_ruleval.h"
#include "ldap_op.h"
#include "ldap_nisdbquery.h"
#include "ldap_attr.h"
#include "ldap_xdr.h"
#include "ldap_ldap.h"


item *
buildItem(int len, void *value) {
        char    *myself = "buildItem";
        item    *i = am(myself, sizeof (*i));
        int     mlen = len;

        if (i == 0)
                return (0);

        /*
         * To this function, a NULL value, or a length less than or equal
         * zero means an item with no value. Hence, buildItem(0, 0) is
         * _not_ the right way to create index_value == 0 to indicate
         * deletion.
         */
```



```
         */
        if (value == 0 || len <= 0) {
                i->itemvalue.itemvalue_len = 0;
                i->itemvalue.itemvalue_val = 0;
                return (i);
        }

        /*
         * NIS+ usually stores the terminating NUL for strings, so we add
         * it here just in case. This means we usually waste a byte for
         * binary column values...
         */
        if (len > 0 && ((char *)value)[len-1] != '\0')
                mlen++;

        i->itemvalue.itemvalue_len = len;
        i->itemvalue.itemvalue_val = am(myself, mlen);
        if (mlen > 0 && i->itemvalue.itemvalue_val == 0) {
                free(i);
                return (0);
        }
        memcpy(i->itemvalue.itemvalue_val, value, len);

        return (i);
}
_____unchanged_portion_omitted_

/*
 * Given an array index[0..num-1] of pointers to strings of the form
 * "name=value", create the corresponding db_queries. "name=" indicates
 * deletion, which results in a db_query component where index_value == 0.
 *
 * The __nis_table_mapping_t structure is used to translate column
 * names to indices.
 *
 * If 'rvP' is non-NULL, the searchable columns from the 'index'
 * name/value pairs are used to retrieve copies of the corresponding NIS+
 * entries, and '*rvP' is initialized with the current entry values
 * and object attributes. Names/values supplied in 'index' override
 * those from existing NIS+ entries.
 */
db_query **
createQuery(int num, char **index, __nis_table_mapping_t *t,
                __nis_rule_value_t **rvP, int *numVals) {
        db_query                **q;
        db_qcomp                *qc;
        int                     i, j, n, a, nv, niv;
        int                     i, j, n, a, nv, niv, stat, sinum;
        __nis_rule_value_t      *rvq;
        __nis_buffer_t          b = {0, 0};
        char                    *table = 0;
        char                    *myself = "createQuery";

        rvq = initRuleValue(1, 0);
        if (rvq == 0)
                return (0);

        if (numVals == 0)
                numVals = &nv;
        *numVals = 0;

        if (rvP != 0) {
                /*
                 * Try to obtain a copy of the table object, in order to
                 * determine the searchable columns. A failure isn't
                 * necessarily fatal; we just try to compose the entire
```

```
248                  * LDAP data from the col=val pairs.
249                  */
250                 table = fullObjName(F, t->objName);
251                 if (table == 0) {
252                         logmsg(MSG_NOTIMECHECK, LOG_ERR,
253                                 "%s: Error converting \"%s\" to FQ object name",
254                                 myself, NIL(t->objName));
255                         freeRuleValue(rvq, 1);
256                         return (0);
257                 }
258         }

260         /* Create a rule-value from the col=val pairs */
261         for (n = 0; n < num; n++) {
260                 char    *name;
262                 char    *value;

264                 if ((value = strchr(index[n], '=')) == 0) {
265                         logmsg(MSG_NOTIMECHECK, LOG_WARNING,
266                                 "%s: no '=' in \"%s\"",
267                                 myself, index[n]);
268                         continue;
269                 }

271                 *value = '\0';
272                 value++;

274                 for (a = 0; a < t->numColumns; a++) {
275                         if (strcmp(index[n], t->column[a]) == 0) {
275                                 int             i, len = slen(value)+1;

277                                 /* Add col=val pair to 'rvq' */
278                                 if (addSCol2RuleValue(index[n], value, rvq)) {
279                                         freeRuleValue(rvq, 1);
280                                         sfree(table);
281                                         return (0);
282                                 }

284                                 break;
285                         }
286                 }
287                 if (a >= t->numColumns) {
288                         logmsg(MSG_NOTIMECHECK, LOG_WARNING,
289                                 "%s: Ignoring unknown column \"%s\"",
290                                 myself, NIL(index[n]));
291                 }
292         }

294         /*
295          * Find out if any of the columns specified via the 'index'
296          * array are multi-valued.
297          */
298         for (n = 0, niv = 1; n < rvq->numColumns; n++) {
299                 if (rvq->colVal[n].numVals > 1)
300                         niv *= rvq->colVal[n].numVals;
301         }

303         *numVals = 1;

305         sfree(b.buf);
306         sfree(table);

308         if (rvq->numColumns <= 0) {
309                 freeRuleValue(rvq, *numVals);
310                 *numVals = 0;
311                 return (0);
```



```
312         }

314         /*
315          * If any column name was repeated in the col=val pairs (but with
316          * different values), 'rvq' will have one or more multi-valued
317          * column values. We now convert those into an array of rule-values
318          * where every column is single-valued.
319          *
320          * Since we want all combinations of column values, the number
321          * of array elements is the product of all column value counts.
322          *
323          * There are four possible combinations of 'index' and NIS+ data:
324          *
325          * (1)  Only single-valued 'index' columns, and at most one NIS+
326          *      entry, so 'rvq' is complete, and '*numVals' == 1.
327          *
328          * (2)  Single-valued 'index' columns, but multiple NIS+ entries.
329          *      '*numVals' reflects the number of NIS+ entries, and no
330          *      expansion of 'index' column values to array elements is
331          *      needed.
332          *
333          * (3)  At least one multi-valued 'index', and multiple NIS+
334          *      entries. We already rejected the NIS+ data for this case
335          *      above, so it is in fact equivalent to case (4).
336          *
337          * (4)  At least one multi-valued 'index', but at most one NIS+
338          *      entry. This is the case where we must expand the multi-valued
339          *      columns to multiple array elements.
340          */
341         if (niv > 1 && *numVals == 1) {
342                 __nis_rule_value_t      *rv;
343                 int                     repeat;

345                 /*
346                  * By using initRuleValue() to create 'rv', and make each
347                  * element a clone of 'rvq', we save a lot of code. The
348                  * down side is that 'rv' only really needs one element
349                  * for each rv[].colVal[].val array, but we know that at
350                  * least one rvq->colVal[].val array has more than one
351                  * element. Hence, making 'rv' a clone of 'rvq' will waste
352                  * memory.
353                  *
354                  * However, we believe this waste is acceptable, because
355                  * we expect that 'niv' will be small. Also, we are executing
356                  * in the context of a utility command, not in a daemon.
357                  */
358                 rv = initRuleValue(niv, rvq);
359                 if (rv == 0) {
360                         freeRuleValue(rvq, 1);
361                         *numVals = 0;
362                         return (0);
363                 }

365                 /*
366                  * For each column value in 'rvq', copy to the appropriate
367                  * place in 'rv', so that the end result is that all
368                  * combinations of values are enumerated, and each
369                  * 'rv[n].colVal[i]' is single-valued.
370                  *
371                  * We do this by traversing the rv[] array 'rvq->numColumns'
372                  * times, where each traversal 'i' works on the values
373                  * for rvq->colVal[i]. A repeat factor 'repeat' starts out
374                  * at '1', and is multiplied by 'rvq->colVal[i].numVals'
375                  * at the end of each traversal. Every value
376                  * rvq->colVal[i].val[j] is repeated 'repeat' times.
377                  *
```

```
 378                  * This algorithm works by regarding the rv[] array as
 379                  * an I-dimensional array (I = rvq->numColumns), where
 380                  * each dimension 'i' corresponds to the values for
 381                  * rvq->colVal[i]. The I-dimensional array is stored
 382                  * in column-major order.
 383                  *
 384                  * Since the 'rv' elements start out as copies of 'rvq',
 385                  * we achieve the "copy" of the 'rvq' column values by
 386                  * deleting those we don't want from the 'rv' elements.
 387                  */
 388                 for (i = 0, repeat = 1; i < rvq->numColumns; i++) {
 389                         int     r, k;
 390                         for (n = 0, j = 0, r = 0; n < niv; n++) {
 391                                 /*
 392                                  * Free all but element 'j' of the
 393                                  * rv[n].colVal[i].val array.
 394                                  */
 395                                 for (k = 0; k < rv[n].colVal[i].numVals; k++) {
 396                                         /* Leave element 'j' in place */
 397                                         if (k == j)
 398                                                 continue;
 399                                         sfree(rv[n].colVal[i].val[k].
 400                                                 value);
 401                                 }
 402                                 rv[n].colVal[i].numVals = 1;
 403                                 /* Move element 'j' to zero */
 404                                 if (j != 0)
 405                                         rv[n].colVal[i].val[0] =
 406                                                 rv[n].colVal[i].val[j];

 408                                 /*
 409                                  * Increment the repeat index 'r'. If >=
 410                                  * 'repeat', reset 'r' and increment the
 411                                  * value index 'j'. If 'j' >=
 412                                  * rvq->colVal[i].numVals, start over on
 413                                  * the column values for column 'i' (i.e.,
 414                                  * reset 'j' to zero).
 415                                  */
 416                                 r += 1;
 417                                 if (r >= repeat) {
 418                                         r = 0;
 419                                         j += 1;
 420                                         if (j >= rvq->colVal[i].numVals)
 421                                                 j = 0;
 422                                 }
 423                         }
 424                         repeat *= rvq->colVal[i].numVals;
 425                 }

 427                 *numVals = niv;
 428                 freeRuleValue(rvq, 1);
 429                 rvq = rv;
 430                 rv = 0;
 431         }

 433         q = am(myself, *numVals * sizeof (q[0]));
 434         if (q == 0) {
 435                 freeRuleValue(rvq, *numVals);
 436                 return (0);
 437         }

 439         /*
 440          * Create queries from the rvq[] array.
 441          */
 442         for (a = 0; a < *numVals; a++) {
 443                 int     nn, err = 0;
```



```
 445                 qc = am(myself, rvq[a].numColumns * sizeof (*qc));
 446                 if (qc != 0) {
 447                         for (nn = 0, i = 0; i < rvq[a].numColumns; i++) {
 448                                 for (j = 0; j < t->numColumns; j++) {
 449                                         if (strcmp(rvq[a].colName[i],
 450                                                         t->column[j]) == 0) {
 451                                                 break;
 452                                         }
 453                                 }
 454                                 if (j >= t->numColumns)
 455                                         continue;
 456                                 qc[nn].which_index = j;
 457                                 if (rvq[a].colVal[i].numVals > 0) {
 458                                         qc[nn].index_value = buildItem(
 459                                                 rvq[a].colVal[i].val[0].length,
 460                                                 rvq[a].colVal[i].val[0].value);
 461                                         if (qc[nn].index_value == 0)
 462                                                 err++;
 463                                 } else {
 464                                         logmsg(MSG_NOTIMECHECK, LOG_ERR,
 465                                                 "%s: No values for [%d]%s",
 466                                                 myself, a, rvq[a].colName[i]);
 467                                         err++;
 468                                 }
 469                                 nn++;
 470                         }
 471                         if (err == 0)
 472                                 q[a] = buildQuery(nn, qc);
 473                 }
 474                 if (err > 0 || q[a] == 0) {
 475                         freeQueries(q, a);
 476                         for (a = 0; a < nn; a++)
 477                                 freeQcomp(&qc[a], F);
 478                         sfree(qc);
 479                         freeRuleValue(rvq, *numVals);
 480                         return (0);
 481                 }
 482         }

 484         if (rvP != 0) {
 485                 *rvP = rvq;
 486         } else {
 487                 freeRuleValue(rvq, 1);
 488                 *numVals = 0;
 489         }

 491         return (q);
 492 }
_____unchanged_portion_omitted_

 686 db_query **
 687 createNisPlusEntry(__nis_table_mapping_t *t, __nis_rule_value_t *rv,
 688                         db_query *qin, __nis_obj_attr_t ***objAttr,
 689                         int *numQueries) {
 690         db_query                **query = 0;
 691         int                     r, i, j, ir;
 692         __nis_value_t           *rval, *lval;
 693         __nis_mapping_item_t    *litem;
 694         int                     numItems;
 695         int                     nq;
 695         int                     nq, iqc;
 696         __nis_obj_attr_t        **attr = 0;
 697         char                    **dn = 0;
 698         int                     numDN = 0;
 699         char                    *myself = "createNisPlusEntry";
```

```
701         if (t == 0 || t->objectDN == 0 || rv == 0)
702                 return (0);
703
704         /* Establish default, per-thread, search base */
705         __nisdb_get_tsd()->searchBase = t->objectDN->read.base;
706
707         for (r = 0, nq = 0; r < t->numRulesFromLDAP; r++) {
708                 int                     nrq, ntq, err;
709                 db_query                **newq;
710                 __nis_obj_attr_t        **newattr;
711
712                 rval = buildRvalue(&t->ruleFromLDAP[r]->rhs,
713                         mit_ldap, rv, NULL);
714                 if (rval == 0)
715                         continue;
716
717                 litem = buildLvalue(&t->ruleFromLDAP[r]->lhs, &rval,
718                                 &numItems);
719                 if (litem == 0) {
720                         freeValue(rval, 1);
721                         /* XXX Should this be a fatal error ? */
722                         continue;
723                 }
724
725                 lval = 0;
726                 for (i = 0; i < numItems; i++) {
727                         __nis_value_t   *tmpval, *old;
728
729                         tmpval = getMappingItem(&litem[i],
730                                 mit_nisplus, 0, 0, NULL);
731
732                         /*
733                          * If the LHS specifies an out-of-context LDAP or
734                          * NIS+ item, we do the update right here. We
735                          * don't add any values to 'lval'; instead, we
736                          * skip to the next item. (However, we still
737                          * get a string representation of the LHS in case
738                          * we need to report an error.)
739                          */
740                         if (litem[i].type == mit_ldap) {
741                                 int     stat;
742
743                                 if (dn == 0)
744                                         dn = findDNs(myself, rv, 1,
745                                                 t->objectDN->write.base,
746                                                 &numDN);
747
748                                 stat = storeLDAP(&litem[i], i, numItems, rval,
749                                         t->objectDN, dn, numDN);
750                                 if (stat != LDAP_SUCCESS) {
751                                         char    *iname = "<unknown>";
752
753                                         if (tmpval != 0 &&
754                                                         tmpval->numVals == 1)
755                                                 iname = tmpval->val[0].value;
756                                         logmsg(MSG_NOTIMECHECK, LOG_ERR,
757                                                 "%s: LDAP store \"%s\": %s",
758                                                 myself, iname,
759                                                 ldap_err2string(stat));
760                                 }
761
762                                 freeValue(tmpval, 1);
763                                 continue;
764                         }
```



```
766                         old = lval;
767                         lval = concatenateValues(old, tmpval);
768                         freeValue(tmpval, 1);
769                         freeValue(old, 1);
770                 }
771
772                 freeMappingItem(litem, numItems);
773                 if (lval == 0 || lval->numVals <= 0 || rval->numVals <= 0) {
774                         freeValue(lval, 1);
775                         freeValue(rval, 1);
776                         continue;
777                 }
778
779                 /*
780                  * We now have a number of possible cases. The notation
781                  * used in the table is:
782                  *
783                  *      single          A single value (numVals == 1)
784                  *      single/rep      A single value with repeat == 1
785                  *      multi[N]        N values
786                  *      multi[N]/rep    M values with repeat == 1
787                  *      (M)             M resulting db_query's
788                  *
789                  * lval \ rval  single  single/rep      multi[N] multi[N]/rep
790                  * single         (1)       (1)          (1)        (1)
791                  * single/rep     (1)       (1)          (N)        (N)
792                  * multi[M]       (1)       (1)          (1)     1+(N-1)/M
793                  * multi[M]/rep   (1)       (1)          (1)     1+(N-1)/M
794                  *
795                  * Of course, we already have 'nq' db_query's from previous
796                  * rules, so the resulting number of queries is max(1,nq)
797                  * times the numbers in the table above.
798                  */
799
800                 /* The number of queries resulting from the current rule */
801                 if (rval->numVals > 1) {
802                         if (lval->numVals == 1 && lval->repeat)
803                                 nrq = rval->numVals;
804                         else if (lval->numVals > 1 && rval->repeat)
805                                 nrq = 1 + ((rval->numVals-1)/lval->numVals);
806                         else
807                                 nrq = 1;
808                 } else {
809                         nrq = 1;
810                 }
811
812                 /* Total number of queries after adding the current rule */
813                 if (nq <= 0)
814                         ntq = nrq;
815                 else
816                         ntq = nq * nrq;
817
818                 if (ntq > nq) {
819                         newq = realloc(query, ntq * sizeof (query[0]));
820                         newattr = realloc(attr, ntq * sizeof (attr[0]));
821                         if (newq == 0 || newattr == 0) {
822                                 logmsg(MSG_NOMEM, LOG_ERR,
823                                         "%s: realloc(%d) => NULL",
824                                         myself, ntq * sizeof (query[0]));
825                                 freeValue(lval, 1);
826                                 freeValue(rval, 1);
827                                 freeQueries(query, nq);
828                                 freeObjAttr(attr, nq);
829                                 sfree(newq);
830                                 freeDNs(dn, numDN);
831                                 return (0);
```

```
832                         }
833                         query = newq;
834                         attr = newattr;
835                 }

837                 /*
838                  * Copy/clone the existing queries to the new array,
839                  * remembering that realloc() has done the first 'nq'
840                  * ones.
841                  *
842                  * If there's an error (probably memory allocation), we
843                  * still go through the rest of the array, so that it's
844                  * simple to free the elements when we clean up.
845                  */
846                 for (i = 1, err = 0; i < nrq; i++) {
847                         for (j = 0; j < nq; j++) {
848                                 query[(nq*i)+j] = cloneQuery(query[j],
849                                                 t->numColumns);
850                                 if (query[(nq*i)+j] == 0 &&
851                                                 query[j] != 0)
852                                         err++;
853                                 attr[(nq*i)+j] = cloneObjAttr(attr[j]);
854                                 if (attr[(nq*i)+j] == 0 &&
855                                                 attr[j] != 0)
856                                         err++;
857                         }
858                 }

860                 if (err > 0) {
861                         freeValue(lval, 1);
862                         freeValue(rval, 1);
863                         freeQueries(query, ntq);
864                         freeObjAttr(attr, ntq);
865                         freeDNs(dn, numDN);
866                         return (0);
867                 }

869                 /*
870                  * Special case if nq == 0 (i.e., the first time we
871                  * allocated db_query's). If so, we now allocate empty
872                  * db_qcomp arrays, which simplifies subsequent
873                  * copying of values.
874                  */
875                 if (nq <= 0) {
876                         (void) memset(query, 0, ntq * sizeof (query[0]));
877                         (void) memset(attr, 0, ntq * sizeof (attr[0]));
878                         for (i = 0, err = 0; i < ntq; i++) {
879                                 query[i] = am(myself, sizeof (*query[i]));
880                                 if (query[i] == 0) {
881                                         err++;
882                                         break;
883                                 }
884                                 query[i]->components.components_val =
885                                                 am(myself, t->numColumns *
886                         sizeof (query[i]->components.components_val[0]));
887                                 if (query[i]->components.components_val == 0) {
888                                         err++;
889                                         break;
890                                 }
891                                 query[i]->components.components_len = 0;
892                         }
893                         if (err > 0) {
894                                 freeValue(lval, 1);
895                                 freeValue(rval, 1);
896                                 freeQueries(query, ntq);
897                                 freeObjAttr(attr, ntq);
```



```
898                                 freeDNs(dn, numDN);
899                                 return (0);
900                         }
901                 }

903                 /* Now we're ready to add the new values */
904                 for (i = 0, ir = 0; i < lval->numVals; i++) {
905                         char    *oaName = 0;
906                         int     index;

908                         /* Find column index */
909                         for (index = 0; index < t->numColumns;
910                                         index++) {
911                                 if (strncmp(t->column[index],
912                                                 lval->val[i].value,
913                                         lval->val[i].length) == 0)
914                                         break;
915                         }
916                         if (index >= t->numColumns) {
917                                 /*
918                                  * Could be one of the special object
919                                  * attributes.
920                                  */
921                                 oaName = isObjAttr(&lval->val[i]);
922                                 if (oaName == 0)
923                                         continue;
924                         }

926                         for (j = i*nrq; j < (i+1)*nrq; j++) {
927                                 int     k;

929                                 /* If we're out of values, repeat last one */
930                                 ir = (j < rval->numVals) ?
931                                         j : rval->numVals - 1;

933                                 /*
934                                  * Step through the query array, adding
935                                  * the new value every 'nrq' queries, and
936                                  * starting at 'query[j % nrq]'.
937                                  */
938                                 for (k = j % nrq, err = 0; k < ntq; k += nrq) {
939                                         int     ic, c;

941                                         if (oaName != 0) {
942                                                 int     fail = setObjAttrField(
943                                                                 oaName,
944                                                                 &rval->val[ir],
945                                                                 &attr[k]);
946                                                 if (fail) {
947                                                         err++;
948                                                         break;
949                                                 }
950                                                 continue;
951                                         }

953                                         ic = query[k]->components.
954                                                 components_len;
955                                         /*
956                                          * If we've already filled this
957                                          * query, the new value is a dup
958                                          * which we'll ignore.
959                                          */
960                                         if (ic >= t->numColumns)
961                                                 continue;

963                                         /*
```

```
964                                      * Do we already have a value for
965                                      * this 'index' ?
966                                      */
967                                     for (c = 0; c < ic; c++) {
968                                             if (query[k]->components.
969                                                     components_val[c].
970                                                     which_index == index)
971                                                     break;
972                                     }

974                                     /* If no previous value, add it */
975                                     if (c >= ic) {
976                                             int     l;
977                                             char    *v;

979                                             query[k]->components.
980                                                     components_val[ic].
981                                                     which_index = index;
982                                             l = rval->val[ir].length;
983                                             v = rval->val[ir].value;
984                                             if (rval->type == vt_string &&
985                                                     l > 0 &&
986                                                     v[l-1] != '\0' &&
987                                                     v[l] == '\0')
988                                                     l++;
989                                             query[k]->components.
990                                                     components_val[ic].
991                                                     index_value =
992                                                     buildItem(l, v);
993                                             if (query[k]->
994                                                     components.
995                                                     components_val[ic].
996                                                     index_value == 0) {
997                                                     err++;
998                                                     break;
999                                             }
1000                                            query[k]->components.
1001                                                    components_len++;
1002                                    }
1003                            }
1004                            if (err > 0) {
1005                                    freeValue(lval, 1);
1006                                    freeValue(rval, 1);
1007                                    freeQueries(query, ntq);
1008                                    freeObjAttr(attr, ntq);
1009                                    freeDNs(dn, numDN);
1010                                    return (0);
1011                            }
1012                    }
1013            }
1014            freeValue(lval, 1);
1015            freeValue(rval, 1);

1017            nq = ntq;
1018    }

1020    freeDNs(dn, numDN);

1022    if (nq <= 0) {
1023            sfree(query);
1024            query = 0;
1025    }

1027    /* Should we filter on index or input query ? */
1028    if (query != 0) {
1029            if (t->index.numIndexes > 0)
```



```
1030                            query = filterQuery(t, query, qin, &attr, &nq);
1031                    else if (qin != 0)
1032                            query = filterQuery(0, query, qin, &attr, &nq);
1033            }

1035            if (query != 0 && numQueries != 0)
1036                    *numQueries = nq;

1038            if (objAttr != 0)
1039                    *objAttr = attr;
1040            else
1041                    freeObjAttr(attr, nq);

1043            return (query);
1044    }
______unchanged_portion_omitted_
```

```
**********************************************************
   66150 Fri Jul 24 12:28:11 2015
new/usr/src/lib/libnisdb/ldap_op.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <synch.h>
#include <strings.h>
#include <sys/time.h>
#include <ctype.h>

#include "ldap_op.h"
#include "ldap_util.h"
#include "ldap_structs.h"
#include "ldap_ruleval.h"
#include "ldap_attr.h"
#include "ldap_print.h"
#include "ldap_glob.h"

#include "nis_parse_ldap_conf.h"

#ifndef LDAPS_PORT
#define LDAPS_PORT      636
#endif

static int setupConList(char *serverList, char *who,
                        char *cred, auth_method_t method);


/*
 * Build one of our internal LDAP search structures, containing copies of
 * the supplied input. return NULL in case of error.
 *
 * If 'filter' is NULL, build an AND-filter using the filter components.
 */
__nis_ldap_search_t *
buildLdapSearch(char *base, int scope, int numFilterComps, char **filterComp,
                char *filter, char **attrs, int attrsonly, int isDN) {
        __nis_ldap_search_t     *ls;
        char                    **a;
        int                     i, na, err = 0;
```



```
        char                    *myself = "buildLdapSearch";

        ls = am(myself, sizeof (*ls));
        if (ls == 0)
                return (0);

        ls->base = sdup(myself, T, base);
        if (ls->base == 0 && base != 0)
                err++;
        ls->scope = scope;

        if (filterComp != 0 && numFilterComps > 0) {
                ls->filterComp = am(myself, numFilterComps *
                                    sizeof (ls->filterComp[0]));
                if (ls->filterComp == 0) {
                        err++;
                        numFilterComps = 0;
                }
                for (i = 0; i < numFilterComps; i++) {
                        ls->filterComp[i] = sdup(myself, T, filterComp[i]);
                        if (ls->filterComp[i] == 0 && filterComp[i] != 0)
                                err++;
                }
                ls->numFilterComps = numFilterComps;
                if (filter == 0) {
                        ls->filter = concatenateFilterComps(ls->numFilterComps,
                                        ls->filterComp);
                        if (ls->filter == 0)
                                err++;
                }
        } else {
                ls->filterComp = 0;
                ls->numFilterComps = 0;
                ls->filter = sdup(myself, T, filter);
                if (ls->filter == 0 && filter != 0)
                        err++;
        }

        if (attrs != 0) {
                for (na = 0, a = attrs; *a != 0; a++, na++);
                ls->attrs = am(myself, (na + 1) * sizeof (ls->attrs[0]));
                if (ls->attrs != 0) {
                        for (i = 0; i < na; i++) {
                                ls->attrs[i] = sdup(myself, T, attrs[i]);
                                if (ls->attrs[i] == 0 && attrs[i] != 0)
                                        err++;
                        }
                        ls->attrs[na] = 0;
                        ls->numAttrs = na;
                } else {
                        err++;
                }
        } else {
                ls->attrs = 0;
                ls->numAttrs = 0;
        }

        ls->attrsonly = attrsonly;
        ls->isDN = isDN;

        if (err > 0) {
                freeLdapSearch(ls);
                ls = 0;
        }

        return (ls);
```

```
 128 }
_____unchanged_portion_omitted_

 511 /* Accept a server/port indication, and call ldap_init() */
 512 static LDAP *
 513 ldapInit(char *srv, int port, bool_t use_ssl) {
 514         LDAP                    *ld;
 515         int                     ldapVersion = LDAP_VERSION3;
 516         int                     derefOption = LDAP_DEREF_ALWAYS;
 517         int                     timelimit = proxyInfo.search_time_limit;
 518         int                     sizelimit = proxyInfo.search_size_limit;
 518         char                    *myself = "ldapInit";

 520         if (srv == 0)
 521                 return (0);

 523         if (use_ssl) {
 524                 ld = ldapssl_init(srv, port, 1);
 525         } else {
 526                 ld = ldap_init(srv, port);
 527         }

 529         if (ld != 0) {
 530                 (void) ldap_set_option(ld, LDAP_OPT_PROTOCOL_VERSION,
 531                                        &ldapVersion);
 532                 (void) ldap_set_option(ld, LDAP_OPT_DEREF, &derefOption);
 533                 (void) ldap_set_option(ld, LDAP_OPT_REFERRALS, LDAP_OPT_OFF);
 534                 (void) ldap_set_option(ld, LDAP_OPT_TIMELIMIT, &timelimit);
 535                 (void) ldap_set_option(ld, LDAP_OPT_SIZELIMIT, &sizelimit);
 536                 (void) ldap_set_option(ld, LDAP_OPT_REBIND_ARG, 0);
 537         }

 539         return (ld);
 540 }
_____unchanged_portion_omitted_

 614 /*
 615  * Free 'lc' and all related memory. Caller must hold the exclusive lock.
 616  * Return LDAP_UNAVAILABLE upon success, in which case the caller mustn't
 617  * try to use the structure pointer in any way.
 618  */
 619 static int
 620 freeCon(__nis_ldap_conn_t *lc) {
 621         char                    *myself = "freeCon";

 622         if (!assertExclusive(lc))
 623                 return (LDAP_PARAM_ERROR);

 625         incrementRC(lc);

 627         /* Must be unused, unbound, and not on the 'ldapCon' list */
 628         if (lc->onList || lc->refCount != 1 || lc->isBound) {
 629                 lc->doDel++;
 630                 decrementRC(lc);
 631                 return (LDAP_BUSY);
 632         }

 634         sfree(lc->sp);
 635         sfree(lc->who);
 636         sfree(lc->cred);

 638         /* Delete structure with both mutex:es held */

 640         free(lc);

 642         return (LDAP_UNAVAILABLE);
```



```
 643 }
_____unchanged_portion_omitted_

1011 static int
1012 setupConList(char *serverList, char *who, char *cred, auth_method_t method) {
1013         char                    *sls, *sl, *s, *e;
1014         __nis_ldap_conn_t       *lc, *tmp;
1015         char                    *myself = "setupConList";

1017         if (serverList == 0)
1018                 return (LDAP_PARAM_ERROR);

1020         (void) rw_wrlock(&ldapConLock);

1022         if (ldapCon != 0) {
1023                 /* Assume we've already been called and done the set-up */
1024                 (void) rw_unlock(&ldapConLock);
1025                 return (LDAP_SUCCESS);
1026         }

1028         /* Work on a copy of 'serverList' */
1029         sl = sls = sdup(myself, T, serverList);
1030         if (sl == 0) {
1031                 (void) rw_unlock(&ldapConLock);
1032                 return (LDAP_NO_MEMORY);
1033         }

1035         /* Remove leading white space */
1036         for (; *sl == ' ' || *sl == '\t'; sl++);
1037         for (0; *sl == ' ' || *sl == '\t'; sl++);

1038         /* Create connection for each server on the list */
1039         for (s = sl; *s != '\0'; s = e+1) {
1040                 int     l;

1042                 /* Find end of server/port token */
1043                 for (e = s; *e != ' ' && *e != '\t' && *e != '\0'; e++);
1044                 if (*e != '\0')
1045                         *e = '\0';
1046                 else
1047                         e--;
1048                 l = slen(s);

1050                 if (l > 0) {
1051                         lc = createCon(s, who, cred, method, 0);
1052                         if (lc == 0) {
1053                                 free(sls);
1054                                 (void) rw_unlock(&ldapConLock);
1055                                 return (LDAP_NO_MEMORY);
1056                         }
1057                         lc->onList = 1;
1058                         if (ldapCon == 0) {
1059                                 ldapCon = lc;
1060                         } else {
1061                                 /* Insert at end of list */
1062                                 for (tmp = ldapCon; tmp->next != 0;
1063                                         tmp = tmp->next);
1064                                 tmp->next = lc;
1065                         }
1066                 }
1067         }

1069         free(sls);

1071         (void) rw_unlock(&ldapConLock);
```

```
1073         return (LDAP_SUCCESS);
1074 }
_____unchanged_portion_omitted_

2211 /*
2212  * Modify the specified 'dn' per the attribute names/values in 'rv'.
2213  * If 'rv' is NULL, we attempt to delete the entire entry.
2214  *
2215  * The 'objClassAttrs' parameter is needed if the entry must be added
2216  * (i.e., created), or a modify fails with an object class violation.
2217  *
2218  * If 'addFirst' is set, we try an add before a modify; modify before
2219  * add otherwise (ignored if we're deleting).
2220  */
2221 int
2222 ldapModify(char *dn, __nis_rule_value_t *rv, char *objClassAttrs,
2223                 int addFirst) {
2224         int                     stat, add = 0;
2225         LDAPMod                 **mods = 0;
2226         __nis_ldap_conn_t       *lc;
2227         struct timeval          tv;
2228         LDAPMessage             *msg = 0;
2230         char                    *myself = "ldapModify";
2229         int                     msgid;
2230         int                     lderr;
2231         char                    **referralsp = NULL;
2232         bool_t                  delete = FALSE;

2234         if (dn == 0)
2235                 return (LDAP_PARAM_ERROR);

2237         if ((lc = findCon(&stat)) == 0)
2238                 return (stat);

2240         if (rv == 0) {
2241                 delete = TRUE;
2242                 /* Simple case: if rv == 0, try to delete the entire entry */
2243                 msgid = ldap_delete(lc->ld, dn);
2244                 if (msgid == -1) {
2245                         (void) ldap_get_option(lc->ld, LDAP_OPT_ERROR_NUMBER,
2246                                                &stat);
2247                         goto cleanup;
2248                 }
2249                 tv = lc->deleteTimeout;
2250                 stat = ldap_result(lc->ld, msgid, 0, &tv, &msg);

2252                 if (stat == 0) {
2253                         stat = LDAP_TIMEOUT;
2254                 } else if (stat == -1) {
2255                         (void) ldap_get_option(lc->ld, LDAP_OPT_ERROR_NUMBER,
2256                                                &stat);
2257                 } else {
2258                         stat = ldap_parse_result(lc->ld, msg, &lderr, NULL,
2259                                 NULL, &referralsp, NULL, 0);
2260                         if (stat == LDAP_SUCCESS)
2261                                 stat = lderr;
2262                 }
2263                 if (proxyInfo.follow_referral == follow &&
2264                                 stat == LDAP_REFERRAL && referralsp != NULL) {
2265                         releaseCon(lc, stat);
2266                         if (msg != NULL)
2267                                 (void) ldap_msgfree(msg);
2268                         msg = NULL;
2269                         lc = findReferralCon(referralsp, &stat);
2270                         ldap_value_free(referralsp);
2271                         if (lc == NULL)
```



```
2272                                 goto cleanup;
2273                         msgid = ldap_delete(lc->ld, dn);
2274                         if (msgid == -1) {
2275                                 (void) ldap_get_option(lc->ld,
2276                                         LDAP_OPT_ERROR_NUMBER, &stat);
2277                                 goto cleanup;
2278                         }
2279                         stat = ldap_result(lc->ld, msgid, 0, &tv, &msg);
2280                         if (stat == 0) {
2281                                 stat = LDAP_TIMEOUT;
2282                         } else if (stat == -1) {
2283                                 (void) ldap_get_option(lc->ld,
2284                                         LDAP_OPT_ERROR_NUMBER, &stat);
2285                         } else {
2286                                 stat = ldap_parse_result(lc->ld, msg, &lderr,
2287                                         NULL, NULL, NULL, NULL, 0);
2288                                 if (stat == LDAP_SUCCESS)
2289                                         stat = lderr;
2290                         }
2291                 }
2292                 /* No such object means someone else has done our job */
2293                 if (stat == LDAP_NO_SUCH_OBJECT)
2294                         stat = LDAP_SUCCESS;
2295         } else {
2296                 if (addFirst) {
2297                         stat = ldapAdd(dn, rv, objClassAttrs, lc);
2298                         lc = NULL;
2299                         if (stat != LDAP_ALREADY_EXISTS)
2300                                 goto cleanup;
2301                         if ((lc = findCon(&stat)) == 0)
2302                                 return (stat);
2303                 }

2305                 /*
2306                  * First try the modify without specifying object classes
2307                  * (i.e., assume they're already present).
2308                  */
2309                 mods = search2LdapMod(rv, 0, 0);
2310                 if (mods == 0) {
2311                         stat = LDAP_PARAM_ERROR;
2312                         goto cleanup;
2313                 }

2315                 msgid = ldap_modify(lc->ld, dn, mods);
2316                 if (msgid == -1) {
2317                         (void) ldap_get_option(lc->ld, LDAP_OPT_ERROR_NUMBER,
2318                                                &stat);
2319                         goto cleanup;
2320                 }
2321                 tv = lc->modifyTimeout;
2322                 stat = ldap_result(lc->ld, msgid, 0, &tv, &msg);
2323                 if (stat == 0) {
2324                         stat = LDAP_TIMEOUT;
2325                 } else if (stat == -1) {
2326                         (void) ldap_get_option(lc->ld, LDAP_OPT_ERROR_NUMBER,
2327                                                &stat);
2328                 } else {
2329                         stat = ldap_parse_result(lc->ld, msg, &lderr, NULL,
2330                                 NULL, &referralsp, NULL, 0);
2331                         if (stat == LDAP_SUCCESS)
2332                                 stat = lderr;
2333                 }
2334                 if (proxyInfo.follow_referral == follow &&
2335                                 stat == LDAP_REFERRAL && referralsp != NULL) {
2336                         releaseCon(lc, stat);
2337                         if (msg != NULL)
```

```
2338                            (void) ldap_msgfree(msg);
2339                    msg = NULL;
2340                    lc = findReferralCon(referralsp, &stat);
2341                    ldap_value_free(referralsp);
2342                    referralsp = NULL;
2343                    if (lc == NULL)
2344                            goto cleanup;
2345                    msgid = ldap_modify(lc->ld, dn, mods);
2346                    if (msgid == -1) {
2347                            (void) ldap_get_option(lc->ld,
2348                                    LDAP_OPT_ERROR_NUMBER, &stat);
2349                            goto cleanup;
2350                    }
2351                    stat = ldap_result(lc->ld, msgid, 0, &tv, &msg);
2352                    if (stat == 0) {
2353                            stat = LDAP_TIMEOUT;
2354                    } else if (stat == -1) {
2355                            (void) ldap_get_option(lc->ld,
2356                                    LDAP_OPT_ERROR_NUMBER, &stat);
2357                    } else {
2358                            stat = ldap_parse_result(lc->ld, msg, &lderr,
2359                                    NULL, NULL, NULL, 0);
2360                            if (stat == LDAP_SUCCESS)
2361                                    stat = lderr;
2362                    }
2363            }

2365            /*
2366             * If the modify failed with an object class violation,
2367             * the most probable reason is that at least on of the
2368             * attributes we're modifying didn't exist before, and
2369             * neither did its object class. So, try the modify again,
2370             * but add the object classes this time.
2371             */
2372            if (stat == LDAP_OBJECT_CLASS_VIOLATION &&
2373                            objClassAttrs != 0) {
2374                    freeLdapMod(mods);
2375                    mods = 0;
2376                    stat = ldapModifyObjectClass(&lc, dn, rv,
2377                            objClassAttrs);
2378            }

2380            if (stat == LDAP_NO_SUCH_ATTRIBUTE) {
2381                    /*
2382                     * If there was at least one attribute delete, then
2383                     * the cause of this error could be that said attribute
2384                     * didn't exist in LDAP. So, do things the slow way,
2385                     * and try to delete one attribute at a time.
2386                     */
2387                    int                     d, numDelete, st;
2388                    __nis_rule_value_t      *rvt;

2390                    for (d = 0, numDelete = 0; d < rv->numAttrs; d++) {
2391                            if (rv->attrVal[d].numVals < 0)
2392                                    numDelete++;
2393                    }

2395                    /* If there's just one, we've already tried */
2396                    if (numDelete <= 1)
2397                            goto cleanup;

2399                    /* Make a copy of the rule value */
2400                    rvt = initRuleValue(1, rv);
2401                    if (rvt == 0)
2402                            goto cleanup;
```



```
2404                    /*
2405                     * Remove all delete attributes from the tmp
2406                     * rule value.
2407                     */
2408                    for (d = 0; d < rv->numAttrs; d++) {
2409                            if (rv->attrVal[d].numVals < 0) {
2410                                    delAttrFromRuleValue(rvt,
2411                                            rv->attrName[d]);
2412                            }
2413                    }

2415                    /*
2416                     * Now put the attributes back in one by one, and
2417                     * invoke ourselves.
2418                     */
2419                    for (d = 0; d < rv->numAttrs; d++) {
2420                            if (rv->attrVal[d].numVals >= 0)
2421                                    continue;
2422                            st = addAttr2RuleValue(rv->attrVal[d].type,
2423                                    rv->attrName[d], 0, 0, rvt);
2424                            if (st != 0) {
2425                                    logmsg(MSG_NOMEM, LOG_ERR,
2426                                    "%s: Error deleting \"%s\" for \"%s\"",
2427                                            NIL(rv->attrName[d]), NIL(dn));
2428                                    stat = LDAP_NO_MEMORY;
2429                                    freeRuleValue(rvt, 1);
2430                                    goto cleanup;
2431                            }
2432                            stat = ldapModify(dn, rvt, objClassAttrs, 0);
2433                            if (stat != LDAP_SUCCESS &&
2434                                    stat != LDAP_NO_SUCH_ATTRIBUTE) {
2435                                    freeRuleValue(rvt, 1);
2436                                    goto cleanup;
2437                            }
2438                            delAttrFromRuleValue(rvt, rv->attrName[d]);
2439                    }

2441                    /*
2442                     * If we got here, then all attributes that should
2443                     * be deleted either have been, or didn't exist. For
2444                     * our purposes, the latter is as good as the former.
2445                     */
2446                    stat = LDAP_SUCCESS;
2447                    freeRuleValue(rvt, 1);
2448            }

2450            if (stat == LDAP_NO_SUCH_OBJECT && !addFirst) {
2451                    /*
2452                     * Entry doesn't exist, so try an ldap_add(). If the
2453                     * ldap_add() also fails, that could be because someone
2454                     * else added it between our modify and add operations.
2455                     * If so, we consider that foreign add to be
2456                     * authoritative (meaning we don't retry our modify).
2457                     *
2458                     * Also, if all modify operations specified by 'mods'
2459                     * are deletes, LDAP_NO_SUCH_OBJECT is a kind of
2460                     * success; we certainly don't want to create the
2461                     * entry.
2462                     */
2463                    int     allDelete;
2464                    LDAPMod **m;

2466                    for (m = mods, allDelete = 1; *m != 0 && allDelete;
2467                                    m++) {
2468                            if (((*m)->mod_op & LDAP_MOD_DELETE) == 0)
2469                                    allDelete = 0;
```

```
2470                        }

2472                        add = 1;

2474                        if (allDelete) {
2475                                stat = LDAP_SUCCESS;
2476                        } else if (objClassAttrs == 0) {
2477                                /* Now we need it, so this is fatal */
2478                                stat = LDAP_PARAM_ERROR;
2479                        } else {
2480                                stat = ldapAdd(dn, rv, objClassAttrs, lc);
2481                                lc = NULL;
2482                        }
2483                }
2484        }

2486 cleanup:
2487        if (stat != LDAP_SUCCESS) {
2488                logmsg(MSG_NOTIMECHECK, LOG_INFO,
2489                        "%s(0x%x (%s), \"%s\") => %d (%s)\n",
2490                        !delete ? (add ? "ldap_add" : "ldap_modify") :
2491                                "ldap_delete",
2492                        lc != NULL ? lc->ld : 0,
2493                        lc != NULL ? NIL(lc->sp) : "nil",
2494                        dn, stat, ldap_err2string(stat));
2495        }

2497        releaseCon(lc, stat);
2498        freeLdapMod(mods);
2499        if (msg != 0)
2500                (void) ldap_msgfree(msg);

2502        return (stat);
2503 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   25948 Fri Jul 24 12:28:11 2015
new/usr/src/lib/libnisdb/ldap_ruleval.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


#include <lber.h>
#include <ldap.h>
#include <strings.h>

#include "nisdb_mt.h"

#include "ldap_util.h"
#include "ldap_val.h"
#include "ldap_attr.h"
#include "ldap_ldap.h"
#include "ldap_ruleval.h"


/*
 * Free an array of 'count' rule-value elements.
 */
void
freeRuleValue(__nis_rule_value_t *rv, int count) {
        int     n, i, j;

        if (rv == 0)
                return;

        for (n = 0; n < count; n++) {

                if (rv[n].colName != 0) {
                        for (i = 0; i < rv[n].numColumns; i++) {
                                sfree(rv[n].colName[i]);
                        }
                        free(rv[n].colName);
                }
                if (rv[n].colVal != 0) {
                        for (i = 0; i < rv[n].numColumns; i++) {
                                for (j = 0; j < rv[n].colVal[i].numVals; j++) {
```



```
                                        sfree(rv[n].colVal[i].val[j].value);
                                }
                                if (rv[n].colVal[i].numVals > 0)
                                        sfree(rv[n].colVal[i].val);
                        }
                        free(rv[n].colVal);
                }

                if (rv[n].attrName != 0) {
                        for (i = 0; i < rv[n].numAttrs; i++) {
                                sfree(rv[n].attrName[i]);
                        }
                        free(rv[n].attrName);
                }
                if (rv[n].attrVal != 0) {
                        for (i = 0; i < rv[n].numAttrs; i++) {
                                for (j = 0; j < rv[n].attrVal[i].numVals;
                                                j++) {
                                        sfree(rv[n].attrVal[i].val[j].value);
                                }
                                if (rv[n].attrVal[i].numVals > 0)
                                        sfree(rv[n].attrVal[i].val);
                        }
                        free(rv[n].attrVal);
                }

        }
        sfree(rv);
}
```

_____***unchanged_portion_omitted_***

```
static const __nis_rule_value_t rvZero = {0};

/*
 * Grow 'old' from 'oldCount' to 'newCount' elements, initialize the
 * new portion to 'rvIn' (empty if not supplied), and return a pointer
 * to the result. Following a call to this function, the caller must
 * refer only to the returned array, not to 'old'.
 */
__nis_rule_value_t *
growRuleValue(int oldCount, int newCount, __nis_rule_value_t *old,
                __nis_rule_value_t *rvIn) {
        __nis_rule_value_t      *rv;
        int                     i;
        int                     i, j;
        char                    *myself = "growRuleValue";

        if (newCount <= 0 || newCount <= oldCount)
                return (old);

        if (oldCount <= 0) {
                oldCount = 0;
                old = 0;
        }

        if (rvIn == 0)
                rvIn = (__nis_rule_value_t *)&rvZero;

        rv = realloc(old, newCount * sizeof (rv[0]));
        if (rv == 0) {
                logmsg(MSG_NOMEM, LOG_ERR,
                        "%s: realloc(%d ((%d+%d)*%d)) => 0",
                        myself, (oldCount+newCount) * sizeof (rv[0]),
                        oldCount, newCount, sizeof (rv[0]));
                freeRuleValue(old, oldCount);
                return (0);
```

```
135         }

137         (void) memset(&rv[oldCount], 0, (newCount-oldCount)*sizeof (rv[0]));

139         for (i = oldCount; i < newCount; i++) {
140                 rv[i].numColumns = rvIn->numColumns;
141                 if (rv[i].numColumns > 0) {
142                         rv[i].colName = cloneName(rvIn->colName,
143                                 rv[i].numColumns);
144                         rv[i].colVal = cloneValue(rvIn->colVal,
145                                 rv[i].numColumns);
146                 }
147                 if (rv[i].numColumns > 0 &&
148                                 (rv[i].colName == 0 || rv[i].colVal == 0)) {
149                         freeRuleValue(rv, i);
150                         return (0);
151                 }
152                 rv[i].numAttrs = rvIn->numAttrs;
153                 rv[i].attrName = cloneName(rvIn->attrName, rv[i].numAttrs);
154                 rv[i].attrVal = cloneValue(rvIn->attrVal, rv[i].numAttrs);
155                 if (rv[i].numAttrs > 0 &&
156                         (rv[i].attrName == 0 || rv[i].attrVal == 0)) {
157                         freeRuleValue(rv, i);
158                         return (0);
159                 }
160         }

162         return (rv);
163 }
```

*unchanged_portion_omitted_*

```
420 /*
421  * Given a table mapping, a NIS+ DB query, and (optionally) an existing
422  * and compatible __nis_rule_value_t, return a new __nis_rule_value_t
423  * with the values from the query added.
424  */
425 __nis_rule_value_t *
426 buildNisPlusRuleValue(__nis_table_mapping_t *t, db_query *q,
427                         __nis_rule_value_t *rv) {
428         int                     i;
428         __nis_single_value_t    *sv;
429         char                    *myself = "buildNisPlusRuleValue";

430         if (t == 0 || q == 0)
431                 return (0);

433         rv = initRuleValue(1, rv);
434         if (rv == 0)
435                 return (0);

437         for (i = 0; i < q->components.components_len; i++) {
439                 int     ic;
440                 int     iv, v, dup;
441                 int     len;

439                 /* Ignore out-of-range column index */
440                 if (q->components.components_val[i].which_index >=
441                                 t->numColumns)
442                         continue;

444                 /*
445                  * Add the query value. A NULL value indicates deletion,
446                  * but addCol2RuleValue() takes care of that for us.
447                  */
448                 if (addCol2RuleValue(vt_string,
449                                 t->column[q->components.components_val[i].
```



```
450                                         which_index],
451                                 q->components.components_val[i].index_value->
452                                         itemvalue.itemvalue_val,
453                                 q->components.components_val[i].index_value->
454                                         itemvalue.itemvalue_len, rv) != 0) {
455                         freeRuleValue(rv, 1);
456                         rv = 0;
457                         break;
458                 }
459         }

461         return (rv);
462 }
```

*unchanged_portion_omitted_*

```
601 /*
602  * Derive values for the LDAP attributes specified by the rule 'r',
603  * and add them to the rule-value 'rv'.
604  *
605  * If 'doAssign' is set, out-of-context assignments are performed,
606  * otherwise not.
607  */
608 __nis_rule_value_t *
609 addLdapRuleValue(__nis_table_mapping_t *t,
610                         __nis_mapping_rule_t *r,
611                         __nis_mapping_item_type_t lnative,
612                         __nis_mapping_item_type_t rnative,
613                         __nis_rule_value_t *rv,
614                         int doAssign, int *stat) {
615         int                     i, j;
620         char                    **new;
616         __nis_value_t           *rval, *lval;
622         __nis_buffer_t          b = {0, 0};
617         __nis_mapping_item_t    *litem;
618         int                     numItems;
619         char                    **dn = 0;
620         int                     numDN = 0;
621         char                    *myself = "addLdapRuleValue";


624         /* Do we have the required values ? */
625         if (rv == 0)
626                 return (0);

628         /*
629          * Establish appropriate search base. For rnative == mit_nisplus,
630          * we're deriving LDAP attribute values from NIS+ columns; in other
631          * words, we're writing to LDAP, and should use the write.base value.
632          */
633         __nisdb_get_tsd()->searchBase = (rnative == mit_nisplus) ?
634                 t->objectDN->write.base : t->objectDN->read.base;

636         /* Set escapeFlag if LHS is "dn" to escape special chars */
637         if (yp2ldap && r->lhs.numElements == 1 &&
638                 r->lhs.element->type == me_item &&
639                 r->lhs.element->element.item.type == mit_ldap &&
640                 strcasecmp(r->lhs.element->element.item.name, "dn") == 0) {
641                         __nisdb_get_tsd()->escapeFlag = '1';
642         }

644         /* Build the RHS value */
645         rval = buildRvalue(&r->rhs, rnative, rv, stat);

647         /* Reset escapeFlag */
648         __nisdb_get_tsd()->escapeFlag = '\0';
```

```
650         if (rval == 0)
651                 return (rv);

653         /*
654          * Special case: If we got no value for the RHS (presumably because
655          * we're missing one or more item values), we don't produce an lval.
656          * Note that this isn't the same thing as an empty value, which we
657          * faithfully try to transmit to LDAP.
658          */
659         if (rval->numVals == 1 && rval->val[0].value == 0) {
660                 freeValue(rval, 1);
661                 return (rv);
662         }

664         /* Obtain the LHS item names */
665         litem = buildLvalue(&r->lhs, &rval, &numItems);
666         if (litem == 0) {
667                 freeValue(rval, 1);
668                 return (rv);
669         }

671         /* Get string representations of the LHS item names */
672         lval = 0;
673         for (i = 0; i < numItems; i++) {
674                 __nis_value_t   *tmpval, *old;

676                 tmpval = getMappingItem(&litem[i], lnative, 0, 0, NULL);

678                 /*
679                  * If the LHS item is out-of-context, we do the
680                  * assignment right here.
681                  */
682                 if (doAssign && litem[i].type == mit_ldap &&
683                                 litem[i].searchSpec.triple.scope !=
684                                         LDAP_SCOPE_UNKNOWN &&
685                                 slen(litem[i].searchSpec.triple.base) > 0 &&
686                                 (slen(litem[i].searchSpec.triple.attrs) > 0 ||
687                                 litem[i].searchSpec.triple.element != 0)) {
688                         int     stat;

690                         if (dn == 0)
691                                 dn = findDNs(myself, rv, 1,
692                                         t->objectDN->write.base,
693                                         &numDN);

695                         stat = storeLDAP(&litem[i], i, numItems, rval,
696                                 t->objectDN, dn, numDN);
697                         if (stat != LDAP_SUCCESS) {
698                                 char    *iname = "<unknown>";

700                                 if (tmpval != 0 &&
701                                                 tmpval->numVals == 1)
702                                         iname = tmpval->val[0].value;
703                                 logmsg(MSG_NOTIMECHECK, LOG_ERR,
704                                         "%s: LDAP store \"%s\": %s",
705                                         myself, iname,
706                                         ldap_err2string(stat));
707                         }

709                         freeValue(tmpval, 1);
710                         continue;
711                 }

713                 old = lval;
714                 lval = concatenateValues(old, tmpval);
715                 freeValue(tmpval, 1);
```



```
716                 freeValue(old, 1);
717         }

719         /* Don't need the LHS items themselves anymore */
720         freeMappingItem(litem, numItems);

722         /*
723          * If we don't have an 'lval' (probably because all litem[i]:s
724          * were out-of-context assignments), we're done.
725          */
726         if (lval == 0 || lval->numVals <= 0) {
727                 freeValue(lval, 1);
728                 freeValue(rval, 1);
729                 return (rv);
730         }

732         for (i = 0, j = 0; i < lval->numVals; i++) {
733                 /* Special case: rval->numVals < 0 means deletion */
734                 if (rval->numVals < 0) {
735                         (void) addAttr2RuleValue(rval->type,
736                                 lval->val[i].value, 0, 0, rv);
737                         continue;
738                 }
739                 /* If we're out of values, repeat the last one */
740                 if (j >= rval->numVals)
741                         j = (rval->numVals > 0) ? rval->numVals-1 : 0;
742                 for (; j < rval->numVals; j++) {
748                 for (0; j < rval->numVals; j++) {
743                         /*
744                          * If this is the 'dn', and the value ends in a
745                          * comma, append the appropriate search base.
746                          */
747                         if (strcasecmp("dn", lval->val[i].value) == 0 &&
748                                         lastChar(&rval->val[j]) == ',' &&
749                                         t->objectDN->write.scope !=
750                                                 LDAP_SCOPE_UNKNOWN) {
751                                 void    *nval;
752                                 int     nlen = -1;

754                                 nval = appendString2SingleVal(
755                                         t->objectDN->write.base, &rval->val[j],
756                                         &nlen);
757                                 if (nval != 0 && nlen >= 0) {
758                                         sfree(rval->val[j].value);
759                                         rval->val[j].value = nval;
760                                         rval->val[j].length = nlen;
761                                 }
762                         }
763                         (void) addAttr2RuleValue(rval->type,
764                                 lval->val[i].value, rval->val[j].value,
765                                 rval->val[j].length, rv);
766                         /*
767                          * If the lval is multi-valued, go on to the
768                          * other values; otherwise, quit (but increment
769                          * the 'rval' value index).
770                          */
771                         if (!lval->repeat) {
772                                 j++;
773                                 break;
774                         }
775                 }
776         }

778         /* Clean up */
779         freeValue(lval, 1);
780         freeValue(rval, 1);
```

```
 782         return (rv);
 783 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   56395 Fri Jul 24 12:28:12 2015
new/usr/src/lib/libnisdb/ldap_val.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2015 Gary Mills
  23  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */


  28 #include <lber.h>
  29 #include <ldap.h>
  30 #include <strings.h>
  31 #include <errno.h>

  33 #include "nisdb_mt.h"

  35 #include "ldap_util.h"
  36 #include "ldap_op.h"
  37 #include "ldap_ruleval.h"
  38 #include "ldap_attr.h"
  39 #include "ldap_val.h"
  40 #include "ldap_ldap.h"

  42 extern int yp2ldap;


  45 __nis_mapping_format_t *
  46 cloneMappingFormat(__nis_mapping_format_t *m) {
  47         __nis_mapping_format_t  *new;
  48         int                     i, nf, err;
  49         char                    *myself = "cloneMappingFormat";

  51         if (m == 0)
  52                 return (0);

  54         for (nf = 0; m[nf].type != mmt_end; nf++);
  55         nf++;

  57         new = am(myself, nf * sizeof (new[0]));
  58         if (new == 0)
  59                 return (0);

  61         /* Copy the whole array */
```



```
  62         memcpy(new, m, nf * sizeof (new[0]));

  64         /* Make copies of allocated stuff */
  65         for (i = 0, err = 0; i < nf; i++) {
  66                 switch (m[i].type) {
  67                 case mmt_string:
  68                         new[i].match.string = sdup(myself, T,
  69                                                    m[i].match.string);
  70                         if (new[i].match.string == 0 && m[i].match.string != 0)
  71                                 err++;
  72                         break;
  73                 case mmt_single:
  74                         new[i].match.single.lo =
  75                                 am(myself, m[i].match.single.numRange *
  76                                         sizeof (new[i].match.single.lo[0]));
  77                         new[i].match.single.hi =
  78                                 am(myself, m[i].match.single.numRange *
  79                                         sizeof (new[i].match.single.hi[0]));
  80                         if (new[i].match.single.lo != 0)
  81                                 memcpy(new[i].match.single.lo,
  82                                         m[i].match.single.lo,
  83                                         m[i].match.single.numRange);
  84                         else if (m[i].match.single.lo != 0)
  85                                 err++;
  86                         if (new[i].match.single.hi != 0)
  87                                 memcpy(new[i].match.single.hi,
  88                                         m[i].match.single.hi,
  89                                         m[i].match.single.numRange);
  90                         else if (m[i].match.single.hi != 0)
  91                                 err++;
  92                         break;
  93                 case mmt_berstring:
  94                         new[i].match.berString = sdup(myself, T,
  95                                                    m[i].match.berString);
  96                         if (new[i].match.berString == 0 &&
  97                                         m[i].match.berString != 0)
  98                                 err++;
  99                         break;
 100                 case mmt_item:
 101                 case mmt_limit:
 102                 case mmt_any:
 103                 case mmt_begin:
 104                 case mmt_end:
 105                 default:
 106                         break;
 107                 }
 108         }

 110         /* If there were memory allocation errors, free the copy */
 111         if (err > 0) {
 112                 freeMappingFormat(new);
 113                 new = 0;
 114         }

 116         return (new);
 117 }
_____unchanged_portion_omitted_

2068 /*
2069  * Perform a match operation. For example, given the rule
2070  *      ("{%s}%s", auth_name, public_data)=nisPublicKey
2071  * and assuming that 'nisPublicKey' has the value "{dh640-0}abcdef12345",
2072  * assign "dh640-0" to 'auth_name' and "abcdef12345" to 'public_data'.
2073  *
2074  * Note that this function doesn't perform the actual assignment. Rather,
2075  * it returns an array of __nis_value_t's, with element zero of the value
```

```
2076  * array being the new value of the first matched item, element one the
2077  * value of the second matched item, etc. In the example above, we'd
2078  * return a value array with two elements.
2079  *
2080  * If there is more than one input value (inVal->numVals > 1), the
2081  * output array elements will also be multi-valued.
2082  *
2083  * f            The match format
2084  * inVal        Input value(s)
2085  * numVal       Number of elements in the output value array
2086  * sepset       List of separators
2087  * outstr       Points to the updated position upto which the
2088  *              input string has been matched
2089  */
2090 __nis_value_t **
2091 matchMappingItem(__nis_mapping_format_t *f, __nis_value_t *inVal,
2092                 int *numVals, char *sepset, char **outstr) {
2093         __nis_value_t           **v = 0;
2094         int                     i, n, ni, numItems, nf;
2093         int                     i, n, ni, numItems, nf, nv = 0;
2095         char                    *str, *valstr;
2096         __nis_mapping_format_t  *ftmp;
2097         char                    *myself = "matchMappingItem";

2099         if (f == 0 ||
2100                 inVal == 0 || inVal->numVals < 1 || inVal->type != vt_string)
2101                 return (0);

2103         /* Count the number of format elements and items in the format */
2104         for (nf = numItems = 0, ftmp = f; ftmp->type != mmt_end; ftmp++) {
2105                 nf++;

2107                 /*
2108                  * Count mmt_item and mmt_berstring (used by N2L to
2109                  * represent address %a)
2110                  */
2111                 if (ftmp->type == mmt_item)
2112                         numItems++;
2113                 else if (ftmp->type == mmt_berstring && ftmp->match.berString &&
2114                         ftmp->match.berString[0] == 'a')
2115                         numItems++;
2116         }
2117         /* Count the mmt_end as well */
2118         nf++;

2120         /*
2121          * If no items, there will be no values. This isn't exactly an error
2122          * from the limited point of view of this function, so we return a
2123          * __nis_value_t with zero values.
2124          */
2125         if (numItems <= 0) {
2126                 v = am(myself, sizeof (v[0]));
2127                 if (v == 0)
2128                         return (0);
2129                 v[0] = am(myself, sizeof (*v[0]));
2130                 if (v[0] == 0) {
2131                         sfree(v);
2132                         return (0);
2133                 }
2134                 v[0]->type = vt_string;
2135                 v[0]->numVals = 0;
2136                 v[0]->val = 0;
2137                 if (numVals != 0)
2138                         *numVals = 1;
2139                 return (v);
2140         }
```



```
2142         /* Allocate and initialize the return array */
2143         v = am(myself, numItems * sizeof (v[0]));
2144         if (v == 0)
2145                 return (0);
2146         for (n = 0; n < numItems; n++) {
2147                 v[n] = am(myself, sizeof (*v[n]));
2148                 if (v[n] == 0) {
2149                         int     j;

2151                         for (j = 0; j < n; j++)
2152                                 freeValue(v[j], 1);
2153                         sfree(v);
2154                         return (0);
2155                 }
2156                 v[n]->type = vt_string;
2157                 v[n]->numVals = 0;
2158                 v[n]->val = am(myself, inVal->numVals * sizeof (v[n]->val[0]));
2159                 if (v[n]->val == 0) {
2160                         int     j;

2162                         for (j = 0; j < n; j++)
2163                                 freeValue(v[j], 1);
2164                         sfree(v);
2165                         return (0);
2166                 }
2167                 for (i = 0; i < inVal->numVals; i++) {
2168                         v[n]->val[i].length = 0;
2169                         v[n]->val[i].value = 0;
2170                 }
2171         }

2173         /* For each input value, perform the match operation */
2174         for (i = 0; i < inVal->numVals; i++) {
2175                 str = inVal->val[i].value;
2176                 if (str == 0)
2177                         continue;
2178                 for (n = 0, ni = 0; n < nf; n++) {
2179                         valstr = 0;
2180                         str = scanMappingFormat(f, n, nf, str, &valstr,
2181                                 0, sepset);
2182                         if (str == 0)
2183                                 break;
2184                         if (valstr != 0 && ni < numItems &&
2185                                         v[ni]->numVals < inVal->numVals) {
2186                                 v[ni]->val[v[ni]->numVals].value = valstr;
2187                                 v[ni]->val[v[ni]->numVals].length =
2188                                                         strlen(valstr) + 1;
2189                                 v[ni]->numVals++;
2190                                 ni++;
2191                         } else if (valstr != 0) {
2192                                 sfree(valstr);
2193                         }
2194                 }
2195                 if (str == 0) {
2196                         for (n = 0; n < numItems; n++)
2197                                 freeValue(v[n], 1);
2198                         sfree(v);
2199                         return (0);
2200                 }
2201         }

2203         if (numVals != 0)
2204                 *numVals = numItems;

2206         /*
```

```
2207          * Update the return string upto the point it has been matched
2208          * This string will be used by the N2L code in its next call
2209          * to this function
2210          */
2211         if (outstr != 0)
2212                 *outstr = str;

2214         return (v);
2215 }
```

_____***unchanged_portion_omitted_***

****
19694 Fri Jul 24 12:28:12 2015
new/usr/src/lib/libnisdb/ldap_xdr.c
5910 libnisdb won't build with modern GCC
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright (c) 2001 by Sun Microsystems, Inc.
 * All rights reserved.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <string.h>
#include <sys/syslog.h>
#include <sys/types.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#include <rpcsvc/nis.h>

#include "db_mindex_c.h"

#include "ldap_xdr.h"
#include "ldap_util.h"

#include "nis_clnt.h"
extern bool_t xdr_nis_object();

/*
 * In order not to change the on-disk NIS+ DB format, we need make sure
 * that XDR does nothing for the new structures added to various C++
 * classes.
 */

bool_t
xdr___nis_table_mapping_t(XDR *xdrs, void *t) {
        return (TRUE);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    2182 Fri Jul 24 12:28:12 2015
new/usr/src/lib/libnisdb/ldap_xdr.h
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright (c) 2001 by Sun Microsystems, Inc.
 * All rights reserved.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifndef _LDAP_XDR_H
#define _LDAP_XDR_H

#include <rpcsvc/nis.h>

#ifdef  __cplusplus
extern "C" {
#endif

/* Structure used to hide flag/counter from rpcgen */
typedef struct {
        long    flag;
} __nisdb_flag_t;
```

_____***unchanged_portion_omitted_***

```

/* Exported functions */
bool_t          xdr_nis_name_abbrev(XDR *xdrs, nis_name *namep,
                        nis_name domainname);
bool_t          xdr_nis_fetus_object(XDR *xdrs, nis_object *objp,
                        nis_object *tobj);
entry_obj       *makePseudoEntryObj(nis_object *obj, entry_obj *eo,
                                nis_object *tobj);
nis_object      *unmakePseudoEntryObj(entry_obj *e, nis_object *tobj);
void            *xdrNisObject(nis_object *obj, entry_obj **ea, int numEa,
                        int *xdrLenP);
nis_object      *unXdrNisObject(void *buf, int bufLen, entry_obj ***eaP,
                        int *numEaP);
void            freeEntryObjArray(entry_obj **ea, int numEa);
bool_t          sameNisPlusObj(nis_object *o1, nis_object *o2);
bool_t          sameNisPlusPseudoObj(nis_object *o1, entry_obj *e2);
bool_t          xdr___nisdb_rwlock_t(XDR *, void *);
bool_t          xdr___nisdb_flag_t(XDR *, void *);
```



```
bool_t          xdr___nisdb_ptr_t(XDR *, void *);
bool_t          xdr___nis_table_mapping_t(XDR *, void *);

#ifdef  __cplusplus
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   45121 Fri Jul 24 12:28:12 2015
new/usr/src/lib/libnisdb/nis_db.cc
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 *      nis_db.cc
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2015 RackTop Systems.
 */


#include <sys/param.h>
#include <strings.h>
#include <syslog.h>
#include "nisdb_mt.h"
#include "db_headers.h"
#include "db_entry.h"
#include "db.h"
#include "db_dictionary.h"
#include "db_pickle.h"
#include "nis_db.h"
#include "nis_ldap.h"
#include "ldap_util.h"
#include "ldap_parse.h"
#include "ldap_glob.h"
#include "ldap_xdr.h"
#include "ldap_glob.h"

db_dictionary   curdict;
db_dictionary   tempdict; /* a temporary one */

db_dictionary *InUseDictionary = &curdict;
db_dictionary *FreeDictionary = &tempdict;

extern "C" {
static db_result        *db_add_entry_x(char *tab, int numattrs,
                                        nis_attr *attrname, entry_obj * newobj,
                                        int skiplog, int nosync);
db_status               db_table_exists(char *table_name);

/*
 * (Imported from rpc.nisd/nis_xx_proc.c)
```



```
 *
 * 'tbl_prototype' is used to create a table that holds a directory.
 */
static table_col cols[2] = {
        {(char *)"object", TA_BINARY+TA_XDR, 0},
        {(char *)"name", TA_CASE+TA_SEARCHABLE, 0}
};
```

*unchanged_portion_omitted_*

```
extern "C" {

bool_t
db_in_dict_file(char *name)
{
        return (InUseDictionary->find_table_desc(name) != NULL);
        return ((bool_t) InUseDictionary->find_table_desc(name));

}
```

*unchanged_portion_omitted_*

```
**********************************************************
   44682 Fri Jul 24 12:28:12 2015
new/usr/src/lib/libnisdb/nis_parse_ldap_conf.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <locale.h>
#include <sys/stat.h>
#include <lber.h>
#include <ldap.h>
#include <deflt.h>

#include "ldap_map.h"

#include "ldap_parse.h"
#include "ldap_glob.h"
#include "nis_parse_ldap_conf.h"

__nis_ldap_proxy_info   proxyInfo               =
        {NULL, (auth_method_t)NO_VALUE_SET, (tls_method_t)NO_VALUE_SET, NULL,
                NULL, NULL, NULL, NULL, (follow_referral_t)NO_VALUE_SET};
__nis_config_t          ldapConfig;
__nisdb_table_mapping_t ldapDBTableMapping;
__nis_table_mapping_t   *ldapTableMapping       = NULL;
__yp_domain_context_t   ypDomains;

parse_error             p_error                 = no_parse_error;
int                     cur_line_num            = 0;
int                     start_line_num          = 0;
int                     seq_num                 = 0;
const char              *warn_file              = NULL;

char                    _key_val[38];
const char              *command_line_source    = NULL;
```



```
const char              *file_source            = NULL;
const char              *ldap_source            = NULL;

static
const char *const       *cmdline_config         = NULL;
static bool_t           got_config_data         = FALSE;

/* high level parsing functions functions */
static int parse_ldap_cmd_line(const char *const *cmdline_options,
    __nis_ldap_proxy_info *proxy_info, __nis_config_t *nis_config,
    __nis_table_mapping_t **table_mapping, __nis_config_info_t *config_info,
    __nisdb_table_mapping_t *table_info);
static int parse_ldap_default_conf(__nis_ldap_proxy_info *proxy_info,
    __nis_config_t *nis_config, __nis_config_info_t *config_info,
    __nisdb_table_mapping_t *table_info);
static int parse_ldap_config_file(const char *config_file,
    __nis_ldap_proxy_info *proxy_info, __nis_config_t *nis_config,
    __nis_table_mapping_t **table_mapping, __nis_config_info_t *config_info,
    __nisdb_table_mapping_t *table_info);
static int parse_ldap_config_dn_attrs(__nis_ldap_proxy_info *proxy_info,
    __nis_config_t *nis_config, __nis_table_mapping_t **table_mapping,
    __nis_config_info_t *config_info, __nisdb_table_mapping_t *table_info);
static int yp_parse_ldap_default_conf(__nis_ldap_proxy_info *proxy_info,
        __nis_config_t *nis_config, __nis_config_info_t *config_info,
        __nisdb_table_mapping_t *table_info);

/* Forward declarations */
int yp_parse_ldap_config_file(const char *, __nis_ldap_proxy_info *,
    __nis_config_t *, __nis_table_mapping_t **, __nis_config_info_t *,
    __nisdb_table_mapping_t *,  __yp_domain_context_t *);


/* helper functions */
static config_key get_attrib_num_cmdline(const char *s,
    const char **begin_s, const char **end_s);
static config_key get_file_attr_val(int fd, char **attr_val);
static void get_attribute_list(
        const __nis_ldap_proxy_info *proxy_info,
        const __nis_config_t *nis_config,
        const __nis_config_info_t *config_info,
        const __nisdb_table_mapping_t *table_info,
        char **ldap_config_attributes);

/*
 * FUNCTION:    parse_ldap_migration
 *
 *      Parses the information for LDAP. The values are first
 *      obtained from the command line, secondly from the preference
 *      file, and finally from an LDAP profile (if so configured in
 *      the command line or preference file). Any unset values will
 *      be set to their default values.
 *
 *      If no command line options, no settings in the /etc/default
 *  configuration file, and no mapping file, then no mapping
 *  should be used.
 *
 * RETURN VALUE:
 *                      0       Success
 *                      -1      Config file stat/open or parse error
 *                      1       No mapping should be used.
 *
 * INPUT:               command line parameters, configuration file
 */

int
parse_ldap_migration(
```

```
128         const char *const       *cmdline_options,
129         const char              *config_file)
130 {
131         int                     rc      = 0;
132         __nis_config_info_t     config_info
133                                 = {NULL, NULL, (auth_method_t)NO_VALUE_SET,
134                                        (tls_method_t)NO_VALUE_SET, NULL,
135                                        NULL, NULL};
136         struct stat             buf;
131         int i = 0;

138         p_error = no_parse_error;

140         if (verbose)
141                 report_info("Getting LDAP configuration", NULL);

143         initialize_parse_structs(&proxyInfo, &ldapConfig, &ldapDBTableMapping);

145         if (yp2ldap)
146                 initialize_yp_parse_structs(&ypDomains);

148         if (cmdline_options != NULL) {
149                 got_config_data = TRUE;
150                 /* NIS to LDAP does not read command line attributes */
151                 if (!yp2ldap)
152                         rc = parse_ldap_cmd_line(cmdline_options, &proxyInfo,
153                             &ldapConfig, &ldapTableMapping, &config_info,
154                             &ldapDBTableMapping);
155                 else
156                         rc = 0;
157         }

159         if (rc == 0) {
160                 if (yp2ldap)
161                         rc = yp_parse_ldap_default_conf(&proxyInfo, &ldapConfig,
162                             &config_info, &ldapDBTableMapping);
163                 else
164                         rc = parse_ldap_default_conf(&proxyInfo, &ldapConfig,
165                             &config_info, &ldapDBTableMapping);
166         }

168         if (config_file == NULL) {
169                 if (yp2ldap) {
170                         if (stat(YP_DEFAULT_MAPPING_FILE, &buf) == 0)
171                                 config_file = YP_DEFAULT_MAPPING_FILE;
172                 } else {
173                         if (stat(DEFAULT_MAPPING_FILE, &buf) == 0)
174                                 config_file = DEFAULT_MAPPING_FILE;
175                 }
176         }

178         if (rc == 0 && config_file != NULL) {
179                 got_config_data = TRUE;
180                 warn_file = config_file;
181                 cmdline_config = cmdline_options;
182                 if (yp2ldap)
183                         rc = yp_parse_ldap_config_file(config_file, &proxyInfo,
184                             &ldapConfig, &ldapTableMapping, &config_info,
185                             &ldapDBTableMapping, &ypDomains);
186                 else
187                         rc = parse_ldap_config_file(config_file, &proxyInfo,
188                             &ldapConfig, &ldapTableMapping, &config_info,
189                             &ldapDBTableMapping);

191                 warn_file = NULL;
192                 cmdline_config = NULL;
```



```
193         }
194         if (rc == 0 && (config_info.config_dn != NULL) &&
195             (config_info.config_dn[0] != '\0')) {
196                 rc = parse_ldap_config_dn_attrs(&proxyInfo,
197                     &ldapConfig, &ldapTableMapping, &config_info,
198                     &ldapDBTableMapping);
199         }

201         free_config_info(&config_info);

203         if (rc == 0 && got_config_data == FALSE)
204                 rc = 1;

206         set_default_values(&proxyInfo, &ldapConfig, &ldapDBTableMapping);

208         if (yp2ldap == 1 && rc == 0) {
209                 rc = second_parser_pass(&ldapTableMapping);
210                 if (rc == 0)
211                         rc = final_parser_pass(&ldapTableMapping, &ypDomains);
212                 if (rc == -2)
213                         return (-1);
214         }

216         if (rc == 0)
217                 rc = finish_parse(&proxyInfo, &ldapTableMapping);

219         if (rc == 0)
220                 rc = linked2hash(ldapTableMapping);

222         if ((rc == 0) && yptol_mode)
223                 rc = map_id_list_init();

225         if (rc != 0) {
226                 free_parse_structs();
227         } else if (verbose)
228                 report_info("LDAP configuration complete", NULL);
229         return (rc);
230 }
_____unchanged_portion_omitted_

406 static int
407 yp_parse_ldap_default_conf(
408         __nis_ldap_proxy_info *proxy_info,
409         __nis_config_t  *nis_config,
410         __nis_config_info_t *config_info,
411         __nisdb_table_mapping_t *table_info)
412 {
413         int rc = 0;
414         char            *ldap_config_attributes[n_config_keys];
415         char            attr_buf[128];
416         char            *attr;
417         char            *attr_val;
418         int             defflags;
419         config_key      attrib_num;
420         int             i, len;
415         int     i, len, attr_len;
421         void            *defp;

423         if ((defp = defopen_r(YP_ETCCONFFILE)) != NULL) {
424                 file_source = YP_ETCCONFFILE;
425                 if (verbose)
426                         report_info("default configuration values: ", NULL);
427                 /* Set defread_r() to be case insensitive */
428                 defflags = defcntl_r(DC_GETFLAGS, 0, defp);
429                 TURNOFF(defflags, DC_CASE);
430                 (void) defcntl_r(DC_SETFLAGS, defflags, defp);
```

```
                get_attribute_list(proxy_info, nis_config, config_info,
                    table_info, ldap_config_attributes);
                i = 0;
                while ((attr = ldap_config_attributes[i++]) != NULL) {
                        if ((strlcpy(attr_buf, attr, sizeof (attr_buf))) >=
                            sizeof (attr_buf)) {
                                report_error(
                                    "Static buffer attr_buf overflow", NULL);
                                defclose_r(defp);
                                return (-1);
                        }

                        if ((attr_val = defread_r(attr_buf, defp)) == NULL)
                                continue;

                        got_config_data = TRUE;
                        attrib_num = get_attrib_num(attr, strlen(attr));
                        if (attrib_num == key_bad) {
                                report_error(attr, NULL);
                                rc = -1;
                                break;
                        }

                        /*
                         * Allow either entries of the form
                         * attr val
                         * or
                         * attr = val
                         */
                        while (is_whitespace(*attr_val))
                                attr_val++;
                        if (*attr_val == '=')
                                attr_val++;
                        while (is_whitespace(*attr_val))
                                attr_val++;
                        len = strlen(attr_val);
                        while (len > 0 && is_whitespace(attr_val[len - 1]))
                                len--;

                        if (verbose) {
                                report_info("\t", attr);
                                report_info("\t\t", attr_val);
                        }
                        if (IS_YP_BIND_INFO(attrib_num)) {
                                rc = add_bind_attribute(attrib_num,
                                    attr_val, len, proxy_info);
                        } else if (IS_YP_OPER_INFO(attrib_num)) {
                                rc = add_operation_attribute(attrib_num,
                                    attr_val, len, nis_config,
                                    table_info);
                        }
                        if (p_error != no_parse_error) {
                                report_error(attr_val, attr);
                                rc = -1;
                                break;
                        }
                }
                file_source = NULL;
                /* Close the /etc/default file */
                defclose_r(defp);
        }
        return (rc);
}
```

*unchanged_portion_omitted_*


```
/*
 * FUNCTION:    yp_parse_ldap_config_file
 *
 * Parses the information for LDAP from a configuration
 * file. If no file is specified, /var/yp/NISLDAPmapping
 * is used
 *
 * RETURN VALUE:    0 on success, -1 on failure
 *
 * INPUT:       configuration file name
 */

int
yp_parse_ldap_config_file(
        const char      *config_file,
        __nis_ldap_proxy_info   *proxy_info,
        __nis_config_t                  *nis_config,
        __nis_table_mapping_t   **table_mapping,
        __nis_config_info_t             *config_info,
        __nisdb_table_mapping_t *table_info,
        __yp_domain_context_t   *ypDomains)
{
        int     rc = 0;
        int     numDomains = 0;
        config_key      attrib_num;
        int     fd;
        char    *attr_val = NULL;
        int             len;

        if ((fd = open(config_file, O_RDONLY)) == -1) {
                p_error = parse_open_file_error;
                report_error(config_file, NULL);
                return (-1);
        }

        start_line_num = 1;
        cur_line_num = 1;

        if (verbose)
                report_info("Reading configuration from ", config_file);

        file_source = config_file;
        while ((attrib_num = get_file_attr_val(fd, &attr_val)) > 0) {
                len = attr_val == NULL ? 0 : strlen(attr_val);
                if (IS_YP_CONFIG_KEYWORD(attrib_num)) {
                        rc = add_config_attribute(attrib_num,
                            attr_val, len, config_info);
                } else if (IS_YP_BIND_INFO(attrib_num)) {
                        rc = add_bind_attribute(attrib_num,
                            attr_val, len, proxy_info);
                } else if (IS_YP_OPER_INFO(attrib_num)) {
                        rc = add_operation_attribute(attrib_num,
                            attr_val, len, nis_config, table_info);
                } else if (IS_YP_DOMAIN_INFO(attrib_num)) {
                        rc = add_ypdomains_attribute(attrib_num,
                            attr_val, len, ypDomains);
                } else if (IS_YP_MAP_ATTR(attrib_num)) {
                        rc = add_mapping_attribute(attrib_num,
                            attr_val, len, table_mapping);
                } else {
                        rc = -1;
                        p_error = parse_unsupported_format;
                }

                if (rc < 0) {
                        report_error(attr_val == NULL ?
```

```
696                             "<no attribute>" : attr_val, _key_val);
697                         if (attr_val)
698                                 free(attr_val);
699                         break;
700                 }
701                 if (attr_val) {
702                         free(attr_val);
703                         attr_val = NULL;
704                 }
705         }

707         (void) close(fd);
708         if (attrib_num == key_bad) {
709                 report_error(_key_val, NULL);
710                 rc = -1;
711         }
712         start_line_num = 0;
713         file_source = NULL;
714         return (rc);
715 }

717 /*
718  * FUNCTION:    get_file_attr_val
719  *
720  *      Gets the next attribute from the configuration file.
721  *
722  * RETURN VALUE:        The config key if more attributes
723  *                      no_more_keys if eof
724  *                      key_bad if error
725  */

727 static config_key
728 get_file_attr_val(int fd, char **attr_val)
729 {
730         char            buf[BUFSIZE];
731         char            *start_tag;
732         char            *start_val;
733         char            *end_val;
734         char            *cut_here;
735         char            *s;
736         char            *a;
737         char            *attribute_value;
738         int             ret;
739         config_key      attrib_num = no_more_keys;
736         int             found_quote = 0;

741         *attr_val = NULL;

743         if ((ret = read_line(fd, buf, sizeof (buf))) > 0) {
744                 for (s = buf; is_whitespace(*s); s++)
745                         ;

747                 start_tag = s;
748                 while (*s != '\0' && !is_whitespace(*s))
749                         s++;

751                 if (verbose)
752                         report_info("\t", start_tag);
753                 attrib_num = get_attrib_num(start_tag, s - start_tag);
754                 if (attrib_num == key_bad)
755                         return (key_bad);

757                 while (is_whitespace(*s))
758                         s++;
759                 if (*s == '\0')
760                         return (attrib_num);
```



```
761                 start_val = s;

763                 /* note that read_line will not return a line ending with \ */
764                 for (; *s != '\0'; s++) {
765                         if (*s == ESCAPE_CHAR)
766                                 s++;
767                 }
768                 while (s > start_val && is_whitespace(s[-1]))
769                         s--;

771                 attribute_value =
772                     calloc(1, (size_t)(s - start_val) + 1);
773                 if (attribute_value == NULL) {
774                         p_error = parse_no_mem_error;
775                         return (key_bad);
776                 }
777                 attr_val[0] = attribute_value;

779                 a = *attr_val;
780                 end_val = s;
781                 cut_here = 0;
782                 for (s = start_val; s < end_val; s++) {
783                         if (*s == POUND_SIGN) {
784                                         cut_here = s;
785                                         while (s < end_val) {
786                                                 if (*s == DOUBLE_QUOTE_CHAR ||
787                                                     *s == SINGLE_QUOTE_CHAR) {
788                                                         cut_here = 0;
789                                                         break;
790                                                 }
791                                                 s++;
792                                         }
793                         }
794                 }
795                 if (cut_here != 0)
796                         end_val = cut_here;

798                 for (s = start_val; s < end_val; s++)
799                         *a++ = *s;
800                 *a++ = '\0';
801         }
802         if (ret == -1)
803                 return (key_bad);

805         return (attrib_num);
806 }

808 static LDAP *
809 connect_to_ldap_config_server(
810         char                    *sever_name,
811         int                     server_port,
812         __nis_config_info_t     *config_info)
813 {
811         int             rc              = 0;
814         LDAP            *ld             = NULL;
815         int             ldapVersion     = LDAP_VERSION3;
816         int             derefOption     = LDAP_DEREF_ALWAYS;
817         int             timelimit       = LDAP_NO_LIMIT;
818         int             sizelimit       = LDAP_NO_LIMIT;
819         int             errnum;
820         bool_t          retrying        = FALSE;
821         int             sleep_seconds   = 1;
822         struct berval   cred;

824         if (config_info->tls_method == no_tls) {
825                 ld = ldap_init(sever_name, server_port);
```

```
826                 if (ld == NULL) {
827                         p_error = parse_ldap_init_error;
828                         report_error(strerror(errno), NULL);
829                         return (NULL);
830                 }
831         } else {
832                 if ((errnum = ldapssl_client_init(
833                     config_info->tls_cert_db, NULL)) < 0) {
834                         p_error = parse_ldapssl_client_init_error;
835                         report_error(ldapssl_err2string(errnum), NULL);
836                         return (NULL);
837                 }
838                 ld = ldapssl_init(sever_name, server_port, 1);
839                 if (ld == NULL) {
840                         p_error = parse_ldapssl_init_error;
841                         report_error(strerror(errno), NULL);
842                         return (NULL);
843                 }
844         }

846         (void) ldap_set_option(ld, LDAP_OPT_PROTOCOL_VERSION,
847             &ldapVersion);
848         (void) ldap_set_option(ld, LDAP_OPT_DEREF, &derefOption);
849         (void) ldap_set_option(ld, LDAP_OPT_REFERRALS, LDAP_OPT_OFF);
850         (void) ldap_set_option(ld, LDAP_OPT_TIMELIMIT, &timelimit);
851         (void) ldap_set_option(ld, LDAP_OPT_SIZELIMIT, &sizelimit);

853         /*
854          * Attempt to bind to the LDAP server.
855          * We will loop until success or until an error other
856          * than LDAP_CONNECT_ERROR or LDAP_SERVER_DOWN
857          */
858         if (verbose)
859                 report_info("Connecting to ", sever_name);

861         for (;;) {
862                 if (config_info->auth_method == simple) {
863                         errnum = ldap_simple_bind_s(ld, config_info->proxy_dn,
864                             config_info->proxy_passwd);
865                 } else if (config_info->auth_method == cram_md5) {
866                         cred.bv_len = strlen(config_info->proxy_passwd);
867                         cred.bv_val = config_info->proxy_passwd;
868                         errnum = ldap_sasl_cram_md5_bind_s(ld,
869                             config_info->proxy_dn, &cred, NULL, NULL);
870                 } else if (config_info->auth_method == digest_md5) {
871                         cred.bv_len = strlen(config_info->proxy_passwd);
872                         cred.bv_val = config_info->proxy_passwd;
873                         errnum = ldap_x_sasl_digest_md5_bind_s(ld,
874                             config_info->proxy_dn, &cred, NULL, NULL);
875                 } else {
876                         errnum = ldap_simple_bind_s(ld, NULL, NULL);
877                 }

879                 if (errnum == LDAP_SUCCESS)
880                         break;

882                 if (errnum == LDAP_CONNECT_ERROR ||
883                     errnum == LDAP_SERVER_DOWN) {
884                         if (!retrying) {
885                                 if (verbose)
886                                         report_info(
887                                         "LDAP server unavailable. Retrying...",
888                                             NULL);
889                                 retrying = TRUE;
890                         }
891                         (void) sleep(sleep_seconds);
```



```
892                         sleep_seconds *= 2;
893                         if (sleep_seconds > MAX_LDAP_CONFIG_RETRY_TIME)
894                                 sleep_seconds = MAX_LDAP_CONFIG_RETRY_TIME;
895                         p_error = no_parse_error;
896                         continue;
897                 }
898                 p_error = parse_ldap_bind_error;
899                 report_error2(config_info->proxy_dn, ldap_err2string(errnum));
900                 (void) ldap_unbind(ld);
901                 return (NULL);
902         }

904         if (verbose)
905                 report_info("Reading values from ", config_info->config_dn);

907         return (ld);
908 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   64602 Fri Jul 24 12:28:12 2015
new/usr/src/lib/libnisdb/nis_parse_ldap_map.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2001-2003 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>
#include <locale.h>

#include "ldap_parse.h"
#include "nis_parse_ldap_conf.h"
#include "nis_parse_ldap_yp_util.h"
#include "nis_parse_ldap_util.h"

/* other attribute functions */
static char *getIndex(const char **s_cur, const char *end_s);
static bool_t get_ttls(const char *s, const char *s_end,
    __nis_table_mapping_t *t_mapping);
static __nis_object_dn_t *parse_object_dn(const char *s, const char *end);
static int      parse_name_fields(const char *name_s, const char *name_s_end,
         __nis_table_mapping_t *t_mapping);
static void get_mapping_rule(const char *s, int len,
    __nis_table_mapping_t *tbl, bool_t to_ldap);
static bool_t get_deleteDisp(const char *s_begin, const char *s_end,
    __nis_object_dn_t *obj_dn);

/* mapping rule functions */
static const char *get_lhs(const char *s, const char *end_s,
    __nis_mapping_rlhs_t *lhs, __nis_mapping_item_type_t item_type);
static const char *get_lhs_match(const char *s, const char *end_s,
    __nis_mapping_rlhs_t *lhs, __nis_mapping_item_type_t item_type);
static const char *get_lhs_paren_item(const char *s, const char *end_s,
    __nis_mapping_rlhs_t *lhs, __nis_mapping_item_type_t item_type);
```



```
static const char *get_rhs(const char *s, const char *end_s,
    __nis_mapping_rlhs_t *lhs, __nis_mapping_item_type_t item_type);
static const char *get_mapping_item(const char *s, const char *end_s,
    __nis_mapping_item_t *item, __nis_mapping_item_type_t type);
static const char *get_print_mapping_element(const char *s,
    const char *end_s, char *fmt_string, __nis_mapping_element_t *e,
    __nis_mapping_item_type_t item_type);
static const char *get_subElement(const char *s, const char *end_s,
    __nis_mapping_sub_element_t *subelement,
    __nis_mapping_item_type_t type);
static bool_t get_mapping_format(const char *fmt_string,
    __nis_mapping_format_t **fmt, int *nfmt, int *numItems,
    bool_t print_mapping);
extern __yp_domain_context_t ypDomains;

/*
 * FUNCTION:    add_mapping_attribute
 *
 *      Adds the attribute value to __nis_table_mapping_t
 *      if the value is not yet set for the given database.
 *
 * RETURN VALUE:        0 on success, -1 on failure
 *
 * INPUT:               attribute number and value
 */

int
add_mapping_attribute(
        config_key              attrib_num,
        const char              *attrib_val,
        int                     attrib_len,
        __nis_table_mapping_t   **table_mapping)
{
        const char              *s;
        const char              *attrib_end;
        const char              *db_id_end;
        const char              *begin_token;
        const char              *end_token;
        char                    *index_string;
        __nis_object_dn_t       *objectDN;
        __nis_table_mapping_t   *t_mapping;
        __nis_table_mapping_t   *t;

        bool_t                  new_mapping     = FALSE;
        int                             nm;
        char                    *tmp_dbId;

        attrib_end = attrib_val + attrib_len;
        for (s = attrib_val; s < attrib_end; s++)
                if (*s == COLON_CHAR)
                        break;

        if (s == attrib_end || *attrib_val == COLON_CHAR) {
                p_error = parse_unexpected_data_end_rule;
                return (-1);
        }

        db_id_end = s;
        while (s > attrib_val && is_whitespace(s[-1]))
                s--;

        if (s == attrib_val) {
                p_error = parse_unexpected_data_end_rule;
                return (-1);
        }
```

```
125         if (yp2ldap) {
126                 tmp_dbId = s_strndup(attrib_val, s - attrib_val);
127                 if (tmp_dbId == NULL) {
128                         p_error = parse_no_mem_error;
129                         return (-1);
130                 }
131                 if (strchr(tmp_dbId, COMMA_CHAR)) {
132                         /* domain explicitly specified */
133                         nm = check_domain_specific_order(tmp_dbId,
134                                 attrib_num, *table_mapping, &ypDomains);
135                         /*
136                          * No logging is needed here, as
137                          * check_domain_specific_order
138                          * will log any appropriate errors.
139                          */
140                         if (nm != 0) {
141                                 free(tmp_dbId);
142                                 return (-1);
143                         }
144                 }
145                 free(tmp_dbId);
146         }
147
148         if ((t_mapping = find_table_mapping(attrib_val,
149                         s - attrib_val, *table_mapping)) == NULL) {
150                 /* No mapping with this id, create one */
151                 t_mapping = (__nis_table_mapping_t *)
152                         s_calloc(1, sizeof (__nis_table_mapping_t));
153
154                 if (t_mapping == NULL) {
155                         p_error = parse_no_mem_error;
156                         return (-1);
157                 }
158                 (void) initialize_table_mapping(t_mapping);
159
160                 /* dbId is the label before the colon */
161                 t_mapping->dbId = s_strndup(attrib_val, s - attrib_val);
162                 if (t_mapping->dbId == NULL) {
163                         p_error = parse_no_mem_error;
164                         free(t_mapping);
165                         return (-1);
166                 }
167                 new_mapping = TRUE;
168         } else {
169                 /* a table mapping already exists, use it */
170                 new_mapping = FALSE;
171         }
172
173         s = db_id_end + 1;
174         while (s < attrib_end && is_whitespace(*s))
175                 s++;
176
177         switch (attrib_num) {
178                 case key_yp_map_flags:
179                         if (t_mapping->usedns_flag != 0 ||
180                                 t_mapping->securemap_flag != 0) {
181                                 warn_duplicate_map(t_mapping->dbId,
182                                         attrib_num);
183                                 break;
184                         }
185                         while (is_whitespace(*s) && s < attrib_end)
186                                 s++;
187                         while (s < attrib_end) {
188                                 if (s < attrib_end && *s == 'b')
189                                         t_mapping->usedns_flag = 1;
190                                 if (s < attrib_end && *s == 's')
```



```
191                                         t_mapping->securemap_flag = 1;
192                                 s++;
193                         }
194                         break;
195                 case key_yp_comment_char:
196                         if (t_mapping->commentChar !=
197                                 DEFAULT_COMMENT_CHAR) {
198                                 warn_duplicate_map(t_mapping->dbId, attrib_num);
199                                 break;
200                         }
201                         while (is_whitespace(*s) && s < attrib_end)
202                                 s++;
203                         if (s < attrib_end && (s+1) < attrib_end &&
204                                 (s+2) <= attrib_end) {
205                                 while (is_whitespace(attrib_end[-1]))
206                                         attrib_end--;
207                                 while (*s != SINGLE_QUOTE_CHAR)
208                                         s++;
209                                 if (*s == SINGLE_QUOTE_CHAR &&
210                                         *(s+2) == SINGLE_QUOTE_CHAR) {
211                                         t_mapping->commentChar = *(s+1);
212                                 } else if (*s == SINGLE_QUOTE_CHAR &&
213                                         *(s+1) == SINGLE_QUOTE_CHAR) {
214                                         t_mapping->commentChar = NULL;
215                                 } else {
216                                         /* anything else is an error */
217                                         p_error = parse_bad_yp_comment_error;
218                                 }
219                                 break;
220                         } else {
221                                 p_error = parse_bad_yp_comment_error;
222                                 break;
223                         }
224                 case key_yp_repeated_field_separators:
225                         while (s < attrib_end && is_whitespace(*s))
226                                 s++;
227                         if (s < attrib_end) {
228                                 while (is_whitespace(attrib_end[-1]))
229                                         attrib_end--;
230                                 while (s < attrib_end &&
231                                                 *s != DOUBLE_QUOTE_CHAR)
232                                         s++;
233                                 s++;
234                                 begin_token = s;
235                                 while (s < attrib_end &&
236                                                 *s != DOUBLE_QUOTE_CHAR) {
237                                         if (*s == ESCAPE_CHAR)
238                                                 s++;
239                                         s++;
240                                 }
241                                 t_mapping->separatorStr =
242                                         s_strndup(begin_token, s - begin_token);
243                                 if (t_mapping->separatorStr == NULL)
244                                         break;
245                         } else {
246                                 p_error = parse_bad_field_separator_error;
247                         }
248                         break;
249                 case key_yp_name_fields:
250                 case key_yp_split_field:
251                         if (t_mapping->e || t_mapping->numSplits > 0) {
252                                 warn_duplicate_map(t_mapping->dbId,
253                                         attrib_num);
254                                 break;
255                         }
256                         if (parse_name_fields(s, attrib_end, t_mapping)) {
```

```
257                             p_error = parse_bad_name_field;
258                     }
259                     break;
260             case key_yp_db_id_map:
261             case key_db_id_map:
262                     if (t_mapping->objName != NULL) {
263                             warn_duplicate_map(t_mapping->dbId, attrib_num);
264                             break;
265                     }

267                     if (s < attrib_end && *s == OPEN_BRACKET) {
268                             index_string = getIndex(&s, attrib_end);
269                             if (index_string == NULL)
270                                     break;
271                             (void) parse_index(index_string,
272                                     index_string + strlen(index_string),
273                                     &t_mapping->index);
274                             free(index_string);
275                             if (p_error != no_parse_error)
276                                     break;
277                     }
278                     while (is_whitespace(*s) && s < attrib_end)
279                             s++;
280                     if (s < attrib_end) {
281                             while (is_whitespace(attrib_end[-1]))
282                                     attrib_end--;
283                             t_mapping->objName =
284                                     s_strndup_esc(s, attrib_end - s);
285                     } else {
286                             if (yp2ldap) {
287                                     p_error = parse_bad_map_error;
288                             } else {
289                                     t_mapping->objName = s_strndup(s, 0);
290                             }
291                     }
292                     break;

294             case key_yp_entry_ttl:
295             case key_entry_ttl:
296                     if (t_mapping->initTtlLo != (time_t)NO_VALUE_SET) {
297                             warn_duplicate_map(t_mapping->dbId, attrib_num);
298                             break;
299                     }

301                     if (!get_ttls(s, attrib_end, t_mapping))
302                             p_error = parse_bad_ttl_format_error;
303                     break;

305             case key_yp_ldap_object_dn:
306             case key_ldap_object_dn:
307                     if (t_mapping->objectDN != NULL) {
308                             warn_duplicate_map(t_mapping->dbId, attrib_num);
309                             break;
310                     }
311                     objectDN = parse_object_dn(s, attrib_end);
312                     if (objectDN == NULL)
313                             break;
314                     t_mapping->objectDN = objectDN;
315                     t_mapping->seq_num = seq_num++;
316                     break;

318             case key_nis_to_ldap_map:
319             case key_nisplus_to_ldap_map:
320                     if (t_mapping->ruleToLDAP != 0) {
321                             warn_duplicate_map(t_mapping->dbId, attrib_num);
322                             break;
```



```
323                     }

325                     get_mapping_rule(s, attrib_end - s, t_mapping, TRUE);
326                     break;

328             case key_ldap_to_nis_map:
329             case key_ldap_to_nisplus_map:
330                     if (t_mapping->ruleFromLDAP != NULL) {
331                             warn_duplicate_map(t_mapping->dbId, attrib_num);
332                             break;
333                     }

335                     get_mapping_rule(s, attrib_end - s, t_mapping, FALSE);
336                     break;

338             default:
339                     p_error = parse_internal_error;
340                     break;
341     }
342     if (p_error == no_parse_error) {
343             if (new_mapping) {
344                     if (*table_mapping == NULL)
345                             *table_mapping = t_mapping;
346                     else {
347                             for (t = *table_mapping; t->next != NULL;
348                                 t = t->next)
349                                     ;
350                             t->next = t_mapping;
351                     }
352             }
353     } else {
354             if (new_mapping)
355                     free_table_mapping(t_mapping);
356     }
357     return (p_error == no_parse_error ? 0 : -1);
358 }

360 /*
361  * FUNCTION:    add_ypdomains_attribute
362  *
363  * Adds the yp domains information to the __yp_domain_context_t
364  * structure.
365  *
366  * RETURN:              0 on success, -1 on failure
367  *
368  * INPUT:               attribute number and value
369  */

371 int
372 add_ypdomains_attribute(
373         config_key              attrib_num,
374         const char              *attrib_val,
375         int                             attrib_len,
376         __yp_domain_context_t   *ypDomains)
377 {
378         const char              *s;
379         const char              *attrib_end;
380         int                             numDomains = 0;
381         int                             i;
382         char                    *tmp_str;
383         int                             ret = 0;

382         attrib_end = attrib_val + attrib_len;
383         for (s = attrib_val; s < attrib_end; s++) {
384                 if (*s == COLON_CHAR) {
385                         break;
```

```
386                 }
387         }
388         while (s > attrib_val && is_whitespace(s[-1]))
389                 s--;

391         if (s == attrib_val) {
392                 p_error = parse_unexpected_data_end_rule;
393                 return (-1);
394         }

396         if (ypDomains == NULL) {
397                 /*
398                  * No point allocating. We cant return the resulting structure,
399                  * so just return failure. Should not ever happen because we
400                  * are always called with a pointer to the global ypDomains
401                  * structure.
402                  */
403                 return (-1);
404         }

406         switch (attrib_num) {
407                 case key_yp_domain_context:
408                         numDomains = ypDomains->numDomains;
409                         ypDomains->domainLabels =
410                                 (char **)s_realloc(ypDomains->domainLabels,
411                                 (numDomains + 1) *
412                                 sizeof (ypDomains->domainLabels[0]));
413                         if (ypDomains->domainLabels == NULL) {
414                                 p_error = parse_no_mem_error;
415                                 free_yp_domain_context(ypDomains);
416                                 break;
417                         }
418                         ypDomains->domainLabels[numDomains] =
419                                 s_strndup(attrib_val, s - attrib_val);
420                         if (ypDomains->domainLabels[numDomains] == NULL) {
421                                 p_error = parse_no_mem_error;
422                                 free_yp_domain_context(ypDomains);
423                                 break;
424                         }
425                         ypDomains->numDomains = numDomains + 1;
426                         while (s < attrib_end && is_whitespace(*s))
427                                 s++;
428                         if (*s == COLON_CHAR)
429                                 s++;
430                         while (s < attrib_end && is_whitespace(*s))
431                                 s++;
432                         ypDomains->domains =
433                                 (char **)s_realloc(ypDomains->domains,
434                                 (numDomains + 1) *
435                                 sizeof (ypDomains->domains[0]));
436                         if (ypDomains->domains == NULL) {
437                                 p_error = parse_no_mem_error;
438                                 free_yp_domain_context(ypDomains);
439                                 break;
440                         }

442                         if (s < attrib_end) {
443                                 while (is_whitespace(attrib_end[-1]))
444                                         attrib_end--;
445                                 ypDomains->domains[numDomains] =
446                                         s_strndup_esc(s, attrib_end - s);
447                                 if (ypDomains->domains[numDomains] == NULL) {
448                                         p_error = parse_no_mem_error;
449                                         free_yp_domain_context(ypDomains);
450                                         break;
451                                 }
```



```
452                         } else {
453                                 p_error = parse_unexpected_yp_domain_end_error;
454                                 free(ypDomains->domainLabels[numDomains]);
455                                 ypDomains->domainLabels[numDomains] = NULL;
456                                 ypDomains->numDomains--;
457                                 free_yp_domain_context(ypDomains);
458                         }
459                         break;
460                 case key_yppasswdd_domains:
461                         ypDomains->yppasswddDomainLabels =
462                                 (char **)s_realloc(
463                                 ypDomains->yppasswddDomainLabels,
464                                 (ypDomains->numYppasswdd + 1) *
465                                 sizeof (ypDomains->yppasswddDomainLabels[0]));
466                         if (ypDomains->yppasswddDomainLabels == NULL) {
467                                 p_error = parse_no_mem_error;
468                                 break;
469                         }
470                         ypDomains->yppasswddDomainLabels
471                                 [ypDomains->numYppasswdd] =
472                                 s_strndup(attrib_val, s - attrib_val);
473                         if (ypDomains->yppasswddDomainLabels
474                                 [ypDomains->numYppasswdd] == NULL) {
475                                 p_error = parse_no_mem_error;
476                         }
477                         ypDomains->numYppasswdd++;
478                         break;
479         }

481         return (p_error == no_parse_error ? 0 : -1);
482 }
```

*_____unchanged_portion_omitted_*

```
1417 /*
1418  * FUNCTION:    get_rhs
1419  *
1420  *      Parse right hand side of mapping rule attribute
1421  *
1422  * RETURN VALUE:        NULL if error
1423  *                      position of beginning next mapping rule
1424  *
1425  * INPUT:               the attribute value and mapping rule type
1426  */

1428 static const char *
1429 get_rhs(
1430         const char                      *s,
1431         const char                      *end_s,
1432         __nis_mapping_rlhs_t            *rhs,
1433         __nis_mapping_item_type_t       item_type)
1434 {
1435         /*
1436          * This handles the following cases:
1437          *      name                            me_item
1438          *      (name)                          me_item
1439          *      (fmt, name-list)                me_print
1440          *      (item, fmt)                     me_extract
1441          */

1443         token_type              t;
1444         const char              *begin_token;
1445         const char              *end_token;
1446         char                    *str            = NULL;
1447         __nis_mapping_format_t  *fmt            = NULL;
1448         __nis_mapping_element_t *e              = NULL;
1449         __nis_mapping_item_t    item;
```

```
1450         int                     n;

1452         (void) memset(&item, 0, sizeof (item));

1454         for (; p_error == no_parse_error; ) {
1455                 begin_token = s;
1456                 end_token = end_s;
1457                 s = get_next_token(&begin_token, &end_token, &t);
1458                 if (s == NULL)
1459                         break;

1461                 e = (__nis_mapping_element_t *)
1462                         s_calloc(1, sizeof (__nis_mapping_element_t));
1463                 if (e == NULL)
1464                         break;

1466                 if (t == string_token) {
1467                         s = get_mapping_item(begin_token, end_s,
1468                                 &e->element.item, item_type);
1469                 } else if (t == open_paren_token) {
1470                         begin_token = s;
1471                         end_token = end_s;
1472                         s = get_next_token(&begin_token, &end_token, &t);
1473                         if (s == NULL)
1474                                 break;
1475                         if (t == string_token) {
1476                                 /* (item, fmt) - me_extract */
1477                                 /* (item, "c") - me_split */
1478                                 s = get_mapping_item(begin_token, end_s,
1479                                     &item, item_type);
1480                                 if (s == NULL)
1481                                     break;
1482                                 begin_token = s;
1483                                 end_token = end_s;
1484                                 s = get_next_token(&begin_token, &end_token,
1485                                     &t);
1486                                 if (s == NULL)
1487                                     break;
1488                                 else if (t == close_paren_token) {
1489                                     item.repeat = TRUE;
1490                                     e->element.item = item;
1491                                     e->type = me_item;
1492                                     rhs->numElements = 1;
1493                                     rhs->element = e;
1494                                     return (s);
1495                                 } else if (t != comma_token) {
1496                                     p_error = parse_comma_expected_error;
1497                                     break;
1498                                 }

1500                                 begin_token = s;
1501                                 end_token = end_s;
1502                                 s = get_next_token(&begin_token, &end_token,
1503                                     &t);
1504                                 if (s == NULL || t != quoted_string_token) {
1505                                     p_error =
1506                                         parse_format_string_expected_error;
1507                                     break;
1508                                 }

1510                                 if (end_token == begin_token + 1 ||
1511                                     (*begin_token == ESCAPE_CHAR &&
1512                                     end_token == begin_token + 2)) {
1514                                     *begin_token == ESCAPE_CHAR &&
1515                                     end_token == begin_token + 2) {
1513                                         e->type = me_split;
```



```
1514                                         e->element.split.item = item;
1515                                         e->element.split.delim = *begin_token;
1516                                 } else {
1517                                         str = s_strndup(begin_token,
1518                                                 end_token - begin_token);
1519                                         if (str == NULL)
1520                                                 break;
1521                                         if (!get_mapping_format(str, &fmt,
1522                                             NULL, &n, FALSE))
1523                                                 break;
1524                                         free(str);
1525                                         str = NULL;
1526                                         if (n != 1) {
1527                                             p_error =
1528                                                 parse_bad_extract_format_spec;
1529                                             break;
1530                                         }
1531                                         e->type = me_extract;
1532                                         e->element.extract.item = item;
1533                                         e->element.extract.fmt = fmt;
1534                                 }
1535                                 s = skip_token(s, end_s, close_paren_token);
1536                         } else if (t == quoted_string_token) {
1537                                 /* (fmt, name-list) - me_print */
1538                                 str = s_strndup(begin_token,
1539                                         end_token - begin_token);
1540                                 if (str == NULL)
1541                                         break;

1543                                 s = get_print_mapping_element(s, end_s,
1544                                         str, e, item_type);
1545                                 free(str);
1546                                 str = NULL;
1547                         } else {
1548                                 p_error = parse_start_rhs_unrecognized;
1549                                 break;
1550                         }
1551                 } else {
1552                         p_error = parse_start_rhs_unrecognized;
1553                         break;
1554                 }
1555                 if (s == NULL)
1556                         break;
1557                 rhs->numElements = 1;
1558                 rhs->element = e;
1559                 if (p_error == no_parse_error)
1560                         return (s);
1561         }
1562         if (str)
1563                 free(str);
1564         if (fmt != NULL)
1565                 free_mapping_format(fmt);
1566         if (e != NULL)
1567                 free_mapping_element(e);
1568         free_mapping_item(&item);

1570         return (NULL);
1571 }

1573 /*
1574  * FUNCTION:    get_print_mapping_element
1575  *
1576  *      Parse a print mapping rule attribute in case of the form
1577  *      (fmt, name-list)
1578  *
1579  * RETURN VALUE:        NULL if error
```

```
 *                      position of beginning next mapping rule
 *
 * INPUT:               the attribute value and mapping rule type
 */

static const char *
get_print_mapping_element(
        const char                      *s,
        const char                      *end_s,
        char                            *fmt_string,
        __nis_mapping_element_t         *e,
        __nis_mapping_item_type_t       item_type)
{
        token_type                      t;
        const char                      *begin_token;
        const char                      *end_token;
        char                            elide;
        bool_t                          doElide;
        __nis_mapping_format_t          *base           = NULL;
        __nis_mapping_sub_element_t     *subElement     = NULL;
        int                             n               = 0;
        int                             nSub            = 0;
        int                             numSubElements;

        for (; p_error == no_parse_error; ) {
                if (!get_mapping_format(fmt_string, &base, &n,
                    &numSubElements, TRUE))
                        break;
                subElement = (__nis_mapping_sub_element_t *)
                        s_calloc(numSubElements,
                        sizeof (__nis_mapping_sub_element_t));
                if (subElement == NULL)
                        break;
                for (n = 0; base[n].type != mmt_end; n++) {
                        if (base[n].type != mmt_item &&
                                base[n].type != mmt_berstring) {
                            if (base[n].type == mmt_berstring_null)
                                base[n].type = mmt_berstring;
                            continue;
                        }
                        if (nSub < numSubElements) {
                                s = skip_token(s, end_s, comma_token);
                                if (s == NULL) {
                                        p_error = parse_bad_print_format;
                                        break;
                                }
                        }

                        /* namelist may have parens around it */
                        s = get_subElement(s, end_s, &subElement[nSub],
                                item_type);
                        if (s == NULL)
                                break;
                        nSub++;
                }
                if (p_error != no_parse_error)
                        break;

                begin_token = s;
                end_token = end_s;
                s = get_next_token(&begin_token, &end_token, &t);
                if (s == NULL || t == no_token) {
                        p_error = parse_unexpected_data_end_rule;
                        break;
                } else if (t == close_paren_token) {
                        doElide = FALSE;
```



```
                        elide = '\0';
                } else if (t == comma_token) {
                        begin_token = s;
                        end_token = end_s;
                        s = get_next_token(&begin_token, &end_token, &t);
                        if (s != NULL && t == quoted_string_token &&
                            (end_token == begin_token + 1 ||
                            (*begin_token == ESCAPE_CHAR &&
                            end_token == begin_token + 2))) {
                                *begin_token == ESCAPE_CHAR &&
                                end_token == begin_token + 2)) {
                                if (numSubElements != 1 ||
                                    subElement->type == me_extract ||
                                    subElement->type == me_split) {
                                        p_error = parse_cannot_elide;
                                        break;
                                }
                                if (subElement->type == me_item &&
                                    !subElement->element.item.repeat) {
                                        p_error = parse_cannot_elide;
                                        break;
                                }
                                elide = *begin_token;
                                doElide = TRUE;

                        } else {
                                p_error = parse_bad_elide_char;
                                break;
                        }
                        s = skip_token(s, end_s, close_paren_token);
                        if (s == NULL)
                                break;
                }

                e->type = me_print;
                e->element.print.fmt = base;
                e->element.print.numSubElements = numSubElements;
                e->element.print.subElement = subElement;
                e->element.print.elide = elide;
                e->element.print.doElide = doElide;

                if (p_error == no_parse_error)
                        return (s);
        }
        if (base)
                free_mapping_format(base);
        if (subElement != NULL) {
                for (n = 0; n < numSubElements; n++)
                        free_mapping_sub_element(&subElement[n]);
                free(subElement);
        }

        return (NULL);
}
```

_____unchanged_portion_omitted_

```
/*
 * FUNCTION:    get_subElement
 *
 *      Parse attribute string to get sub element item
 *
 * RETURN VALUE:        NULL if error
 *                      position of beginning next token after item
 *
 * INPUT:               the attribute value and mapping rule type
 */
```

```
1937 static const char *
1938 get_subElement(
1939         const char                      *s,
1940         const char                      *end_s,
1941         __nis_mapping_sub_element_t     *subelement,
1942         __nis_mapping_item_type_t       type)
1943 {
1944         token_type                      t;
1945         const char                      *begin_token;
1946         const char                      *end_token;
1947         char                            *fmt_string;
1948         __nis_mapping_item_t            item;
1949         __nis_mapping_element_type_t    e_type;
1950         __nis_mapping_item_t            *print_item     = NULL;
1951         __nis_mapping_format_t          *base           = NULL;
1952         int                             n               = 0;
1953         int                             numItems        = 0;
1954         unsigned char                   delim;
1955         __nis_mapping_sub_element_t     sub;

1957 /*
1958  *      What is the form of we are expecting here
1959  *      item                                    me_item
1960  *      (item)                                  me_item
1961  *      ("fmt", item1, item2, ..., item n)      me_print
1962  *      ("fmt", (item), "elide")                me_print
1963  *      (name, "delim")                         me_split
1964  *      (item, "fmt")                           me_extract
1965  */
1966         (void) memset(&item, 0, sizeof (item));

1968         for (; p_error == no_parse_error; ) {
1969                 begin_token = s;
1970                 end_token = end_s;
1971                 s = get_next_token(&begin_token, &end_token, &t);
1972                 if (s == NULL)
1973                         break;
1974                 if (t == string_token) {        /* me_item */
1975                         s = get_mapping_item(begin_token, end_s,
1976                                 &subelement->element.item, type);
1977                         if (s == NULL)
1978                                 break;
1979                         subelement->type = me_item;
1980                         return (s);
1981                 } else if (t != open_paren_token) {
1982                         p_error = parse_item_expected_error;
1983                         break;
1984                 }

1986                 begin_token = s;
1987                 end_token = end_s;
1988                 s = get_next_token(&begin_token, &end_token, &t);
1989                 if (s == NULL)
1990                         break;

1992                 if (t != string_token && t != quoted_string_token) {
1993                         p_error = parse_item_expected_error;
1994                         break;
1995                 }
1996                 e_type = me_print;
1997                 if (t == string_token) {
1998                         /* me_item, me_extract or me_split */
1999                         s = get_mapping_item(begin_token, end_s, &item, type);
2000                         if (s == NULL)
2001                                 break;
```



```
2003                         begin_token = s;
2004                         end_token = end_s;
2005                         s = get_next_token(&begin_token, &end_token, &t);
2006                         if (s == NULL) {
2007                                 p_error = parse_unexpected_data_end_rule;
2008                                 break;
2009                         } else if (t == close_paren_token) {
2010                                 subelement->type = me_item;
2011                                 item.repeat = TRUE;
2012                                 subelement->element.item = item;
2013                                 if (yp2ldap) {
2014                                         while (s < end_s && is_whitespace(*s))
2015                                                 s++;
2016                                         if (s == end_s) {
2017                                                 p_error =
2018                                                 parse_unexpected_data_end_rule;
2019                                                 break;
2020                                         }
2021                                         if (*s == DASH_CHAR && s < end_s) {
2022                                                 s++;
2023                                                 while (s < end_s &&
2024                                                         is_whitespace(*s))
2025                                                         s++;
2026                                                 begin_token = s;
2027                                                 end_token = end_s;

2029                                                 subelement->element.item.exItem
2030                                                         =
2031                                                         (__nis_mapping_item_t *)
2032                                         s_malloc(sizeof (__nis_mapping_item_t));
2033                                                 if (!subelement->
2034                                                 element.item.exItem)
2035                                                         break;
2036                                                 s = get_mapping_item(s, end_s,
2037                                                         subelement->
2038                                                         element.item.exItem,
2039                                                         type);
2040                                                 if (s == NULL) {
2041                                                         p_error =
2042                                                         parse_internal_error;
2043                                                         free_mapping_item(
2044                                                         subelement->
2045                                                         element.item.exItem);
2046                                                         subelement->
2047                                                         element.item.exItem =
2048                                                                 NULL;
2049                                                         break;
2050                                                 }
2051                                         }
2052                                 }
2053                                 return (s);
2054                         } else if (t != comma_token) {
2055                                 p_error = parse_comma_expected_error;
2056                                 break;
2057                         }

2059                         begin_token = s;
2060                         end_token = end_s;
2061                         s = get_next_token(&begin_token, &end_token, &t);
2062                         if (s == NULL || t != quoted_string_token) {
2063                                 p_error = parse_format_string_expected_error;
2064                                 break;
2065                         }
2066                         if (end_token == begin_token + 1 ||
2067                             (*begin_token == ESCAPE_CHAR &&
```

```
2068                            end_token == begin_token + 2)) {
2070                            *begin_token == ESCAPE_CHAR &&
2071                            end_token == begin_token + 2) {
2069                                    /* me_split */
2070                                delim = (unsigned char)end_token[-1];
2071                                s = skip_token(s, end_s, close_paren_token);
2072                                if (s == NULL)
2073                                        break;
2074                                subelement->element.split.item = item;
2075                                subelement->element.split.delim = delim;
2076                                subelement->type = me_split;
2077                                return (s);
2078                        }
2079                        e_type = me_extract;
2080                }
2081                fmt_string = s_strndup(begin_token, end_token - begin_token);
2082                if (fmt_string == NULL)
2083                        break;
2084                if (!get_mapping_format(fmt_string, &base, &n, &numItems,
2085                    e_type == me_print)) {
2086                        free(fmt_string);
2087                        break;
2088                }
2089                free(fmt_string);

2091                if (numItems != 1 && e_type == me_extract) {
2092                        p_error = numItems == 0 ?
2093                                parse_not_enough_extract_items :
2094                                parse_too_many_extract_items;
2095                        break;
2096                } else if (numItems > 0 && e_type == me_print) {
2097                        print_item = (__nis_mapping_item_t *)s_calloc(numItems,
2098                                sizeof (__nis_mapping_item_t));
2099                        if (print_item == NULL)
2100                                break;
2101                }

2103                if (e_type == me_print) {
2104                        sub.element.print.numItems = numItems;
2105                        sub.element.print.fmt = base;
2106                        sub.element.print.item = print_item;
2107                        s = get_print_sub_element(s, end_s, type, &sub);
2108                        if (s == NULL)
2109                                break;
2110                }
2111                s = skip_token(s, end_s, close_paren_token);
2112                if (s == NULL)
2113                        break;

2115                subelement->type = e_type;
2116                if (e_type == me_extract) {
2117                        subelement->element.extract.fmt = base;
2118                        subelement->element.extract.item = item;
2119                } else {
2120                        subelement->type = me_print;
2121                        subelement->element.print.fmt = base;
2122                        subelement->element.print.numItems = numItems;
2123                        subelement->element.print.item = print_item;
2124                        subelement->element.print.doElide =
2125                                sub.element.print.doElide;
2126                        subelement->element.print.elide =
2127                                sub.element.print.elide;
2128                }
2129                if (p_error == no_parse_error)
2130                        return (s);
2131        }
```



```
2132        free_mapping_item(&item);
2133        if (base != NULL)
2134                free_mapping_format(base);
2135        if (print_item) {
2136                for (n = 0; n < numItems; n++)
2137                        free_mapping_item(&print_item[n]);
2138                free(print_item);
2139        }

2141        return (NULL);
2142 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   61765 Fri Jul 24 12:28:12 2015
new/usr/src/lib/libnisdb/nis_parse_ldap_util.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <locale.h>
#include <lber.h>
#include <ldap.h>
#include <syslog.h>
#include <dlfcn.h>      /* for dynamic loading only */

#include "ldap_parse.h"
#include "nis_parse_ldap_conf.h"
#include "nis_parse_ldap_err.h"
#include "ldap_util.h"
#include "ldap_util.h"

void append_dot(char **str);
void    append_comma(char **str);
bool_t make_full_dn(char **dn, const char *base);
bool_t make_fqdn(__nis_object_dn_t *dn, const char *base);
char *get_default_ldap_base(const char *domain);
bool_t add_domain(char **objName, const char *domain);
bool_t add_column(__nis_table_mapping_t *t, const char *col_name);
__nis_mapping_rule_t **dup_mapping_rules(
        __nis_mapping_rule_t **rules, int n_rules);
__nis_mapping_rule_t *dup_mapping_rule(
        __nis_mapping_rule_t *in);
void *s_malloc(size_t size);
__nis_mapping_format_t *dup_format_mapping(
```



```
        __nis_mapping_format_t *in);
bool_t dup_mapping_element(__nis_mapping_element_t *in,
        __nis_mapping_element_t *out);
bool_t is_string_ok(char *, int);

extern FILE *cons;

/*
 * FUNCTION:    free_parse_structs
 *
 *      Release the resources in parse results
 *
 */

void
free_parse_structs()
{
        __nis_table_mapping_t   *t;
        __nis_table_mapping_t   *t1;

        free_proxy_info(&proxyInfo);
        for (t = ldapTableMapping; t != NULL; t = t1) {
                t1 = t->next;
                free_table_mapping(t);
        }
        ldapTableMapping = NULL;
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    1280 Fri Jul 24 12:28:13 2015
new/usr/src/lib/libnisdb/nis_parse_ldap_util.h
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2015 Gary Mills
 */

#ifndef _NIS_PARSE_LDAP_UTIL_H
#define _NIS_PARSE_LDAP_UTIL_H

/*
 * Functions defined in nis_parse_ldap_util.c needed elsewhere
 */

#ifdef __cplusplus
extern "C" {
#endif

extern void *s_malloc(size_t);
extern bool_t add_column(__nis_table_mapping_t *, const char *);
extern bool_t dup_index(__nis_index_t *, __nis_index_t *);
extern bool_t dup_mapping_element(__nis_mapping_element_t *,
    __nis_mapping_element_t *);
extern bool_t make_fqdn(__nis_object_dn_t *, const char *);
extern bool_t make_full_dn(char **, const char *);
extern void append_dot(char **);
extern void append_comma(char **);
extern __nis_mapping_rule_t **dup_mapping_rules(__nis_mapping_rule_t **,
    int n_rules);
extern __nis_mapping_rule_t *dup_mapping_rule(__nis_mapping_rule_t *);

#ifdef __cplusplus
}
#endif

#endif /* _NIS_PARSE_LDAP_UTIL_H */
```

```
**********************************************************
   33225 Fri Jul 24 12:28:13 2015
new/usr/src/lib/libnisdb/nis_parse_ldap_yp_util.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <errno.h>
#include <syslog.h>

#include "ldap_parse.h"
#include "nis_parse_ldap_conf.h"
#include "nis_parse_ldap_util.h"
#include "nis_parse_ldap_err.h"
#include "ldap_util.h"

/* Forward declarations */
int getfullmapname(char **, const char *);
int checkfullmapname(const char *, const char *, __nis_table_mapping_t **,
    __nis_table_mapping_t **);
int append_domainContext(__nis_table_mapping_t **, char *, char *);
extern __nis_mapping_rule_t **dup_mapping_rules(
        __nis_mapping_rule_t **rules, int n_rules);
extern __nis_mapping_rule_t *dup_mapping_rule(
        __nis_mapping_rule_t *in);

static int      merge_table_mapping(__nis_table_mapping_t *in,
        __nis_table_mapping_t *out);
__nis_table_mapping_t *new_merged_mapping(const char *,
        __nis_table_mapping_t *intbl);
static int append_mapping_rule(__nis_mapping_rule_t *src_rule,
        __nis_table_mapping_t *tbl, int flag);


static int copy_object_dn(__nis_object_dn_t     *in,
                __nis_object_dn_t       *newdn);
```



```
/*
 * FUNCTION:    initialize_table_mapping
 *
 * Initialize the __nis_table_mapping_t structure.
 *
 * INPUT:       __nis_table_mapping_t
 *
 */
void
initialize_table_mapping(
        __nis_table_mapping_t *mapping)
{
        if (mapping != NULL) {
                mapping->dbId = NULL;

                mapping->index.numIndexes = 0;
                mapping->index.name = NULL;
                mapping->index.value = NULL;

                mapping->numColumns = 0;
                mapping->column = NULL;

                mapping->initTtlLo = (time_t)NO_VALUE_SET;
                mapping->initTtlHi = (time_t)NO_VALUE_SET;
                mapping->ttl = (time_t)NO_VALUE_SET;

                mapping->usedns_flag = 0;
                mapping->securemap_flag = 0;
                mapping->commentChar = DEFAULT_COMMENT_CHAR;
                mapping->numSplits = 0;

                mapping->objectDN = NULL;

                mapping->separatorStr = DEFAULT_SEP_STRING;

                mapping->numRulesFromLDAP = 0;
                mapping->numRulesToLDAP = 0;

                mapping->ruleFromLDAP = NULL;
                mapping->ruleToLDAP = NULL;

                mapping->e = NULL;
                mapping->objName = NULL;
                mapping->objPath = NULL;
                mapping->obj = NULL;
                mapping->isMaster = 0;
                mapping->seq_num = NO_VALUE_SET;
        }
}
______unchanged_portion_omitted_

/*
 * FUNCTION:    merge_table_mapping
 *
 * Merges information from one table_mapping struct
 * into another
 *
 * INPUT: Source and Destination table_mapping structs.
 * RETURN: 0 on success and > 0 on error.
 */

static int
merge_table_mapping(
        __nis_table_mapping_t *in,
        __nis_table_mapping_t *out)
```

```
141 {
142         int i;
141         int len;
143         int orig_num_rules;
144         int append;

146         if (in == NULL)
147                 return (1);

149         if (in->dbId == NULL)
150                 return (1);

152         /*
153          * If 'in' is generic (non-expanded) and 'out' is domain-specific,
154          * then rules from 'in' should not be appended to those in 'out'.
155          */
156         if (!strchr(in->dbId, COMMA_CHAR) && strchr(out->dbId, COMMA_CHAR))
157                 append = 0;
158         else
159                 append = 1;


162         if (!out->index.numIndexes && in->index.numIndexes > 0) {
163                 if (!dup_index(&in->index, &out->index))
164                         return (1);
165         }

167         /* add_column() increments numColumns, so we don't */
168         if (!out->numColumns && in->numColumns > 0) {
169                 for (i = 0; i < in->numColumns; i++) {
170                         if (!add_column(out, in->column[i]))
171                                 return (1);
172                 }
173         }

175         if (out->commentChar == DEFAULT_COMMENT_CHAR &&
176             in->commentChar != DEFAULT_COMMENT_CHAR)
177                 out->commentChar = in->commentChar;

179         if (out->usedns_flag == 0)
180                 out->usedns_flag = in->usedns_flag;

182         if (out->securemap_flag == 0)
183                 out->securemap_flag = in->securemap_flag;

185         if ((strcmp(out->separatorStr, DEFAULT_SEP_STRING) == 0) &&
186             (strcmp(in->separatorStr, DEFAULT_SEP_STRING) != 0)) {
187                 out->separatorStr = s_strdup(in->separatorStr);
188                 if (!out->separatorStr)
189                         return (2);
190         }

192         if (!out->numSplits && !out->e && in->e) {
193                 out->numSplits = in->numSplits;
194                 out->e = (__nis_mapping_element_t *)
195                     s_calloc(1, (in->numSplits+1) *
196                     sizeof (__nis_mapping_element_t));
197                 if (!out->e)
198                         return (2);
199                 for (i = 0; i <= in->numSplits; i++) {
200                         if (!dup_mapping_element(&in->e[i], &out->e[i])) {
201                                 for (; i > 0; i--) {
202                                         free_mapping_element(&out->e[i - 1]);
203                                 }
204                                 out->e = NULL;
205                                 return (1);
```



```
206                         }
207                 }
208         }

210         if (out->initTtlLo == (time_t)NO_VALUE_SET &&
211             in->initTtlLo != (time_t)NO_VALUE_SET)
212                 out->initTtlLo = in->initTtlLo;

214         if (out->initTtlHi == (time_t)NO_VALUE_SET &&
215             in->initTtlHi != (time_t)NO_VALUE_SET)
216                 out->initTtlHi = in->initTtlHi;

218         if (out->ttl == (time_t)NO_VALUE_SET &&
219             in->ttl != (time_t)NO_VALUE_SET)
220                 out->ttl = in->ttl;

222         if (!out->numRulesFromLDAP && in->numRulesFromLDAP) {
223                 out->ruleFromLDAP = dup_mapping_rules(in->ruleFromLDAP,
224                     in->numRulesFromLDAP);
225                 if (!out->ruleFromLDAP)
226                         return (1);
227                 out->numRulesFromLDAP = in->numRulesFromLDAP;
228         } else if (append && out->numRulesFromLDAP && in->numRulesFromLDAP) {
229                 orig_num_rules = out->numRulesFromLDAP;
230                 for (i = 0; i < in->numRulesFromLDAP; i++) {
231                         if (append_mapping_rule(in->ruleFromLDAP[i], out, 0)) {
232                                 for (i = out->numRulesFromLDAP;
233                                     i > orig_num_rules; i--) {
234                                         free_mapping_rule(out->ruleFromLDAP[i]);
235                                         out->ruleFromLDAP[i] = NULL;
236                                 }
237                                 return (1);

239                         }
240                 }
241         }

243         if (!out->numRulesToLDAP && in->numRulesToLDAP) {
244                 out->ruleToLDAP = dup_mapping_rules(in->ruleToLDAP,
245                     in->numRulesToLDAP);
246                 if (!out->ruleToLDAP)
247                         return (1);
248                 out->numRulesToLDAP = in->numRulesToLDAP;
249         } else if (append && out->numRulesToLDAP && in->numRulesToLDAP) {
250                 orig_num_rules = out->numRulesToLDAP;
251                 for (i = 0; i < in->numRulesToLDAP; i++) {
252                         if (append_mapping_rule(in->ruleToLDAP[i], out, 1)) {
253                                 for (i = out->numRulesToLDAP;
254                                     i > orig_num_rules; i--) {
255                                         free_mapping_rule(out->ruleToLDAP[i]);
256                                         out->ruleToLDAP[i] = NULL;
257                                 }
258                                 return (1);
259                         }
260                 }
261         }
262         if (!out->objectDN && in->objectDN) {
263                 out->objectDN = (__nis_object_dn_t *)
264                     s_calloc(1, sizeof (__nis_object_dn_t));
265                 if (!out->objectDN)
266                         return (2);
267                 if (copy_object_dn(in->objectDN, out->objectDN)) {
268                         free_object_dn(out->objectDN);
269                         out->objectDN = NULL;
270                         return (1);
271                 }
```

```
272         }

274         if (!out->objName && in->objName) {
275                 if (!strchr(in->objName, SPACE_CHAR)) {
276                         /* objName has no space- a single map dbIdMapping */
277                         out->objName = s_strndup(in->objName,
278                             strlen(in->objName));
279                         if (!out->objName)
280                                 return (2);
281                 }
282         }

284         if (!out->objName && out->dbId) {
285                 out->objName = s_strndup(out->dbId, strlen(out->dbId));
286                 if (!out->objName)
287                         return (2);
288         }

290         if (out->seq_num == NO_VALUE_SET && in->seq_num >= 0)
291                 out->seq_num = in->seq_num;

293         return (p_error == no_parse_error ? 0 : 1);
294 }
```

unchanged_portion_omitted_

```
423 /*
424  * FUNCTION:    second_parser_pass
425  *
426  * Prepares the linked list of table_mappings for processing
427  * by finish_parse(), adding, merging and deleting structures
428  * as necessary. Also adds dummy objectDN info. for splitField's.
429  *
430  * RETURN VALUE: 0 on success, > 0 on failure.
431  */
432 int
433 second_parser_pass(__nis_table_mapping_t **table_mapping)
434 {
435         __nis_table_mapping_t   *t, *t2;
436         __nis_table_mapping_t   *t_new = NULL, *tg;
437         __nis_table_mapping_t   *prev = NULL;
437         __nis_object_dn_t   *objectDN;
438         char    *objs, *dom;
439         char    *objName = NULL;
440         char    *lasts;
441         char    *tobj, *alias, *dupalias, *tmp;
442         char    *myself = "second_parser_pass";
443         int     i = 0, len;
444         int     remove_t = 0;
445         int     add_t = 0;

445         prev = NULL;
446         for (t = *table_mapping; t != NULL; ) {
447                 /*
448                  * Temporarily using this field to flag deletion.
449                  * 0 : don't delete
450                  * 1 : delete
451                  * The mapping structure will be deleted in final_parser_pass
452                  */
453                 t->isMaster = 0;

455                 if (!t->dbId) {
456                         p_error = parse_bad_map_error;
457                         logmsg(MSG_NOTIMECHECK, LOG_ERR,
458                             "%s: no dbId field", myself);
459                         return (1);
460                 }
```



```
461                 tg = NULL;
462                 dom = strchr(t->dbId, COMMA_CHAR);
463                 if (t->objName != NULL) {
464                         objName = strdup(t->objName);
465                         if (objName == NULL) {
466                                 p_error = parse_no_mem_error;
467                                 logmsg(MSG_NOMEM, LOG_ERR,
468                                     "%s: Cannot allocate memory for objName",
469                                     myself);
470                                 return (1);
471                         }
472                         objs = (char *)strtok_r(objName, " ", &lasts);
473                         /* Get the generic mapping */
474                         if (dom != NULL) {
475                                 tg = find_table_mapping(t->dbId, dom - t->dbId,
476                                     *table_mapping);
477                         }
478                 } else {
479                         objs = NULL;
480                         if (dom == NULL) {
481                                 t->objName = s_strndup(t->dbId,
482                                     strlen(t->dbId));
483                                 if (!t->objName) {
484                                         logmsg(MSG_NOMEM, LOG_ERR,
485                                             "%s: Cannot allocate memory for "
486                                             "t->objName", myself);
487                                         objs = NULL;
488                                         return (2);
489                                 }
490                         } else {
491                                 /* Force relationship for domain specific */

493                                 /* Get the generic mapping */
494                                 tg = find_table_mapping(t->dbId, dom - t->dbId,
495                                     *table_mapping);
496                                 if (tg == NULL || tg->objName == NULL) {
497                                         /* If not found, use dbId for objName */
498                                         t->objName = s_strndup(t->dbId,
499                                             strlen(t->dbId));
500                                         if (t->objName == NULL) {
501                                                 logmsg(MSG_NOMEM, LOG_ERR,
502                                             "%s: Cannot allocate memory for t->objName",
503                                                     myself);
504                                                 return (2);
505                                         }
506                                 } else {
507                                         dom++;
508                                         tobj = s_strndup(tg->objName,
509                                             strlen(tg->objName));
510                                         if (tobj == NULL) {
511                                                 logmsg(MSG_NOMEM, LOG_ERR,
512                                             "%s: Cannot allocate memory for t->objName",
513                                                     myself);
514                                                 return (2);
515                                         }
516                                         alias = (char *)strtok_r(tobj, " ",
517                                             &lasts);

519                                         /* Loop 'breaks' on errors */
520                                         while (alias) {
521                                                 tmp = NULL;
522                                                 dupalias = s_strndup(alias,
523                                                     strlen(alias));
524                                                 if (!dupalias)
525                                                         break;
526                                                 if (getfullmapname(&dupalias,
```

```
527                                                 dom)) {
528                                                     i = 1;
529                                                     break;
530                                             }
531                                             if (t->objName == NULL)
532                                                     t->objName = dupalias;
533                                             else {
534                                                     len = strlen(t->objName)
535                                                         + strlen(dupalias) +
536                                                         2;
537                                                     tmp = s_calloc(1, len);
538                                                     if (tmp == NULL)
539                                                             break;
540                                                     snprintf(tmp, len,
541                                                         "%s %s",
542                                                         t->objName,
543                                                         dupalias);
544                                                     free(dupalias);
545                                                     dupalias = NULL;
546                                                     free(t->objName);
547                                                     t->objName = tmp;
548                                             }
549                                             alias = (char *)strtok_r(NULL,
550                                                 " ", &lasts);
551                                     }

553                                     if (tobj)
554                                             free(tobj);

556                                     if (alias ||
557                                         (objName = s_strdup(t->objName))
558                                         == NULL) {
559                                             if (i)
560                                                     logmsg(MSG_NOTIMECHECK,
561                                                         LOG_ERR,
562                     "%s: getfullmapname failed for %s for domain \"%s\"",
563                                                         myself, dupalias,
564                                                         dom);
565                                             else {
566                                                     p_error =
567                                                         parse_no_mem_error;
568                                                     logmsg(MSG_NOMEM,
569                                                         LOG_ERR,
570                                                 "%s: Cannot allocate memory",
571                                                         myself);
572                                             }
573                                             if (dupalias)
574                                                     free(dupalias);
575                                             if (t->objName)
576                                                     free(t->objName);
577                                             return (2);

579                                     }
580                                     objs = (char *)strtok_r(objName, " ",
581                                         &lasts);
582                             }
583                     }
584             }

586             if (tg != NULL) {
587                     if (merge_table_mapping(tg, t)) {
588                             logmsg(MSG_NOTIMECHECK, LOG_ERR,
589         "Error merging information from the %s to the %s mapping structure",
590                                 tg->dbId, t->dbId);
591                         objs = NULL;
592                         if (objName)
```



```
593                                     free(objName);
594                             return (1);
595                     }
596             }

598             /*
599              * If objName is "map1 map2" then do the second pass.
600              * If it is just "map1" however skip the expansion.
601              * Also skip it if t->objName is null.
602              */
603             if (objs && strncasecmp(objs, t->objName,
604                 strlen(t->objName))) {
605                     t2 = find_table_mapping(objs, strlen(objs),
606                         *table_mapping);
607                     if (t2) {
608                             if (merge_table_mapping(t, t2)) {
609                                     logmsg(MSG_NOTIMECHECK, LOG_ERR,
610         "Error merging information from the %s to the %s mapping structure",
611                                         t->dbId, t2->dbId);
612                                     objs = NULL;
613                                     if (objName)
614                                             free(objName);
615                                     return (1);
616                             }
617                             t->isMaster = 1;
618                     } else {
619                             t_new = new_merged_mapping(objs, t);
620                             if (t_new) {
621                                     t->isMaster = 1;
622                                     if (prev != NULL)
623                                             prev->next = t_new;
624                                     else
625                                             *table_mapping = t_new;
626                                     prev = t_new;
627                                     prev->next = t;
628                             } else {
629                                     logmsg(MSG_NOTIMECHECK, LOG_ERR,
630                                         "Error creating a new mapping structure %s",
631                                             objs);
632                                     objs = NULL;
633                                     if (objName)
634                                             free(objName);
635                                     return (1);
636                             }
637                     }
638                     while ((objs = (char *)strtok_r(NULL, " ", &lasts))
639                         != NULL) {
640                             t2 = find_table_mapping(objs, strlen(objs),
641                                 *table_mapping);
642                             if (t2) {
643                                     if (merge_table_mapping(t, t2)) {
644                                             logmsg(MSG_NOTIMECHECK, LOG_ERR,
645         "Error merging information from the %s to the %s mapping structure",
646                                                 t->dbId, t2->dbId);
647                                             objs = NULL;
648                                             if (objName)
649                                                     free(objName);
650                                             return (1);
651                                     }
652                                     t->isMaster = 1;
653                             } else {
654                                     /*
655                                      * create a new t_map with dbId = objs
656                                      * and copy t->* into new t_map
657                                      */
658                                     t_new = new_merged_mapping(objs, t);
```

```
659                                         if (t_new) {
660                                                 t->isMaster = 1;
661                                                 if (prev != NULL)
662                                                         prev->next = t_new;
663                                                 else
664                                                         *table_mapping = t_new;
665                                                 prev = t_new;
666                                                 prev->next = t;
667                                         } else {
668                                                 logmsg(MSG_NOTIMECHECK, LOG_ERR,
669                                             "Error creating a new mapping structure %s",
670                                                         objs);
671                                                 objs = NULL;
672                                                 if (objName)
673                                                         free(objName);
674                                                 return (1);
675                                         }
676                                 }
677                         }
678                 } /* if objs!= NULL */

680                 prev = t;
681                 t = t->next;

683                 if (objName) {
684                         free(objName);
685                         objName = NULL;
686                         objs = NULL;
687                 }
688         } /* for t = table_mapping loop */
689         return (0);
690 }
```

_____***unchanged_portion_omitted_***

****
886 Fri Jul 24 12:28:13 2015
new/usr/src/lib/libnisdb/nis_parse_ldap_yp_util.h
5910 libnisdb won't build with modern GCC
****

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2015 Gary Mills
 */

#ifndef _NIS_PARSE_LDAP_YP_UTIL_H
#define _NIS_PARSE_LDAP_YP_UTIL_H

/*
 * Functions defined in nis_parse_ldap_yp_util.c needed elsewhere
 */

#ifdef __cplusplus
extern "C" {
#endif

extern int check_domain_specific_order(const char *, config_key,
    __nis_table_mapping_t *, __yp_domain_context_t *);
extern void initialize_table_mapping(__nis_table_mapping_t *);

#ifdef __cplusplus
}
#endif

#endif /* _NIS_PARSE_LDAP_YP_UTIL_H */
```

```
**********************************************************
   15830 Fri Jul 24 12:28:13 2015
new/usr/src/lib/libnisdb/nisdb_rw.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright (c) 2001 by Sun Microsystems, Inc.
 * All rights reserved.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
#include "db_dictionary_c.h"
#include "nisdb_rw.h"
#include "nisdb_ldap.h"

/*
 * Nesting-safe RW locking functions. Return 0 when successful, an
 * error number from the E-series when not.
 */

int
__nisdb_rwinit(__nisdb_rwlock_t *rw) {

        int     ret;

        if (rw == 0) {
#ifdef  NISDB_MT_DEBUG
                abort();
#endif  /* NISDB_MT_DEBUG */
                return (EFAULT);
        }

        if ((ret = mutex_init(&rw->mutex, USYNC_THREAD, 0)) != 0)
                return (ret);
        if ((ret = cond_init(&rw->cv, USYNC_THREAD, 0)) != 0)
                return (ret);
        rw->destroyed = 0;

        /*
         * If we allow read-to-write lock migration, there's a potential
```



```
         * race condition if two or more threads want to upgrade at the
         * same time. The simple and safe (but crude and possibly costly)
         * method to fix this is to always use exclusive locks, and so
         * that has to be the default.
         *
         * There are two conditions under which it is safe to set
         * 'force_write' to zero for a certain lock structure:
         *
         * (1)  The lock will never be subject to migration, or
         *
         * (2)  It's OK if the data protected by the lock has changed
         *      (a)     Every time the lock (read or write) has been
         *              acquired (even if the lock already was held by
         *              the thread), and
         *      (b)     After every call to a function that might have
         *              acquired the lock.
         */
        rw->force_write = NISDB_FORCE_WRITE;

        rw->writer_count = rw->reader_count = rw->reader_blocked = 0;
        rw->writer.id = rw->reader.id = INV_PTHREAD_ID;
        rw->writer.count = rw->reader.count = 0;
        rw->writer.next = rw->reader.next = 0;

        return (0);
}
```

_____unchanged_portion_omitted_

```
int
__nisdb_rw_readlock_ok(__nisdb_rwlock_t *rw) {
        int             ret;
        pthread_t       myself = pthread_self();
        __nisdb_rl_t    *rr;

        if (rw == 0)
                return (EFAULT);

        if (rw->destroyed != 0)
                return (ESHUTDOWN);

        if ((ret = mutex_lock(&rw->mutex)) != 0)
                return (ret);

        /*
         * Only allow changing 'force_write' when it's really safe; i.e.,
         * the lock hasn't been destroyed, and there are no readers.
         */
        if (rw->destroyed == 0 && rw->reader_count == 0) {
                rw->force_write = 0;
                ret = 0;
        } else {
                ret = EBUSY;
        }

        (void) mutex_unlock(&rw->mutex);

        return (ret);
}


int
__nisdb_rw_force_writelock(__nisdb_rwlock_t *rw) {
        int             ret;
        pthread_t       myself = pthread_self();
        __nisdb_rl_t    *rr;
```

```
140         if (rw == 0 || rw->destroyed != 0)
141                 return (ESHUTDOWN);

143         if ((ret = mutex_lock(&rw->mutex)) != 0)
144                 return (ret);

146         /*
147          * Only allow changing 'force_write' when it's really safe; i.e.,
148          * the lock hasn't been destroyed, and there are no readers.
149          */
150         if (rw->destroyed == 0 && rw->reader_count == 0) {
151                 rw->force_write = 1;
152                 ret = 0;
153         } else {
154                 ret = EBUSY;
155         }

157         (void) mutex_unlock(&rw->mutex);

159         return (ret);
160 }


163 int
164 __nisdb_wlock_trylock(__nisdb_rwlock_t *rw, int trylock) {

166         int             ret;
167         pthread_t       myself = pthread_self();
168         int             all_readers_blocked = 0;
169         __nisdb_rl_t    *rr = 0;

171         if (rw == 0) {
172 #ifdef  NISDB_MT_DEBUG
173                 /* This shouldn't happen */
174                 abort();
175 #endif  /* NISDB_MT_DEBUG */
176                 return (EFAULT);
177         }

179         if (rw->destroyed != 0)
180                 return (ESHUTDOWN);

182         if ((ret = mutex_lock(&rw->mutex)) != 0)
183                 return (ret);

185         if (rw->destroyed != 0) {
186                 (void) mutex_unlock(&rw->mutex);
187                 return (ESHUTDOWN);
188         }

190         /* Simplest (and probably most common) case: no readers or writers */
191         if (rw->reader_count == 0 && rw->writer_count == 0) {
192                 rw->writer_count = 1;
193                 rw->writer.id = myself;
194                 rw->writer.count = 1;
195                 return (mutex_unlock(&rw->mutex));
196         }

198         /*
199          * Need to know if we're holding a read lock already, and if
200          * all other readers are blocked waiting for the mutex.
201          */
202         if (rw->reader_count > 0) {
203                 if ((rr = find_reader(myself, rw)) != 0) {
204                         if (rr->count) {
```



```
209                         if (rr->count)
205                                 /*
206                                  * We're already holding a read lock, so
207                                  * if the number of readers equals the number
208                                  * of blocked readers plus one, all other
209                                  * readers are blocked.
210                                  */
211                                 if (rw->reader_count ==
212                                         (rw->reader_blocked + 1))
213                                         all_readers_blocked = 1;
214                         } else {
219                         else
215                                 /*
216                                  * We're not holding a read lock, so the
217                                  * number of readers should equal the number
218                                  * of blocked readers if all readers are
219                                  * blocked.
220                                  */
221                                 if (rw->reader_count == rw->reader_blocked)
222                                         all_readers_blocked = 1;
223                         }
224                 }
225         }

227         /* Wait for reader(s) or writer to finish */
228         while (1) {
229                 /*
230                  * We can stop looping if one of the following holds:
231                  *      - No readers, no writers
232                  *      - No writers (or writer is myself), and one of:
233                  *              - No readers
234                  *              - One reader, and it's us
235                  *              - N readers, but all blocked on the mutex
236                  */
237                 if (
238                     (rw->writer_count == 0 && rw->reader_count == 0) ||
239                     (((rw->writer_count == 0 || rw->writer.id == myself) &&
240                     (rw->reader_count == 0)) ||
243                         ((rw->writer_count == 0 || rw->writer.id == myself) &&
244                                 (rw->reader_count == 0) ||
241                         (rw->reader_count == 1 &&
242                         rw->reader.id == myself))) {
243                             break;
244                 }
245                 /*
246                  * Provided that all readers are blocked on the mutex
247                  * we break a potential dead-lock by acquiring the
248                  * write lock.
249                  */
250                 if (all_readers_blocked) {
251                         if (rw->writer_count == 0 || rw->writer.id == myself) {
252                                 break;
253                         }
254                 }

256                 /*
257                  * If 'trylock' is set, tell the caller that we'd have to
258                  * block to obtain the lock.
259                  */
260                 if (trylock) {
261                         (void) mutex_unlock(&rw->mutex);
262                         return (EBUSY);
263                 }

265                 /* If we're also a reader, indicate that we're blocking */
266                 if (rr != 0) {
```

```
267                         rr->wait = 1;
268                         rw->reader_blocked++;
269                 }
270                 if ((ret = cond_wait(&rw->cv, &rw->mutex)) != 0) {
271                         if (rr != 0) {
272                                 rr->wait = 0;
273                                 if (rw->reader_blocked > 0)
274                                         rw->reader_blocked--;
275 #ifdef  NISDB_MT_DEBUG
276                                 else
277                                         abort();
278 #endif  /* NISDB_MT_DEBUG */
279                         }
280                         (void) mutex_unlock(&rw->mutex);
281                         return (ret);
282                 }
283                 if (rr != 0) {
284                         rr->wait = 0;
285                         if (rw->reader_blocked > 0)
286                                 rw->reader_blocked--;
287 #ifdef  NISDB_MT_DEBUG
288                         else
289                                 abort();
290 #endif  /* NISDB_MT_DEBUG */
291                 }
292         }

294         /* OK to grab the write lock */
295         rw->writer.id = myself;
296         /* Increment lock depth */
297         rw->writer.count++;
298         /* Set number of writers (doesn't increase with lock depth) */
299         if (rw->writer_count == 0)
300                 rw->writer_count = 1;

302         return (mutex_unlock(&rw->mutex));
303 }
```

unchanged_portion_omitted_

```
642 int
643 __nisdb_destroy_lock(__nisdb_rwlock_t *rw) {

645         int             ret;
646         pthread_t       myself = pthread_self();
651         __nisdb_rl_t    *rr;


649         if (rw == 0) {
650 #ifdef  NISDB_MT_DEBUG
651                 abort();
652 #endif  /* NISDB_MT_DEBUG */
653                 return (EFAULT);
654         }

656         if (rw->destroyed != 0)
657                 return (ESHUTDOWN);

659         if ((ret = mutex_lock(&rw->mutex)) != 0)
660                 return (ret);

662         if (rw->destroyed != 0) {
663                 (void) mutex_unlock(&rw->mutex);
664                 return (ESHUTDOWN);
665         }
```



```
667         /*
668          * Only proceed if if there are neither readers nor writers
669          * other than this thread. Also, no nested locks may be in
670          * effect.
671          */
672         if (((rw->writer_count > 0 &&
673             (rw->writer.id != myself || rw->writer.count != 1)) ||
677         if ((rw->writer_count > 0 &&
678                         (rw->writer.id != myself || rw->writer.count != 1) ||
674             (rw->reader_count > 0 &&
675             !(rw->reader_count == 1 && rw->reader.id == myself &&
676             rw->reader.count == 1))) ||
677             (rw->writer_count > 0 && rw->reader_count > 0)) {
678 #ifdef  NISDB_MT_DEBUG
679                 abort();
680 #endif  /* NISDB_MT_DEBUG */
681                 (void) mutex_unlock(&rw->mutex);
682                 return (ENOLCK);
683         }

685         /*
686          * Mark lock destroyed, so that any thread waiting on the mutex
687          * will know what's what. Of course, this is a bit iffy, since
688          * we're probably being called from a destructor, and the structure
689          * where we live will soon cease to exist (i.e., be freed and
690          * perhaps re-used). Still, we can only do our best, and give
691          * those other threads the best chance possible.
692          */
693         rw->destroyed++;

695         return (mutex_unlock(&rw->mutex));
696 }
```

unchanged_portion_omitted_

****
742 Fri Jul 24 12:28:13 2015
new/usr/src/lib/libnisdb/xdr_nullptr.h
5910 libnisdb won't build with modern GCC
****

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2015 Gary Mills
 */

#ifndef _XDR_NULLPTR_H
#define _XDR_NULLPTR_H

/*
 * This function is used to control serializing and de-serializing of the
 * database objects.
 */

#ifdef __cplusplus
extern "C" {
#endif

extern bool_t xdr_nullptr(XDR *, void **);

#ifdef __cplusplus
}
#endif

#endif /* _XDR_NULLPTR_H */
```

****
48386 Fri Jul 24 12:28:13 2015
new/usr/src/lib/libnisdb/yptol/dit_access_utils.c
5910 libnisdb won't build with modern GCC
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * DESCRIPTION: Contains dit_access interface support functions.
 */
#include <sys/systeminfo.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/systeminfo.h>
#include <unistd.h>
#include <stdlib.h>
#include <syslog.h>
#include <ndbm.h>
#include <strings.h>
#include <errno.h>
#include <ctype.h>
#include "../ldap_util.h"
#include "../ldap_map.h"
#include "../ldap_parse.h"
#include "../ldap_structs.h"
#include "../ldap_val.h"
#include "../ldap_ruleval.h"
#include "../ldap_op.h"
#include "../ldap_attr.h"
#include "../ldap_nisdbquery.h"
#include "../nisdb_mt.h"
#include "shim.h"
#include "yptol.h"
#include "dit_access_utils.h"

#define YPMULTI         "YP_MULTI_"
#define YPMULTISZ       9               /* == strlen(YPMULTI) */

/*
 * Returns 'map,domain.'
 */
```



```
char *
getFullMapName(char *map, char *domain) {
        char    *myself = "getFullMapName";
        char    *objPath;
        if (map == 0 || domain == 0) {
                return (0);
        }
        objPath =  scat(myself, T, scat(myself, F, map, ","),
                scat(myself, F, domain, "."));

        return (objPath);
}
```

*_____unchanged_portion_omitted_*

```
/*
 * Returns an array of rule-values corresponding to the
 * splitfields.
 */
__nis_rule_value_t *
processSplitField(__nis_table_mapping_t *sf, __nis_value_t *inVal,
                        int *nv, int *statP) {

        char                    *sepset;
        __nis_rule_value_t      *rvq;
        __nis_mapping_format_t  *ftmp;
        __nis_value_t           **valA, *tempVal;
        int                     i, j, res, numVals, oldlen, count;
        char                    *str, *oldstr;
        char                    *myself = "processSplitField";

        /* sf will be non NULL */

        if (inVal == 0 || inVal->type != vt_string) {
                *statP =  MAP_PARAM_ERROR;
                return (0);
        }

        /* Get the separator list */
        sepset = sf->separatorStr;

        /* Initialize rule-value */
        rvq = 0;
        count = 0;

        if ((tempVal = stringToValue(inVal->val->value,
                        inVal->val->length)) == 0) {
                *statP = MAP_NO_MEMORY;
                return (0);
        }

        str = oldstr = tempVal->val->value;
        oldlen = tempVal->val->length;

        while (str) {
                tempVal->val->value = str;
                tempVal->val->length = strlen(str) + 1;

                /* Loop to check which format matches str */
                for (i = 0; i <= sf->numSplits; i++) {
                        valA = matchMappingItem(sf->e[i].element.match.fmt,
                                tempVal, &numVals, sepset, &str);
                        if (valA == 0) {
                                /* The format didn't match. Try the next one */
                                continue;
                        }
```

```
169                         /*
170                          * If we are here means we had a match.
171                          * Each new set of values obtained from the match is
172                          * added to a new rule-value. This is to preserve the
173                          * the distinction between each set.
174                          */
175                         rvq = growRuleValue(count, count + 1, rvq, 0);
176                         if (rvq == 0) {
177                                 *statP = MAP_INTERNAL_ERROR;
178                                 for (j = 0; j < numVals; j++)
179                                         freeValue(valA[j], 1);
180                                 sfree(valA);
181                                 tempVal->val->value = oldstr;
182                                 tempVal->val->length = oldlen;
183                                 freeValue(tempVal, 1);
184                                 return (0);
185                         }
186                         count++;

188                         for (j = 0; j < numVals; j++) {
189                                 res = addCol2RuleValue(vt_string,
190                                         sf->e[i].element.match.item[j].name,
191                                         valA[j]->val->value,
192                                         valA[j]->val->length,
193                                         &rvq[count - 1]);
194                                 if (res == -1) {
195                                         *statP = MAP_INTERNAL_ERROR;
196                                         for (; j < numVals; j++)
197                                                 freeValue(valA[j], 1);
198                                         sfree(valA);
199                                         tempVal->val->value = oldstr;
200                                         tempVal->val->length = oldlen;
201                                         freeValue(tempVal, 1);
202                                         freeRuleValue(rvq, count);
203                                         return (0);
204                                 }
205                                 freeValue(valA[j], 1);
206                         }
207                         sfree(valA);

209                         /*
210                          * Since we had a match, break out of this loop
211                          * to parse remainder of str
212                          */
213                         break;
214                 }

216                 /* Didn't find any match, so get out of the loop */
217                 if (i > sf->numSplits) {
218                         str = 0;
219                         break;
220                 }

222                 /* Skip the separators before looping back */
223                 if (str) {
224                         str = str + strspn(str, sepset);
225                         if (*str == '\0')
226                                 break;
227                 }
228         }

230         tempVal->val->value = oldstr;
231         tempVal->val->length = oldlen;
232         freeValue(tempVal, 1);

234         if (str == 0) {
```



```
235                 freeRuleValue(rvq, count);
236                 return (0);
237         }

239         if (nv != 0)
240                 *nv = count;

242         return (rvq);
243 }

245 /*
246  * Convert the datum to an array of RuleValues
247  */
248 __nis_rule_value_t *
249 datumToRuleValue(datum *key, datum *value, __nis_table_mapping_t *t,
250                         int *nv, char *domain, bool_t readonly, int *statP) {

252         __nis_rule_value_t      *rvq, *subrvq, *newrvq;
253         __nis_value_t           *val;
254         __nis_value_t           **valA;
255         __nis_table_mapping_t   *sf;
256         int                     valueLen, comLen, numVals, nr, count = 1;
257         int                     i, j, k, l;
257         int                     i, j, k, l, af;
258         char                    *ipaddr, *ipvalue;

260         /*  At this point, 't' is always non NULL */

262         /* Initialize rule-value */
263         if ((rvq = initRuleValue(1, 0)) == 0) {
264                 *statP = MAP_INTERNAL_ERROR;
265                 return (0);
266         }

268         /* Add domainname to rule-value */
269         if (addCol2RuleValue(vt_string, N2LDOMAIN, domain, strlen(domain),
270                                                 rvq)) {
271                 freeRuleValue(rvq, 1);
272                 *statP = MAP_INTERNAL_ERROR;
273                 return (0);
274         }

276         /* Handle key */
277         if (key != 0) {
278                 /* Add field=value pair for N2LKEY */
279                 i = addCol2RuleValue(vt_string, N2LKEY, key->dptr, key->dsize,
280                                                 rvq);

282                 /* For readonly, add field=value pair for N2LSEARCHKEY */
283                 if (readonly == TRUE && i == 0) {
284                         i = addCol2RuleValue(vt_string, N2LSEARCHKEY, key->dptr,
285                                                 key->dsize, rvq);
286                 }
287                 if (i) {
288                         freeRuleValue(rvq, 1);
289                         *statP = MAP_INTERNAL_ERROR;
290                         return (0);
291                 }

293                 /* Add field=value pairs for IP addresses */
294                 if (checkIPaddress(key->dptr, key->dsize, &ipaddr) > 0) {
295                         /* If key is IPaddress, use preferred format */
296                         ipvalue = ipaddr;
297                         valueLen = strlen(ipaddr);
298                         i = addCol2RuleValue(vt_string, N2LIPKEY, ipvalue,
299                                                 valueLen, rvq);
```

```
300                 } else {
301                         /* If not, use original value for N2LSEARCHIPKEY */
302                         ipaddr = 0;
303                         ipvalue = key->dptr;
304                         valueLen = key->dsize;
305                         i = 0;
306                 }

308                 if (readonly == TRUE && i == 0) {
309                         i = addCol2RuleValue(vt_string, N2LSEARCHIPKEY, ipvalue,
310                                                                 valueLen, rvq);
311                 }
312                 sfree(ipaddr);
313                 if (i) {
314                         freeRuleValue(rvq, 1);
315                         *statP = MAP_INTERNAL_ERROR;
316                         return (0);
317                 }
318         }

320         /* Handle datum value */
321         if (value != 0 && t->e) {
322                 valueLen = value->dsize;
323                 /*
324                  * Extract the comment, if any, and add it to
325                  * the rule-value.
326                  */
327                 if (t->commentChar != '\0') {
328                         /*
329                          * We loop on value->dsize because value->dptr
330                          * may not be NULL-terminated.
331                          */
332                         for (i = 0; i < value->dsize; i++) {
333                                 if (value->dptr[i] == t->commentChar) {
334                                         valueLen = i;
335                                         comLen = value->dsize - i - 1;
336                                         if (comLen == 0)
337                                                 break;
338                                         if (addCol2RuleValue(vt_string,
339                                                 N2LCOMMENT, value->dptr + i + 1,
340                                                 comLen, rvq)) {
341                                                 freeRuleValue(rvq, 1);
342                                                 *statP = MAP_INTERNAL_ERROR;
343                                                 return (0);
344                                         }
345                                         break;
346                                 }
347                         }
348                 }

350                 /* Skip trailing whitespaces */
351                 for (; valueLen > 0 && (value->dptr[valueLen - 1] == ' ' ||
352                         value->dptr[valueLen - 1] == '\t'); valueLen--);

354                 /*
355                  * At this point valueLen is the effective length of
356                  * the data. Convert value into __nis_value_t so that
357                  * we can use the matchMappingItem function to break it
358                  * into fields.
359                  */
360                 if ((val = stringToValue(value->dptr, valueLen)) == 0) {
361                         freeRuleValue(rvq, 1);
362                         *statP = MAP_NO_MEMORY;
363                         return (0);
364                 }
```



```
366                 /* Perform namefield match */
367                 valA = matchMappingItem(t->e->element.match.fmt, val,
368                                                         &numVals, 0, 0);
369                 if (valA == 0) {
370                         freeValue(val, 1);
371                         freeRuleValue(rvq, 1);
372                         *statP = MAP_NAMEFIELD_MATCH_ERROR;
373                         return (0);
374                 }

376                 /* We don't need val anymore, so free it */
377                 freeValue(val, 1);

379                 /*
380                  * Since matchMappingItem only returns us an array of
381                  * __nis_value_t's, we need to associate each value
382                  * in the array with the corresponding item name.
383                  * This code assumes that numVals will be less than or
384                  * equal to the number of item names associated with
385                  * the format.
386                  * These name=value pairs are added to rvq.
387                  */
388                 for (i = 0, *statP = SUCCESS; i < numVals; i++) {
389                         for (j = 0; j < count; j++) {
390                                 if (addCol2RuleValue(vt_string,
391                                         t->e->element.match.item[i].name,
392                                         valA[i]->val->value,
393                                         valA[i]->val->length, &rvq[j])) {
394                                         *statP = MAP_INTERNAL_ERROR;
395                                         break;
396                                 }
397                         }
398                         if (*statP == MAP_INTERNAL_ERROR)
399                                 break;

401                         /*
402                          * Check if splitField exists for the field.
403                          * Since splitfields are also stored as mapping
404                          * structures, we need to get the hash table entry
405                          * corresponding to the splitfield name
406                          */
407                         sf = mappingFromMap(t->e->element.match.item[i].name,
408                                         domain, statP);
409                         if (*statP == MAP_NO_MEMORY)
410                                 break;
411                         *statP = SUCCESS;
412                         if (sf == 0)
413                                 continue;

415                         /*
416                          * Process and add splitFields to rule-value rvq
417                          */
418                         subrvq = processSplitField(sf, valA[i], &nr, statP);

420                         if (subrvq == 0) {
421                                 /* statP would have been set */
422                                 break;
423                         }

425                         /*
426                          * We merge 'count' rule-values in rvq with 'nr'
427                          * rule-values from subrvq to give us a whopping
428                          * 'count * nr' rule-values
429                          */

431                         /* Initialize the new rule-value array */
```

```
                        if ((newrvq = initRuleValue(count * nr, 0)) == 0) {
                                *statP = MAP_INTERNAL_ERROR;
                                freeRuleValue(subrvq, nr);
                                break;
                        }

                        for (j = 0, l = 0; j < nr; j++) {
                                for (k = 0; k < count; k++, l++) {
                                        if ((mergeRuleValue(&newrvq[l],
                                                        &rvq[k]) == -1) ||
                                                        (mergeRuleValue(
                                                        &newrvq[l],
                                                        &subrvq[j]) == -1)) {
                                                *statP = MAP_INTERNAL_ERROR;
                                                for (i = 0; i < numVals; i++)
                                                        freeValue(valA[i], 1);
                                                sfree(valA);
                                                freeRuleValue(rvq, count);
                                                freeRuleValue(newrvq,
                                                                count * nr);
                                                freeRuleValue(subrvq, nr);
                                                return (0);
                                        }
                                }
                        }

                        freeRuleValue(rvq, count);
                        rvq = newrvq;
                        count = l;
                        freeRuleValue(subrvq, nr);

                }

                /* We don't need valA anymore, so free it */
                for (i = 0; i < numVals; i++)
                        freeValue(valA[i], 1);
                sfree(valA);

                if (*statP != SUCCESS) {
                        freeRuleValue(rvq, count);
                        return (0);
                }

        } /* if value */

        if (nv != 0)
                *nv = count;
        return (rvq);

}
```

*unchanged_portion_omitted_*

```
/*
 * Updates 'rv' with NIS name=value pairs suitable to
 * construct datum from namefield information.
 * Some part based on createNisPlusEntry (from ldap_nisdbquery.c)
 * This code assumes that from a given LDAP entry, applying the
 * mapping rules, would give us one or more NIS entries, differing
 * only in key.
 */
suc_code
buildNISRuleValue(__nis_table_mapping_t *t, __nis_rule_value_t *rv,
                                char *domain) {
        int                     r, i, j, k, l, nrq, res, len;
        int                     r, i, j, k, l, index, nrq, res, len;
        int                     numItems, splitname, count, statP;
```



```
        __nis_value_t           *rval;
        __nis_mapping_item_t    *litem;
        __nis_mapping_rule_t    *rl;
        __nis_rule_value_t      *rvq;
        char                    *value, *emptystr = "";

        statP = SUCCESS;

        /* Initialize default base */
        __nisdb_get_tsd()->searchBase = t->objectDN->read.base;

        /* Initialize rule-value rvq */
        rvq = 0;
        count = 0;

        /* Add domainname to rule-value */
        if (addCol2RuleValue(vt_string, N2LDOMAIN, domain, strlen(domain),
                                                        rv)) {
                return (MAP_INTERNAL_ERROR);
        }

        for (r = 0; r < t->numRulesFromLDAP; r++) {
                rl = t->ruleFromLDAP[r];

                /* Set escapeFlag if RHS is "dn" to remove escape chars */
                if (rl->rhs.numElements == 1 &&
                        rl->rhs.element->type == me_item &&
                        rl->rhs.element->element.item.type == mit_ldap &&
                        strcasecmp(rl->rhs.element->element.item.name, "dn")
                                                                == 0) {
                                __nisdb_get_tsd()->escapeFlag = '2';
                }

                rval = buildRvalue(&rl->rhs, mit_ldap, rv, NULL);

                /* Reset escapeFlag */
                __nisdb_get_tsd()->escapeFlag = '\0';

                if (rval == 0) {
                        continue;
                }

                if (rval->numVals <= 0) {
                        /* Treat as invalid */
                        freeValue(rval, 1);
                        continue;
                }

                litem = buildLvalue(&rl->lhs, &rval, &numItems);
                if (litem == 0) {
                        /* This will take care of numItems == 0 */
                        freeValue(rval, 1);
                        continue;
                }

                if (rval->numVals > 1) {
                        if (numItems == 1 && litem->repeat)
                                nrq = rval->numVals;
                        else if (numItems > 1 && rval->repeat)
                                nrq = 1 + ((rval->numVals-1)/numItems);
                        else
                                nrq = 1;
                } else
                        nrq = 1;

                /* Set splitname if splitfield names are specified */
```

```
716                 for (i = 0; i < numItems; i++) {
717                         if (strcasecmp(litem[i].name, N2LKEY) == 0 ||
718                                 strcasecmp(litem[i].name, N2LIPKEY) == 0 ||
719                                 strcasecmp(litem[i].name, N2LCOMMENT) == 0)
720                                 continue;
721                         for (j = 0; j < t->numColumns; j++) {
722                                 if (strcmp(litem[i].name, t->column[j]) == 0)
723                                         break;
724                         }
725                         if (j == t->numColumns)
726                                 break;
727                 }

729                 splitname = (i < numItems)?1:0;

731                 for (j = 0; j < nrq; j++) {
732                         if (splitname == 1) {
733                                 /*
734                                  * Put every value of splitfieldname in a new
735                                  * rule-value. Helps generating splitfields.
736                                  */
737                                 rvq = growRuleValue(count, count + 1, rvq, 0);
738                                 if (rvq == 0) {
739                                         freeRuleValue(rvq, count);
740                                         freeValue(rval, 1);
741                                         freeMappingItem(litem, numItems);
742                                         return (MAP_INTERNAL_ERROR);
743                                 }
744                                 count++;
745                         }

747                         for (k = j % nrq, l = 0; l < numItems; k += nrq, l++) {
748                                 /* If we run out of values, use empty strings */
749                                 if (k >= rval->numVals) {
750                                         value = emptystr;
751                                         len = 0;
752                                 } else {
753                                         value = rval->val[k].value;
754                                         len = rval->val[k].length;
755                                 }
756                                 res = (splitname == 1)?addCol2RuleValue(
757                                         vt_string, litem[l].name, value,
758                                         len, &rvq[count - 1]):0;
759                                 if (res != -1)
760                                         res = addCol2RuleValue(vt_string,
761                                                 litem[l].name, value, len, rv);
762                                 if (res == -1) {
763                                         freeRuleValue(rvq, count);
764                                         freeValue(rval, 1);
765                                         freeMappingItem(litem, numItems);
766                                         return (MAP_INTERNAL_ERROR);
767                                 }
768                         }
769                 }
770                 freeValue(rval, 1);
771                 rval = 0;
772                 freeMappingItem(litem, numItems);
773                 litem = 0;
774                 numItems = 0;
775         } /* for r < t->numRulesFromLDAP */

777         statP = addSplitFieldValues(t, rvq, rv, count, domain);

779         if (rvq)
780                 freeRuleValue(rvq, count);
```



```
782         if (verifyIndexMatch(t, 0, rv, 0, 0) == 0)
783                 return (MAP_INDEXLIST_ERROR);
784         return (statP);

786 } /* end of buildNISRuleValue */
_____unchanged_portion_omitted_

1003 /*
1004  * Read (i.e get and map) a single NIS entry from the LDAP DIT
1005  */
1006 bool_t
1007 singleReadFromDIT(char *map, char *domain, datum *key, datum *value,
1008                                                         int *statP) {
1009         __nis_table_mapping_t   *t;
1010         __nis_rule_value_t      *rv_request = 0, *rv_result = 0;
1011         __nis_ldap_search_t     *ls;
1012         __nis_object_dn_t       *objectDN = NULL;
1013         int                     i, rc, nr = 0;
1013         int                     i, rc, nr = 0, nv = 0;
1014         datum                   *datval = 0;
1015         char                    *skey, *str;
1015         char                    *skey, *str, *sipkey;
1016         char                    *myself = "singleReadFromDIT";

1018         *statP = SUCCESS;

1020         if (!map || !domain || !key || !value) {
1021                 *statP = MAP_PARAM_ERROR;
1022                 return (FALSE);
1023         }


1026         /* Get the mapping information for the map */
1027         if ((t = mappingFromMap(map, domain, statP)) == 0) {
1028                 /*
1029                  * No problem. We don't handle this map and domain. Maybe it's
1030                  * handled by a service other than NIS.
1031                  */
1032                 return (FALSE);
1033         }

1035         /* NULL-terminated version of datum key for logging */
1036         if ((skey = am(myself, key->dsize + 1)) == 0) {
1037                 *statP = MAP_NO_MEMORY;
1038                 return (FALSE);
1039         }
1040         (void) memcpy(skey, key->dptr, key->dsize);

1042         if ((str = getFullMapName(map, domain)) == 0) {
1043                 *statP = MAP_NO_MEMORY;
1044                 return (FALSE);
1045         }

1047         /* For each alternate mapping */
1048         for (; t != 0; t = t->next) {
1049                 /* Verify objName */
1050                 if (strcmp(str, t->objName) != 0) {
1051                         continue;
1052                 }

1054                 /* Verify if key matches the index */
1055                 if (verifyIndexMatch(t, 0, 0, N2LKEY, skey) == 0 ||
1056                         verifyIndexMatch(t, 0, 0, N2LIPKEY, skey) == 0)
1057                         continue;

1059                 /* Check if rulesFromLDAP are provided */
```

```
1060                if (t->numRulesFromLDAP == 0) {
1061                        logmsg(MSG_NOTIMECHECK, LOG_INFO,
1062                                "%s: No rulesFromLDAP information available "
1063                                "for %s (%s)", myself, t->dbId, map);
1064                        continue;
1065                }

1067                /* Convert key into rule-value */
1068                if ((rv_request = datumToRuleValue(key, 0, t, 0, domain, TRUE,
1069                                                        statP)) == 0) {
1070                        logmsg(MSG_NOTIMECHECK, LOG_ERR,
1071                                "%s: Conversion error %d (NIS to name=value "
1072                                "pairs) for NIS key (%s) for %s (%s)",
1073                                myself, *statP, skey, t->dbId, map);
1074                        continue;
1075                }
1076                /* Convert rule-value into ldap request */
1077                for (objectDN = t->objectDN; objectDN &&
1078                                objectDN->read.base;
1079                                objectDN = objectDN->next) {
1080                        ls = createLdapRequest(t, rv_request, 0, 1, NULL,
1081                                                        objectDN);
1082                        if (ls == 0) {
1083                                *statP = MAP_CREATE_LDAP_REQUEST_ERROR;
1084                                logmsg(MSG_NOTIMECHECK, LOG_ERR,
1085                                        "%s: Failed to create ldapSearch "
1086                                        "request for "
1087                                        "NIS key (%s) for %s (%s) "
1088                                        "for base %s",
1089                                        myself, skey, t->dbId, map,
1090                                        objectDN->read.base);
1091                                continue;
1092                        }
1093                        ls->timeout.tv_sec = SINGLE_ACCESS_TIMEOUT_SEC;
1094                        ls->timeout.tv_usec = SINGLE_ACCESS_TIMEOUT_USEC;
1095                        /* Query LDAP */
1096                        nr = (ls->isDN)?0:-1;
1097                        rv_result = ldapSearch(ls, &nr, 0, statP);
1098                        freeLdapSearch(ls);
1099                        if (rv_result == 0) {
1100                                if (*statP == LDAP_NO_SUCH_OBJECT) {
1101                                        /* Entry does not exist in */
1102                                        /* the ldap server */
1103                                }
1104                                continue;
1105                        }
1106                        freeRuleValue(rv_request, 1);
1107                        rv_request = 0;

1109                        /* if result > 1, first match will be returned */
1110                        if (nr > 1) {
1111                                logmsg(MSG_NOTIMECHECK, LOG_INFO,
1112                                        "%s: %d ldapSearch results "
1113                                        "for NIS key (%s) "
1114                                        "for %s (%s) for base %s. "
1115                                        "First match will be returned ",
1116                                        myself, nr, skey, t->dbId, map,
1117                                        objectDN->read.base);
1118                        }

1120                        for (i = 0; i < nr; i++) {
1121                                /* Convert LDAP data to NIS equivalents */
1122                                *statP = buildNISRuleValue(t, &rv_result[i],
1123                                                        domain);
1124                                if (*statP == MAP_INDEXLIST_ERROR)
1125                                        continue;
```



```
1127                                if (*statP != SUCCESS) {
1128                                        logmsg(MSG_NOTIMECHECK, LOG_WARNING,
1129                                        "%s: Conversion error %d (LDAP to "
1130                                        "name=value pairs) for NIS key (%s) "
1131                                        "for %s (%s) for base %s", myself,
1132                                        *statP, skey,
1133                                        t->dbId, map, objectDN->read.base);
1134                                        continue;
1135                                }

1137                        /*
1138                         * Check if 'key' from the ldap result matches the key
1139                         * provided by our caller
1140                         */
1141                                if ((rc = verifyKey(skey, &rv_result[i]))
1142                                        == -1) {
1143                                        logmsg(MSG_NOTIMECHECK, LOG_INFO,
1144                                        "%s: Cannot verify key from ldap "
1145                                        "result for NIS key (%s) for %s (%s) "
1146                                        "for base %s",
1147                                        myself, skey, t->dbId, map,
1148                                        objectDN->read.base);
1149                                        continue;
1150                                }

1152                                if (rc == 1) {
1153                                        datval = ruleValueToDatum(t,
1154                                                        &rv_result[i], statP);
1155                                        if (datval == 0) {
1156                                                logmsg(MSG_NOTIMECHECK,
1157                                                LOG_WARNING,
1158                                                "%s: Conversion error %d "
1159                                                "(name=value pairs to NIS) "
1160                                                "for NIS key (%s) for %s (%s)"
1161                                                " for base %s",
1162                                                myself,
1163                                                *statP, skey, t->dbId, map,
1164                                                objectDN->read.base);
1165                                                continue;
1166                                        }
1167                                        if (value) {
1168                                                value->dptr = datval->dptr;
1169                                                value->dsize = datval->dsize;
1170                                        }
1171                                        sfree(datval);
1172                                        sfree(skey);
1173                                        freeRuleValue(rv_result, nr);
1174                                        rv_result = 0;
1175                                        *statP = SUCCESS;

1177                                        /* Free full map name */
1178                                        sfree(str);

1180                                        return (TRUE);
1181                                }
1182                        }
1183                        freeRuleValue(rv_result, nr);
1184                        rv_result = 0;
1185                }   /* end of for over objectDN */

1187                if (rv_request != 0) {
1188                        freeRuleValue(rv_request, 1);
1189                        rv_request = 0;
1190                }
1191                if (rv_result != 0) {
```

```
1192                        freeRuleValue(rv_result, nr);
1193                        rv_result = 0;
1194                }
1195        }
1196        sfree(skey);
1197        *statP = MAP_NO_MATCHING_KEY;

1199        /* Free full map name */
1200        sfree(str);

1202        return (FALSE);
1203 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   10135 Fri Jul 24 12:28:13 2015
new/usr/src/lib/libnisdb/yptol/lock_update.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2015 Gary Mills
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * DESCRIPTION: Contains code supporting the 'update in progress' flag. This is
 *              a near copy of lock flag code (in
 *              usr/src/cmd/ypcmd/shared/lockmp.c) If we implement a clean
 *              version of the locking code this file will probably disappear.
 *
 *              These locks are held while a map is being updated from the
 *              DIT. They prevent a second update being started while this is
 *              in progress. This is independant from the 'lockmap' mechanism
 *              which protects maps, generally for a much shorter period,
 *              while their control structures are modified.
 */

#include <unistd.h>
#include <syslog.h>
#include <sys/mman.h>
#include <thread.h>
#include <synch.h>
#include <ndbm.h>
#include <strings.h>
#include "ypsym.h"
#include "shim.h"
#include "yptol.h"
#include "../ldap_util.h"

#define LOCKFILE "/var/run/yp_mapupdate"
struct updatearray {
        mutex_t         updatenode[MAXHASH];
};
typedef struct updatearray updatearray;

/*
```



```
 * Cross-process robust mutex locks.
 * Provide synchronization between YP processes
 * by implementing an exclusive locking mechanism
 * via a memory-mapped file.
 */
static struct updatearray       *shmupdatearray;
static int      lockfile;

bool_t
init_update_locks_mem()
{
        int iiter, rc;
        int ebusy_cnt = 0;

        /*
         * Initialize cross-process locks in memory-mapped file.
         */
        for (iiter = 0; iiter < MAXHASH; iiter++) {
                if ((rc = mutex_init(&(shmupdatearray->updatenode[iiter]),
                    USYNC_PROCESS | LOCK_ROBUST, 0)) != 0) {
                if (rc = mutex_init(&(shmupdatearray->updatenode[iiter]),
                    USYNC_PROCESS | LOCK_ROBUST, 0)) {
                        if (rc == EBUSY) {
                                ebusy_cnt++;
                        } else {
                                logmsg(MSG_NOTIMECHECK, LOG_ERR,
                                        "init_update_locks_mem():mutex_init():"
                                        "error=%d", rc);
                                return (FALSE);
                        }
                }
        }

        /*
         * EBUSY for all locks OK, it means another process
         * has already initialized locks.
         */
        if ((ebusy_cnt > 0) && (ebusy_cnt != MAXHASH)) {
                logmsg(MSG_NOTIMECHECK, LOG_ERR,
                "%s inconsistent. Remove this file and restart NIS (YP)",
                                                                LOCKFILE);
                return (FALSE);
        }
        return (TRUE);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    9820 Fri Jul 24 12:28:13 2015
new/usr/src/lib/libnisdb/yptol/map_conv.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * DESCRIPTION: Contains functions relating to movement of entire maps.
 */

#include <unistd.h>
#include <syslog.h>
#include <ndbm.h>
#include <string.h>
#include "ypsym.h"
#include "ypdefs.h"
#include "shim.h"
#include "yptol.h"
#include "../ldap_util.h"

/*
 * Switch on parts of ypdefs.h
 */
USE_YPDBPATH
USE_DBM

/*
 * Decs
 */
void add_separator(char *);
suc_code dump_domain_to_dit(char *, bool_t);
suc_code dump_map_to_dit(char *, char *, bool_t);
suc_code dump_maps_to_dit(bool_t);
suc_code dump_dit_to_map();
suc_code dump_dit_to_maps();

/*
 * FUNCTION :   dump_maps_to_dit()
```



```
 *
 * DESCRIPTION: Dump all the OLD STYLE NIS maps into the DIT.
 *
 *              Since the DIT is not yet set up details about which maps and
 *              domains exist are gathered from the N2L config file and the
 *              existing map files.
 *
 * GIVEN :      Flag indicating if containers and domains should be set up.
 *
 * RETURNS :    Success code
 */
suc_code
dump_maps_to_dit(bool_t init_containers)
{
        char **dom_list;
        int num_doms, i;

        num_doms = get_mapping_domain_list(&dom_list);

        /* Dump all domains in list */
        for (i = 0; i < num_doms; i++) {
                if (FAILURE == dump_domain_to_dit(dom_list[i], init_containers))
                        return (FAILURE);
        }

        return (SUCCESS);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
   15750 Fri Jul 24 12:28:13 2015
new/usr/src/lib/libnisdb/yptol/shim.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * DESCRIPTION: Contains the top level shim hook functions. These must have
 *              identical interfaces to the equivalent standard dbm calls.
 *
 *              Unfortunately many of these will do a copy of a datum structure
 *              on return. This is a side effect of the original DBM function
 *              being written to pass structures rather than pointers.
 *
 * NOTE :       There is a major bug/feature in dbm. A key obtained by
 *              dbm_nextkey() of dbm_firstkey() cannot be passed to dbm_store().
 *              When the store occurs dbm's internal memory get's reorganized
 *              and the static strings pointed to by the key are destroyed. The
 *              data is then stored in the wrong place. We attempt to get round
 *              this by dbm_firstkey() and dbm_nextkey() making a copy of the
 *              key data in malloced memory. This is freed when map_ctrl is
 *              freed.
 */

#include <unistd.h>
#include <syslog.h>
#include <ndbm.h>
#include <strings.h>
#include "ypsym.h"
#include "ypdefs.h"
#include "shim.h"
#include "yptol.h"
#include "stubs.h"
#include "../ldap_parse.h"
#include "../ldap_util.h"

/*
 * Globals
 */
```



```
bool_t yptol_mode = FALSE;      /* Set if in N2L mode */
bool_t yptol_newlock = FALSE;
                                /*
                                 * Set if in N2L mode and we want to use the new
                                 * lock mapping mechanism
                                 */
bool_t ypxfrd_flag = FALSE;     /* Set if called from ypxfrd */
pid_t parent_pid;                       /* ID of calling parent process */


/*
 * Decs
 */
void check_old_map_date(map_ctrl *);

/*
 * Constants
 */
/* Number of times to try to update a map before giving up */
/* #define MAX_UPDATE_ATTEMPTS 3 */
#define MAX_UPDATE_ATTEMPTS 1

/*
 * FUNCTION:    shim_dbm_close();
 *
 * INPUTS:      Identical to equivalent dbm call.
 *
 * OUTPUTS:     Identical to equivalent dbm call.
 *
 */
void
shim_dbm_close(DBM *db)
{
        map_ctrl *map;

        /* Lock the map */
        map = get_map_ctrl(db);
        if (map == NULL)
                return;

        free_map_ctrl(map);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    9071 Fri Jul 24 12:28:14 2015
new/usr/src/lib/libnisdb/yptol/shim_ancil.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T     */
/*        All Rights Reserved   */

/*
 * Portions of this source code were derived from Berkeley 4.3 BSD
 * under license from the Regents of the University of California.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifndef lint
static  char sccsid[] = "%Z%%M% %I%     %E% SMI";
#endif

#include <stdlib.h>
#include <dirent.h>
#include <strings.h>
#include "ypsym.h"
#include "ypdefs.h"
USE_YPDBPATH
USE_DBM
#include "shim.h"
#include "../ldap_util.h"

/*
 * This constructs a file name from a passed domain name, a passed map name,
 * and a globally known YP data base path prefix.
 *
 * Has to be in shim because it needs the N2L prefix
 *
 * RETURNS :    TRUE = A name was successfully created
 *              FALSE = A name could not be created
 */

bool_t
```



```
ypmkfilename(domain, map, path)
        char *domain;
        char *map;
        char *path;
{
        int length;

        /* Do not allow any path as a domain name. */
        if (strchr(domain, '/') != NULL)
                return (FALSE);

        length = strlen(domain) + strlen(map) + ypdbpath_sz + 3;
        if (yptol_mode)
                length += strlen(NTOL_PREFIX) + 1;

        if ((MAXNAMLEN + 1) < length) {
                (void) fprintf(stderr, "ypserv:  Map name string too long.\n");
                return (FALSE);
        }

        strcpy(path, ypdbpath);
        strcat(path, "/");
        strcat(path, domain);
        strcat(path, "/");

        /* If in N2L mode add N2L prefix */
        if (yptol_mode)
                strcat(path, NTOL_PREFIX);
        strcat(path, map);

        return (TRUE);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   34492 Fri Jul 24 12:28:14 2015
new/usr/src/lib/libnisdb/yptol/shim_changepasswd.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * DESCRIPTION: This is the N2L equivalent of changepasswd.c. The traditional
 *              version modifies the NIS source files and then initiates a
 *              ypmake to make the maps and push them.
 *
 *              For N2L there are no source files and the policy is that the
 *              definitive information is that contained in the DIT. Old
 *              information is read from LDAP. Assuming this authenticates, and
 *              the change is acceptable, this information is modified and
 *              written back to LDAP.
 *
 *              Related map entries are then found and  updated finally
 *              yppushes of the changed maps are initiated. Since the
 *              definitive information has already correctly been updated the
 *              code is tolerant of some errors during this operation.
 *
 *              What was previously in the maps is irrelevant.
 *
 *              Some less than perfect code (like inline constants for
 *              return values and a few globals) is retained from the original.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <ctype.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <syslog.h>
#include <pwd.h>
#include <signal.h>
```



```
#include <crypt.h>
#include <rpc/rpc.h>
#include <rpcsvc/yppasswd.h>
#include <utmpx.h>
#include <shadow.h>

#include <ndbm.h>
/* DO NOT INCLUDE SHIM_HOOKS.H */
#include "shim.h"
#include "yptol.h"
#include "../ldap_util.h"

/*
 * Undocumented external function in libnsl
 */
extern int  getdomainname(char *, int);

/* Constants */
#define CRYPTPWSIZE CRYPT_MAXCIPHERTEXTLEN
#define STRSIZE 100
#define FINGERSIZE (4 * STRSIZE - 4)
#define SHELLSIZE (STRSIZE - 2)

#define UTUSERLEN (sizeof (((struct utmpx *)0)->ut_user))
#define COLON_CHAR ':'

/*
 * Path to DBM files. This is only required for N2L mode. Traditional mode
 * works with the source files and uses the NIS Makefile to generate the maps.
 * Seems to be hard coded in the rest of NIS so same is done here.
 */
#define YPDBPATH "/var/yp"

/* Names of password and adjunct mappings. Used to access DIT */
#define BYNAME ".byname"
#define BYUID ".byuid"
#define BYGID ".bygid"
#define PASSWD_MAPPING "passwd" BYNAME
#define PASSWD_ADJUNCT_MAPPING "passwd.adjunct" BYNAME
#define AGEING_MAPPING "ageing" BYNAME

/* Bitmasks used in list of fields to change */
#define CNG_PASSWD      0x0001
#define CNG_SH          0x0002
#define CNG_GECOS       0x0004

/* Globals :-( */
extern int single, nogecos, noshell, nopw, mflag;

/*
 * Structure for containing the information is currently in the DIT. This is
 * similar to the passwd structure defined in getpwent(3C) apart from.
 *
 * 1. Since GID and UID are never changed they are not converted to integers.
 * 2. There are extra fields to hold adjunct information.
 * 3. There are extra fields to hold widely used information.
 */
struct passwd_entry {
        char    *pw_name;
        char    *pw_passwd;
        char    *pw_uid;
        char    *pw_gid;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
        char    *adjunct_tail;  /* Tail of adjunct entry (opaque) */
```

```
126         bool_t  adjunct;        /* Flag indicating if DIT has adjunct info */
127         char    *pwd_str;       /* New password string */
128         char    *adjunct_str;   /* New adjunct string */
129 };
______unchanged_portion_omitted_

418 /*
419  * FUNCTION:    proc_map_list()
420  *
421  * DESCRIPTION: Finds entries in one list of map that need to be updated.
422  *              updates them and writes them back.
423  *
424  * INPUTS:      Null terminated list of maps to process.
425  *              Domain name
426  *              Information to write (including user name)
427  *              Flag indicating if this is the adjunct list
428  *
429  * OUTPUTS:     An error code
430  */
431 int
432 proc_map_list(char **map_list, char *domain,
433                                 struct passwd_entry *pwd, bool_t adjunct_flag)
434 {
435         char    *myself = "proc_map_list";
436         char    *map_name;
437         char    cmdbuf[BUFSIZ];
438         int     map_name_len = 0;
439         int     index, ans = 0;
436         int     res;

441         /* If this is a adjunct list check LDAP had some adjunct info */
442         if ((adjunct_flag) && (!pwd->adjunct)) {
443                 logmsg(MSG_NOTIMECHECK, LOG_INFO,
444                         "Have adjunct map list but no adjunct data in DIT");
445                 /* Not a disaster */
446                 return (0);
447         }

449         /* Allocate enough buffer to take longest map name */
450         for (index = 0; map_list[index] != NULL; index ++)
451                 if (map_name_len < strlen(map_list[index]))
452                         map_name_len = strlen(map_list[index]);
453         map_name_len += strlen(YPDBPATH);
454         map_name_len += strlen(NTOL_PREFIX);
455         map_name_len += strlen(domain);
456         map_name_len += 3;
457         if (NULL == (map_name = am(myself, map_name_len))) {
458                 logmsg(MSG_NOMEM, LOG_ERR, "Could not alloc map name");
459                 return (2);
460         }

462         /* For all maps in list */
463         for (index = 0; map_list[index] != NULL; index ++) {

465                 /* Generate full map name */
466                 strcpy(map_name, YPDBPATH);
467                 add_separator(map_name);
468                 strcat(map_name, domain);
469                 add_separator(map_name);
470                 strcat(map_name, NTOL_PREFIX);
471                 strcat(map_name, map_list[index]);

473                 if (0 != (ans = update_single_map(map_name, pwd, adjunct_flag)))
474                         break;
475         }
```



```
477         /* Done with full map path */
478         sfree(map_name);

480         /*
481          * If (ans != 0) then one more maps have failed. LDAP has however been
482          * updates. This is the definitive source for information there is no
483          * need to unwind. (This was probably due to maps that were already
484          * corrupt).
485          */

487         /*
488          * If it all worked fork off push operations for the maps. Since we
489          * want the map to end up with it's traditional name on the slave send
490          * the name without its LDAP_ prefix. The slave will call ypxfrd
491          * which, since it is running in N2L mode, will put the prefix back on
492          * before reading the file.
493          */
494         if (mflag && (0 == ans)) {
495                 for (index = 0; (map_name = map_list[index]) != NULL;
496                                                                 index ++) {
497                         if (fork() == 0) {
498                                 /*
499                                  * Define full path to yppush. Probably also
500                                  * best for security.
501                                  */
502                                 strcpy(cmdbuf, "/usr/lib/netsvc/yp/yppush ");
503                                 strcat(cmdbuf, map_name);
504                                 if (0 > system(cmdbuf))
505                                         logmsg(MSG_NOTIMECHECK, LOG_ERR,
506                                                 "Could not initiate yppush");
507                                 exit(0);
508                         }
509                 }
510         }
511         return (ans);
512 }
______unchanged_portion_omitted_

1039 /*
1040  * FUNCTION:    get_old_info()
1041  *
1042  * DESCRIPTION: Gets as much information as possible from LDAP about one user.
1043  *
1044  *              This goes through the mapping system. This is messy because
1045  *              them mapping system will build up a password entry from the
1046  *              contents of the DIT. We then have to parse this to recover
1047  *              it's individual fields.
1048  *
1049  * INPUT:       Pointer to user name
1050  *              Domain
1051  *
1052  * OUTPUT:      The info in malloced space. To be freed by caller.
1053  *              NULL on failure.
1054  */
1055 struct passwd_entry *
1056 get_old_info(char *name, char *domain)
1057 {
1058         char *myself = "get_old_info";
1059         struct passwd_entry *old_passwd;
1057         char    *p;
1060         datum   key, data;
1061         suc_code res;

1063         /* Get the password entry */
1064         key.dptr = name;
1065         key.dsize = strlen(key.dptr);
```

```
1066         read_from_dit(PASSWD_MAPPING, domain, &key, &data);
1067         if (NULL == data.dptr) {
1068                 logmsg(MSG_NOTIMECHECK, LOG_ERR,
1069                                         "Could not read old pwd for %s", name);
1070                 return (NULL);
1071         }

1073         /* Pull password apart */
1074         old_passwd = am(myself, sizeof (struct passwd_entry));
1075         if (NULL == old_passwd) {
1076                 logmsg(MSG_NOMEM, LOG_ERR, "Could not alloc for pwd decode");
1077                 sfree(data.dptr);
1078                 return (NULL);
1079         }

1081         /* No data yet */
1082         old_passwd->pw_name = NULL;
1083         old_passwd->pw_passwd = NULL;
1084         old_passwd->pw_uid = NULL;
1085         old_passwd->pw_gid = NULL;
1086         old_passwd->pw_gecos = NULL;
1087         old_passwd->pw_dir = NULL;
1088         old_passwd->pw_shell = NULL;
1089         old_passwd->adjunct_tail = NULL;
1090         old_passwd->pwd_str = NULL;
1091         old_passwd->adjunct_str = NULL;
1092         old_passwd->adjunct = FALSE;

1094         res = decode_pwd_entry(&data, old_passwd, FALSE);
1095         sfree(data.dptr);
1096         if (SUCCESS != res) {
1097                 free_pwd_entry(old_passwd);
1098                 return (NULL);
1099         }

1101         /* Try to get the adjunct entry */
1102         read_from_dit(PASSWD_ADJUNCT_MAPPING, domain, &key, &data);
1103         if (NULL == data.dptr) {
1104                 /* Fine just no adjunct data */
1105                 old_passwd->adjunct = FALSE;
1106         } else {
1107                 res = decode_pwd_entry(&data, old_passwd, TRUE);
1108                 sfree(data.dptr);
1109                 if (SUCCESS != res) {
1110                         free_pwd_entry(old_passwd);
1111                         return (NULL);
1112                 }
1113         }

1115         return (old_passwd);
1116 }
```

_____***unchanged_portion_omitted_***

****
2229 Fri Jul 24 12:28:14 2015
new/usr/src/lib/libnisdb/yptol/shim_lockmap.c
5910 libnisdb won't build with modern GCC
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * DESCRIPTION: Contains a front end to the map locking code. These are called
 *              when a map, or its map_ctrl structure, needs to be locked
 *              for a short time for internal modification. This lock should
 *              not be held between DBM operations.
 *
 * NOTE :       This is not the same mechanism as the 'update lock' which is
 *              held for a relatively long period when a map is being update
 *              from the DIT.
 */

#include <unistd.h>
#include <syslog.h>
#include <sys/mman.h>
#include <thread.h>
#include <synch.h>
#include <ndbm.h>
#include "ypsym.h"
#include "shim.h"
#include "stubs.h"

/*
 * FUNCTION :   lock_map_ctrl()
 *
 * DESCRIPTION: Front end to the lock routine taking map_ctrl structure as
 *              argument. Saves cost of a hash operation.
 *
 * GIVEN :      Map_ctrl structure .
 *
 * RETURNS :    Same as lock core
 */
int
```



```
lock_map_ctrl(map_ctrl *map)
{
        int ret;

        ret = lock_core(map->hash_val);

        return (ret);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7983 Fri Jul 24 12:28:14 2015
new/usr/src/lib/libnisdb/yptol/ttl_utils.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2015 Gary Mills
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * DESCRIPTION: Contains utilities relating to TTL calculation.
 */
#include <unistd.h>
#include <syslog.h>
#include <errno.h>
#include <strings.h>
#include <ndbm.h>
#include "ypsym.h"
#include "ypdefs.h"
#include "shim.h"
#include "yptol.h"
#include "../ldap_util.h"

/*
 * Constants used in time calculations
 */
#define MILLION 1000000

/*
 * Decs
 */
suc_code is_greater_timeval(struct timeval *, struct timeval *);
suc_code add_to_timeval(struct timeval *, int);

/*
 * FUNCTION:    has_entry_expired()
 *
 * DESCRIPTION: Determines if an individual entry has expired.
 *
 * INPUTS:      Map control structure for an open map
 *              Entry key
```



```
 *
 * OUTPUTS:     TRUE =  Entry has expired or cannot be found this will cause
 *                      missing entries to be pulled out of the DIT.
 *              FALSE = Entry has not expired
 *
 */
bool_t
has_entry_expired(map_ctrl *map, datum *key)
{
        datum ttl;
        struct timeval  now;
        struct timeval  old_time;
        char    *key_name;
        char *myself = "has_entry_expired";

        if ((map == NULL) || (map->ttl == NULL))
                return (FALSE);

        /* Get expiry time entry for key */
        ttl = dbm_fetch(map->ttl, *key);

        if (NULL == ttl.dptr) {
                /*
                 * If we failed to get a map expiry key, which must always be
                 * present, then something is seriously wrong. Try to recreate
                 * the map.
                 */
                if ((key->dsize == strlen(MAP_EXPIRY_KEY)) &&
                        (0 == strncmp(key->dptr, MAP_EXPIRY_KEY, key->dsize))) {
                        logmsg(MSG_NOTIMECHECK, LOG_ERR, "Cannot find %s TTL "
                                "for map %s. Will attempt to recreate map",
                                MAP_EXPIRY_KEY, map->map_name);
                        return (TRUE);
                }

                /*
                 * Not a problem just no TTL entry for this entry. Maybe it has
                 * not yet been downloaded. Maybe it will be handled by a
                 * service other than NIS. Check if the entire map has expired.
                 * This prevents repeated LDAP reads when requests are made for
                 * nonexistant entries.
                 */
                if (has_map_expired(map)) {
                        /* Kick of a map update */
                        update_map_if_required(map, FALSE);
                }

                /* Don't update the entry */
                return (FALSE);
        }

        if (ttl.dsize != sizeof (struct timeval)) {
                /*
                 * Need to malloc some memory before can syslog the key name
                 * but this may fail. Solution log a simple message first THEn
                 * a more detailed one if it works.
                 */
                logmsg(MSG_NOTIMECHECK, LOG_ERR,
                        "Invalid TTL key in map %s. error %d",
                                        map->map_name, dbm_error(map->ttl));

                /* Log the key name */
                key_name = (char *)am(myself, key->dsize + 1);
                if (NULL == key_name) {
                        logmsg(MSG_NOMEM, LOG_ERR,
                                        "Could not alloc memory for keyname");
```

```
126                 } else {
127                         strncpy(key_name, key->dptr, key->dsize);
128                         key_name[key->dsize] = '\0';
129                         logmsg(MSG_NOTIMECHECK, LOG_ERR,
130                                                 "Key name was %s", key_name);
131                         sfree(key_name);
132                 }
133                 /* Update it Anyway */
134                 return (TRUE);
135         }

137         /* Get current time */
138         gettimeofday(&now, NULL);

140         /*
141          * Because dptr may not be int aligned need to build an int
142          * out of what it points to or will get a bus error
143          */
144         bcopy(ttl.dptr, &old_time, sizeof (struct timeval));

146         return (is_greater_timeval(&now, &old_time));
147 }
_____unchanged_portion_omitted_

266 /*
267  * FUNCTION:    add_to_timeval()
268  *
269  * DESCRIPTION: Adds an int to a timeval
270  *
271  * NOTE :       Seems strange that there is not a library function to do this
272  *              if one exists then this function can be removed.
273  *
274  * NOTE :       Does not handle UNIX clock wrap round but this is a much bigger
275  *              problem.
276  *
277  * INPUTS:      Time value to add to
278  *              Time value to add in seconds
279  *
280  * OUTPUTS:     SUCCESS = Addition successful
281  *              FAILURE = Addition failed (probably wrapped)
282  *
283  */
284 suc_code
285 add_to_timeval(struct timeval *t1, int t2)
286 {
288         long usec;
287         struct timeval oldval;

289         oldval.tv_sec = t1->tv_sec;

291         /* Add seconds part */
292         t1->tv_sec += t2;

294         /* Check for clock wrap */
295         if (!(t1->tv_sec >= oldval.tv_sec)) {
296                 logmsg(MSG_NOTIMECHECK, LOG_ERR,
297                         "Wrap when adding %d to %d", t2, oldval.tv_sec);
298                 return (FAILURE);
299         }

301         return (SUCCESS);
302 }
_____unchanged_portion_omitted_
```

```
**********************************************************
    9292 Fri Jul 24 12:28:14 2015
new/usr/src/lib/libnisdb/yptol/update.c
5910 libnisdb won't build with modern GCC
**********************************************************
```

```
  1 /*
  2  * CDDL HEADER START
  3  *
  4  * The contents of this file are subject to the terms of the
  5  * Common Development and Distribution License, Version 1.0 only
  6  * (the "License").  You may not use this file except in compliance
  7  * with the License.
  8  *
  9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 10  * or http://www.opensolaris.org/os/licensing.
 11  * See the License for the specific language governing permissions
 12  * and limitations under the License.
 13  *
 14  * When distributing Covered Code, include this CDDL HEADER in each
 15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 16  * If applicable, add the following below this CDDL HEADER, with the
 17  * fields enclosed by brackets "[]" replaced with your own identifying
 18  * information: Portions Copyright [yyyy] [name of copyright owner]
 19  *
 20  * CDDL HEADER END
 21  */
 22 /*
 23  * Copyright 2015 Gary Mills
 24  * Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
 25  * Use is subject to license terms.
 26  */

 27 #pragma ident   "%Z%%M% %I%     %E% SMI"

 28 /*
 29  * DESCRIPTION: Contains the map update thread and related code.
 30  */

 32 #include <unistd.h>
 33 #include <syslog.h>
 34 #include <ndbm.h>
 35 #include <thread.h>
 36 #include <unistd.h>
 37 #include <strings.h>
 38 #include "ypsym.h"
 39 #include "ypdefs.h"
 40 #include "shim.h"
 41 #include "yptol.h"
 42 #include "../ldap_util.h"

 44 /* Enable standard YP code features defined in ypdefs.h */
 45 USE_YP_PREFIX
 46 USE_YP_MASTER_NAME
 47 USE_YP_LAST_MODIFIED
 48 USE_YP_INPUT_FILE
 49 USE_YP_OUTPUT_NAME
 50 USE_YP_DOMAIN_NAME
 51 USE_YP_SECURE
 52 USE_YP_INTERDOMAIN

 54 /*
 55  * Decs
 56  */
 57 suc_code update_from_dit(map_ctrl *, datum *);
 58 void * update_thread(void *);
```



```
 60 /*
 61  * Globals
 62  */
 63 extern pid_t parent_pid;

 65 /*
 66  * FUNCTION:    update_entry_if_required()
 67  *
 68  * DESCRIPTION: Determines if an entry is to be updated and if it is does the
 69  *              update.
 70  *
 71  * GIVEN :      Pointer to the open map ctrl
 72  *              Pointer to the entry key
 73  *
 74  * RETURNS :    SUCCESS = Entry is in a state to be returned to the client
 75  *              i.e. either got updated, did not need to be updated or we are
 76  *              in a mode where it is acceptable to return out of date
 77  *              information.
 78  *              FAILURE = Entry need an update but it could not be done.
 79  */
 80 suc_code
 81 update_entry_if_required(map_ctrl *map, datum *key)
 82 {

 84         /* Only update individual entries if entire map is */
 85         /* not being updated */
 86         if (is_map_updating(map))
 87                 return (SUCCESS);

 89         /*
 90          * If we are being asked for the order then need to check if
 91          * the map is in need of an update. If it is then fake a
 92          * recent order. The client will then read the map, using
 93          * dbm_firstkey and this will do the update.
 94          */
 95         if (0 == strncmp(key->dptr, yp_last_modified, yp_last_modified_sz)) {
 96                 if (has_map_expired(map))
 97                         update_timestamp(map->entries);
 98                 return (SUCCESS);
 99         }

101         /* Never update special keys. Have no TTLs */
102         if (is_special_key(key))
103                 return (SUCCESS);

105         if (!has_entry_expired(map, key))
106                 /* Didn't need an update */
107                 return (SUCCESS);

109         /* Do the update */
110         return (update_from_dit(map, key));
111 }
_____unchanged_portion_omitted_

324 /*
325  * FUNCTION:    update_thread()
326  *
327  * DESCRIPTION: The update thread this is called to update an entire NIS map.
328  *              if several NIS maps are found to be out of date several
329  *              instances of this may be running at the same time.
330  *
331  *              Since we are using a duplicate map_ctrl we do not have to lock
332  *              it. If we did would end up using the same mutex as the parent
333  *              map ctrl an possibly deadlocking.
334  *
335  * INPUTS:      Map handle (because we need access to name and lock)
```

```
 336  *
 337  * OUTPUTS:     None exits when finished.
 338  */

 340 void *
 341 update_thread(void *arg)
 342 {
 343         void *ret = (void *)-1;
 344         map_ctrl *map;

 346         /* Cast argument pointer to correct type */
 347         map = (map_ctrl *)arg;

 349         /* Actually do the work */
 350         if (SUCCESS == update_map_from_dit(map, FALSE))
 351                 ret = 0;

 353         /* Update complete or failed */
 354         unlock_map_update(map);

 356         /* Free up duplicate copy of the map_ctrl */
 357         free_map_ctrl(map);

 359         thr_exit(ret);

 361         return (NULL);
 362 }
```

_____***unchanged_portion_omitted_***