

 f8f776dd18 ▾


...

[TizenRT](#) / [external](#) / [iotivity](#) / [iotivity_1.2-rel](#) / [resource](#) / [csdk](#) / [security](#) / [provisioning](#) / [src](#) / [provisioningdatabasemanager.c](#)



heejin-kim Add iotivity_1.2-rel

 History

 1 contributor

1061 lines (906 sloc) | 32.1 KB

...

```

1  /* *****
2  *
3  * Copyright 2015 Samsung Electronics All Rights Reserved.
4  *
5  *
6  *
7  * Licensed under the Apache License, Version 2.0 (the "License");
8  * you may not use this file except in compliance with the License.
9  * You may obtain a copy of the License at
10 *
11 *     http://www.apache.org/licenses/LICENSE-2.0
12 *
13 * Unless required by applicable law or agreed to in writing, software
14 * distributed under the License is distributed on an "AS IS" BASIS,
15 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
16 * See the License for the specific language governing permissions and
17 * limitations under the License.
18 *
19 * *****/
20
21 #include <stdio.h>
22 #include <stdbool.h>
23 #include <string.h>
24 #include <stdlib.h>
25
26 #include "sqlite3.h"
27 #include "logger.h"
28 #include "oic_malloc.h"

```

```

29 #include "provisioningdatabasemanager.h"
30 #include "putility.h"
31 #include "oic_string.h"
32 #include "utlist.h"
33
34
35 #define DB_FILE "PDM.db"
36
37 #define TAG "OIC_PDM"
38
39 #define PDM_FIRST_INDEX 0
40 #define PDM_SECOND_INDEX 1
41
42 #define PDM_BIND_INDEX_FIRST 1
43 #define PDM_BIND_INDEX_SECOND 2
44 #define PDM_BIND_INDEX_THIRD 3
45
46 #define PDM_CREATE_T_DEVICE_LIST "create table T_DEVICE_LIST(ID INTEGER PRIMARY KEY AUTOINCREMENT,
47                                UUID BLOB NOT NULL UNIQUE, STATE INT NOT NULL);"
48
49 #define PDM_CREATE_T_DEVICE_LINK "create table T_DEVICE_LINK_STATE(ID INT NOT NULL, ID2 INT NOT \
50                                NULL,STATE INT NOT NULL, PRIMARY KEY (ID, ID2));"
51 /**
52  * Macro to verify sqlite success.
53  * eg: VERIFY_NON_NULL(TAG, ptrData, ERROR,OC_STACK_ERROR);
54  */
55 #define PDM_VERIFY_SQLITE_OK(tag, arg, logLevel, retValue) do{ if (SQLITE_OK != (arg)) \
56     { OIC_LOG_V((logLevel), tag, "Error in " #arg " , Error Message: %s", \
57     sqlite3_errmsg(g_db)); return retValue; }}while(0)
58
59 #define PDM_SQLITE_TRANSACTION_BEGIN "BEGIN TRANSACTION;"
60 #define PDM_SQLITE_TRANSACTION_COMMIT "COMMIT;"
61 #define PDM_SQLITE_TRANSACTION_ROLLBACK "ROLLBACK;"
62 #define PDM_SQLITE_GET_STALE_INFO "SELECT ID,ID2 FROM T_DEVICE_LINK_STATE WHERE STATE = ?"
63 #define PDM_SQLITE_INSERT_T_DEVICE_LIST "INSERT INTO T_DEVICE_LIST VALUES(?,?,?)"
64 #define PDM_SQLITE_GET_ID "SELECT ID FROM T_DEVICE_LIST WHERE UUID like ?"
65 #define PDM_SQLITE_INSERT_LINK_DATA "INSERT INTO T_DEVICE_LINK_STATE VALUES(?,?,?)"
66 #define PDM_SQLITE_DELETE_LINK "DELETE FROM T_DEVICE_LINK_STATE WHERE ID = ? and ID2 = ?"
67 #define PDM_SQLITE_DELETE_DEVICE_LINK "DELETE FROM T_DEVICE_LINK_STATE WHERE ID = ? or ID2 = ?"
68 #define PDM_SQLITE_DELETE_DEVICE "DELETE FROM T_DEVICE_LIST WHERE ID = ?"
69 #define PDM_SQLITE_DELETE_DEVICE_WITH_STATE "DELETE FROM T_DEVICE_LIST WHERE STATE= ?"
70 #define PDM_SQLITE_UPDATE_LINK "UPDATE T_DEVICE_LINK_STATE SET STATE = ? WHERE ID = ? and ID2 = ?"
71 #define PDM_SQLITE_LIST_ALL_UUID "SELECT UUID FROM T_DEVICE_LIST WHERE STATE = 0"
72 #define PDM_SQLITE_GET_UUID "SELECT UUID,STATE FROM T_DEVICE_LIST WHERE ID = ?"
73 #define PDM_SQLITE_GET_LINKED_DEVICES "SELECT ID,ID2 FROM T_DEVICE_LINK_STATE WHERE \
74                                     (ID = ? or ID2 = ?) and state = 0"
75 #define PDM_SQLITE_GET_DEVICE_LINKS "SELECT ID,ID2 FROM T_DEVICE_LINK_STATE WHERE \
76                                     ID = ? and ID2 = ? and state = 0"
77 #define PDM_SQLITE_UPDATE_DEVICE "UPDATE T_DEVICE_LIST SET STATE = ? WHERE UUID like ?"

```

```

78 #define PDM_SQLITE_GET_DEVICE_STATUS "SELECT STATE FROM T_DEVICE_LIST WHERE UUID like ?"
79 #define PDM_SQLITE_UPDATE_LINK_STALE_FOR_STALE_DEVICE "UPDATE T_DEVICE_LINK_STATE SET STATE = 1\
80                                     WHERE ID = ? or ID2 = ?"
81
82 #define ASCENDING_ORDER(id1, id2) do{if( (id1) > (id2) )\
83     { int temp; temp = id1; id1 = id2; id2 = temp; }}while(0)
84
85 #define CHECK_PDM_INIT(tag) do{if(true != gInit)\
86     { OIC_LOG(ERROR, (tag), "PDB is not initialized"); \
87     return OC_STACK_PDM_IS_NOT_INITIALIZED; }}while(0)
88
89 static sqlite3 *g_db = NULL;
90 static bool gInit = false; /* Only if we can open sqlite db successfully, gInit is true. */
91
92 /**
93  * function to create DB in case DB doesn't exists
94  */
95 static OCStackResult createDB(const char* path)
96 {
97     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
98
99     int result = 0;
100     result = sqlite3_open_v2(path, &g_db, SQLITE_OPEN_READWRITE|SQLITE_OPEN_CREATE, NULL);
101     PDM_VERIFY_SQLITE_OK(TAG, result, ERROR, OC_STACK_ERROR);
102
103     result = sqlite3_exec(g_db, PDM_CREATE_T_DEVICE_LIST, NULL, NULL, NULL);
104     PDM_VERIFY_SQLITE_OK(TAG, result, ERROR, OC_STACK_ERROR);
105
106     OIC_LOG(INFO, TAG, "Created T_DEVICE_LIST");
107     result = sqlite3_exec(g_db, PDM_CREATE_T_DEVICE_LINK, NULL, NULL, NULL);
108     PDM_VERIFY_SQLITE_OK(TAG, result, ERROR, OC_STACK_ERROR);
109
110     OIC_LOG(INFO, TAG, "Created T_DEVICE_LINK_STATE");
111     gInit = true;
112
113     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
114
115     return OC_STACK_OK;
116 }
117
118
119 /**
120  * Function to begin any transaction
121  */
122 static OCStackResult begin()
123 {
124     int res = 0;
125     res = sqlite3_exec(g_db, PDM_SQLITE_TRANSACTION_BEGIN, NULL, NULL, NULL);
126     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

```

```

127     return OC_STACK_OK;
128 }
129
130 /**
131  * Function to commit any transaction
132  */
133 static OCStackResult commit()
134 {
135     int res = 0;
136     res = sqlite3_exec(g_db, PDM_SQLITE_TRANSACTION_COMMIT, NULL, NULL, NULL);
137     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
138     return OC_STACK_OK;
139 }
140
141 /**
142  * Function to rollback any transaction
143  */
144 static OCStackResult rollback()
145 {
146     int res = 0;
147     res = sqlite3_exec(g_db, PDM_SQLITE_TRANSACTION_ROLLBACK, NULL, NULL, NULL);
148     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
149     return OC_STACK_OK;
150 }
151
152 /**
153  * Error log callback called by SQLite stack in case of error
154  */
155 void errLogCallback(void *pArg, int iErrCode, const char *zMsg)
156 {
157     (void) pArg;
158     (void) iErrCode;
159     (void) zMsg;
160     OIC_LOG_V(DEBUG, TAG, "%s : (%d) %s", __func__, iErrCode, zMsg);
161 }
162
163 OCStackResult PDMInit(const char *path)
164 {
165     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
166
167     int rc;
168     const char *dbPath = NULL;
169     if (SQLITE_OK != sqlite3_config(SQLITE_CONFIG_LOG, errLogCallback, NULL))
170     {
171         OIC_LOG(INFO, TAG, "Unable to enable debug log of sqlite");
172     }
173
174     if (NULL == path || !*path)
175     {

```

```

176         dbPath = DB_FILE;
177     }
178     else
179     {
180         dbPath = path;
181     }
182     rc = sqlite3_open_v2(dbPath, &g_db, SQLITE_OPEN_READWRITE, NULL);
183     if (SQLITE_OK != rc)
184     {
185         OIC_LOG_V(INFO, TAG, "ERROR: Can't open database: %s", sqlite3_errmsg(g_db));
186         sqlite3_close(g_db);
187         OCStackResult ret = createDB(dbPath);
188         if (OC_STACK_OK != ret)
189         {
190             sqlite3_close(g_db);
191         }
192         return ret;
193     }
194     gInit = true;
195
196     /*
197     * Remove PDM_DEVICE_INIT status devices.
198     * PDM_DEVICE_INIT means that the OTM process is in progress.
199     * PDM_DEVICE_INIT state device can be existed when the program is terminated during the OTM p
200     * For this reason, PDM_DEVICE_INIT devices should be removed at PDM initialization time.
201     */
202     if(OC_STACK_OK != PDMDeleteDeviceWithState(PDM_DEVICE_INIT))
203     {
204         OIC_LOG_V(WARNING, TAG, "Failed to delete init state devices.");
205     }
206
207     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
208
209     return OC_STACK_OK;
210 }
211
212
213 OCStackResult PDMAAddDevice(const OicUuid_t *UUID)
214 {
215     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
216
217     CHECK_PDM_INIT(TAG);
218     if (NULL == UUID)
219     {
220         return OC_STACK_INVALID_PARAM;
221     }
222
223     sqlite3_stmt *stmt = 0;
224     int res =0;

```

```

225     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_INSERT_T_DEVICE_LIST,
226                             strlen(PDM_SQLITE_INSERT_T_DEVICE_LIST) + 1, &stmt, NULL);
227     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
228
229     res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_SECOND, UUID, UUID_LENGTH, SQLITE_STATIC);
230     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
231
232     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_THIRD, PDM_DEVICE_INIT);
233     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
234
235     res = sqlite3_step(stmt);
236     if (SQLITE_DONE != res)
237     {
238         if (SQLITE_CONSTRAINT == res)
239         {
240             //new OCStack result code
241             OIC_LOG_V(ERROR, TAG, "Error Occured: %s",sqlite3_errmsg(g_db));
242             sqlite3_finalize(stmt);
243             return OC_STACK_DUPLICATE_UUID;
244         }
245         OIC_LOG_V(ERROR, TAG, "Error Occured: %s",sqlite3_errmsg(g_db));
246         sqlite3_finalize(stmt);
247         return OC_STACK_ERROR;
248     }
249     sqlite3_finalize(stmt);
250
251     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
252     return OC_STACK_OK;
253 }
254
255 /**
256  *function to get Id for given UUID
257  */
258 static OCStackResult getIdForUUID(const OicUuid_t *UUID , int *id)
259 {
260     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
261
262     sqlite3_stmt *stmt = 0;
263     int res = 0;
264     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_ID, strlen(PDM_SQLITE_GET_ID) + 1, &stmt, NULL);
265     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
266
267     res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_FIRST, UUID, UUID_LENGTH, SQLITE_STATIC);
268     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
269
270     OIC_LOG(DEBUG, TAG, "Binding Done");
271     while (SQLITE_ROW == sqlite3_step(stmt))
272     {
273         int tempId = sqlite3_column_int(stmt, PDM_FIRST_INDEX);

```

```

274     OIC_LOG_V(DEBUG, TAG, "ID is %d", tempId);
275     *id = tempId;
276     sqlite3_finalize(stmt);
277     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
278     return OC_STACK_OK;
279 }
280 sqlite3_finalize(stmt);
281 return OC_STACK_INVALID_PARAM;
282 }
283
284 /**
285  * Function to check duplication of device's Device ID.
286  */
287 OCStackResult PDMIsDuplicateDevice(const OicUuid_t* UUID, bool *result)
288 {
289     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
290
291     CHECK_PDM_INIT(TAG);
292     if (NULL == UUID || NULL == result)
293     {
294         OIC_LOG(ERROR, TAG, "UUID or result is NULL");
295         return OC_STACK_INVALID_PARAM;
296     }
297     sqlite3_stmt *stmt = 0;
298     int res = 0;
299     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_ID, strlen(PDM_SQLITE_GET_ID) + 1, &stmt, NULL);
300     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
301
302     res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_FIRST, UUID, UUID_LENGTH, SQLITE_STATIC);
303     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
304
305     OIC_LOG(DEBUG, TAG, "Binding Done");
306     bool retValue = false;
307     while(SQLITE_ROW == sqlite3_step(stmt))
308     {
309         OIC_LOG(INFO, TAG, "Duplicated UUID");
310         retValue = true;
311     }
312
313     sqlite3_finalize(stmt);
314     *result = retValue;
315
316     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
317     return OC_STACK_OK;
318 }
319
320 /**
321  * Function to add link in sqlite
322  */

```

```

323 static OCStackResult addlink(int id1, int id2)
324 {
325     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
326
327     sqlite3_stmt *stmt = 0;
328     int res = 0;
329     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_INSERT_LINK_DATA,
330                             strlen(PDM_SQLITE_INSERT_LINK_DATA) + 1, &stmt, NULL);
331     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
332
333     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id1);
334     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
335
336     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id2);
337     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
338
339     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_THIRD, PDM_DEVICE_ACTIVE);
340     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
341
342     if (sqlite3_step(stmt) != SQLITE_DONE)
343     {
344         OIC_LOG_V(ERROR, TAG, "Error Occured: %s", sqlite3_errmsg(g_db));
345         sqlite3_finalize(stmt);
346         return OC_STACK_ERROR;
347     }
348     sqlite3_finalize(stmt);
349     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
350     return OC_STACK_OK;
351 }
352
353 OCStackResult PDMLinkDevices(const OicUuid_t *UUID1, const OicUuid_t *UUID2)
354 {
355     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
356
357     CHECK_PDM_INIT(TAG);
358     if (NULL == UUID1 || NULL == UUID2)
359     {
360         OIC_LOG(ERROR, TAG, "Invalid PARAM");
361         return OC_STACK_INVALID_PARAM;
362     }
363
364     PdmDeviceState_t state = PDM_DEVICE_UNKNOWN;
365     if (OC_STACK_OK != PDMGetDeviceState(UUID1, &state))
366     {
367         OIC_LOG(ERROR, TAG, "Internal error occurred");
368         return OC_STACK_ERROR;
369     }
370     if (PDM_DEVICE_ACTIVE != state)
371     {

```



```

372     OIC_LOG_V(ERROR, TAG, "UUID1: Device state is not active : %d", state);
373     return OC_STACK_INVALID_PARAM;
374 }
375
376 state = PDM_DEVICE_UNKNOWN;
377 if (OC_STACK_OK != PDMGetDeviceState(UUID2, &state))
378 {
379     OIC_LOG(ERROR, TAG, "Internal error occurred");
380     return OC_STACK_ERROR;
381 }
382 if (PDM_DEVICE_ACTIVE != state)
383 {
384     OIC_LOG_V(ERROR, TAG, "UUID2: Device state is not active : %d", state);
385     return OC_STACK_INVALID_PARAM;
386 }
387
388 int id1 = 0;
389 if (OC_STACK_OK != getIdForUUID(UUID1, &id1))
390 {
391     OIC_LOG(ERROR, TAG, "Requested value not found");
392     return OC_STACK_INVALID_PARAM;
393 }
394 int id2 = 0;
395 if (OC_STACK_OK != getIdForUUID(UUID2, &id2))
396 {
397     OIC_LOG(ERROR, TAG, "Requested value not found");
398     return OC_STACK_INVALID_PARAM;
399 }
400
401 ASCENDING_ORDER(id1, id2);
402 OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
403 return addlink(id1, id2);
404 }
405
406 /**
407  * Function to remove created link
408  */
409 static OCStackResult removeLink(int id1, int id2)
410 {
411     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
412
413     int res = 0;
414     sqlite3_stmt *stmt = 0;
415     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_DELETE_LINK, strlen(PDM_SQLITE_DELETE_LINK) + 1, &stmt, 0);
416     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
417
418     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id1);
419     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
420

```

```

421     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id2);
422     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
423
424     if (SQLITE_DONE != sqlite3_step(stmt))
425     {
426         OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
427         sqlite3_finalize(stmt);
428         return OC_STACK_ERROR;
429     }
430     sqlite3_finalize(stmt);
431     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
432     return OC_STACK_OK;
433 }
434
435 OCStackResult PDMUnlinkDevices(const OicUuid_t *UUID1, const OicUuid_t *UUID2)
436 {
437     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
438
439     CHECK_PDM_INIT(TAG);
440     if (NULL == UUID1 || NULL == UUID2)
441     {
442         OIC_LOG(ERROR, TAG, "Invalid PARAM");
443         return OC_STACK_INVALID_PARAM;
444     }
445
446     int id1 = 0;
447     if (OC_STACK_OK != getIdForUUID(UUID1, &id1))
448     {
449         OIC_LOG(ERROR, TAG, "Requested value not found");
450         return OC_STACK_INVALID_PARAM;
451     }
452
453     int id2 = 0;
454     if (OC_STACK_OK != getIdForUUID(UUID2, &id2))
455     {
456         OIC_LOG(ERROR, TAG, "Requested value not found");
457         return OC_STACK_INVALID_PARAM;
458     }
459     ASCENDING_ORDER(id1, id2);
460     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
461     return removeLink(id1, id2);
462 }
463
464 static OCStackResult removeFromDeviceList(int id)
465 {
466     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
467
468     sqlite3_stmt *stmt = 0;
469     int res = 0;

```

```

470     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_DELETE_DEVICE,
471                               strlen(PDM_SQLITE_DELETE_DEVICE) + 1, &stmt, NULL);
472     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
473
474     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id);
475     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
476
477     if (sqlite3_step(stmt) != SQLITE_DONE)
478     {
479         OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
480         sqlite3_finalize(stmt);
481         return OC_STACK_ERROR;
482     }
483     sqlite3_finalize(stmt);
484     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
485     return OC_STACK_OK;
486 }
487
488 OCStackResult PDMDeleteDevice(const OicUuid_t *UUID)
489 {
490     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
491
492     CHECK_PDM_INIT(TAG);
493     if (NULL == UUID)
494     {
495         return OC_STACK_INVALID_PARAM;
496     }
497     int id = 0;
498     if (OC_STACK_OK != getIdForUUID(UUID, &id))
499     {
500         OIC_LOG(ERROR, TAG, "Requested value not found");
501         return OC_STACK_INVALID_PARAM;
502     }
503     begin();
504     if(OC_STACK_OK != removeFromDeviceList(id))
505     {
506         rollback();
507         OIC_LOG(ERROR, TAG, "Requested value not found");
508         return OC_STACK_ERROR;
509     }
510     commit();
511     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
512     return OC_STACK_OK;
513 }
514
515
516 static OCStackResult updateLinkState(int id1, int id2, int state)
517 {
518     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

```

```

519
520     sqlite3_stmt *stmt = 0;
521     int res = 0 ;
522     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_UPDATE_LINK,
523                             strlen(PDM_SQLITE_UPDATE_LINK) + 1, &stmt, NULL);
524     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
525
526     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, state);
527     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
528
529     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id1);
530     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
531
532     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_THIRD, id2);
533     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
534
535     if (SQLITE_DONE != sqlite3_step(stmt))
536     {
537         OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
538         sqlite3_finalize(stmt);
539         return OC_STACK_ERROR;
540     }
541     sqlite3_finalize(stmt);
542     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
543     return OC_STACK_OK;
544 }
545
546 OCStackResult PDMSetLinkStale(const OicUuid_t* uuidOfDevice1, const OicUuid_t* uuidOfDevice2)
547 {
548     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
549
550     CHECK_PDM_INIT(TAG);
551     if (NULL == uuidOfDevice1 || NULL == uuidOfDevice2)
552     {
553         OIC_LOG(ERROR, TAG, "Invalid PARAM");
554         return OC_STACK_INVALID_PARAM;
555     }
556
557     int id1 = 0;
558     if (OC_STACK_OK != getIdForUUID(uuidOfDevice1, &id1))
559     {
560         OIC_LOG(ERROR, TAG, "Requested value not found");
561         return OC_STACK_INVALID_PARAM;
562     }
563
564     int id2 = 0;
565     if (OC_STACK_OK != getIdForUUID(uuidOfDevice2, &id2))
566     {
567         OIC_LOG(ERROR, TAG, "Requested value not found");

```

```

568         return OC_STACK_INVALID_PARAM;
569     }
570     ASCENDING_ORDER(id1, id2);
571     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
572     return updateLinkState(id1, id2, PDM_DEVICE_STALE);
573 }
574
575 OCStackResult PDMGetOwnedDevices(OCUuidList_t **uuidList, size_t *numOfDevices)
576 {
577     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
578
579     CHECK_PDM_INIT(TAG);
580     if (NULL != *uuidList)
581     {
582         OIC_LOG(ERROR, TAG, "Not null list will cause memory leak");
583         return OC_STACK_INVALID_PARAM;
584     }
585     sqlite3_stmt *stmt = 0;
586     int res = 0;
587     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_LIST_ALL_UUID,
588                             strlen(PDM_SQLITE_LIST_ALL_UUID) + 1, &stmt, NULL);
589     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
590
591     size_t counter = 0;
592     while (SQLITE_ROW == sqlite3_step(stmt))
593     {
594         const void *ptr = sqlite3_column_blob(stmt, PDM_FIRST_INDEX);
595         OicUuid_t *uid = (OicUuid_t *)ptr;
596         OCUuidList_t *temp = (OCUuidList_t *) OICMalloc(1, sizeof(OCUuidList_t));
597         if (NULL == temp)
598         {
599             OIC_LOG_V(ERROR, TAG, "Memory allocation problem");
600             sqlite3_finalize(stmt);
601             return OC_STACK_NO_MEMORY;
602         }
603         memcpy(&temp->dev.id, uid->id, UUID_LENGTH);
604         LL_PREPEND(*uuidList, temp);
605         ++counter;
606     }
607     *numOfDevices = counter;
608     sqlite3_finalize(stmt);
609     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
610     return OC_STACK_OK;
611 }
612
613 static OCStackResult getUUIDforId(int id, OicUuid_t *uid, bool *result)
614 {
615     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
616

```

```

617     sqlite3_stmt *stmt = 0;
618     int res = 0;
619     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_UUID,
620                             strlen(PDM_SQLITE_GET_UUID) + 1, &stmt, NULL);
621     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
622
623     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id);
624     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
625
626     while (SQLITE_ROW == sqlite3_step(stmt))
627     {
628         const void *ptr = sqlite3_column_blob(stmt, PDM_FIRST_INDEX);
629         memcpy(uid, ptr, sizeof(OicUuid_t));
630
631         int temp = sqlite3_column_int(stmt, PDM_SECOND_INDEX);
632         if(PDM_DEVICE_STALE == temp)
633         {
634             if(result)
635             {
636                 *result = true;
637             }
638         }
639         else
640         {
641             if(result)
642             {
643                 *result = false;
644             }
645         }
646         sqlite3_finalize(stmt);
647         return OC_STACK_OK;
648     }
649     sqlite3_finalize(stmt);
650     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
651     return OC_STACK_INVALID_PARAM;
652 }
653
654 OCStackResult PDMGetLinkedDevices(const OicUuid_t *UUID, OCUuidList_t **UUIDLIST, size_t *numOfDev
655 {
656     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
657
658     CHECK_PDM_INIT(TAG);
659     if (NULL == UUID || NULL == numOfDevices || !UUIDLIST)
660     {
661         return OC_STACK_INVALID_PARAM;
662     }
663     if (NULL != *UUIDLIST )
664     {
665         OIC_LOG(ERROR, TAG, "Not null list will cause memory leak");

```

```

666         return OC_STACK_INVALID_PARAM;
667     }
668     PdmDeviceState_t state = PDM_DEVICE_UNKNOWN;
669     OCStackResult ret = PDMGetDeviceState(UUID, &state);
670     if (OC_STACK_OK != ret)
671     {
672         OIC_LOG(ERROR, TAG, "Internal error occurred");
673         return OC_STACK_ERROR;
674     }
675     if (PDM_DEVICE_ACTIVE != state)
676     {
677         OIC_LOG_V(ERROR, TAG, "Device state is not active : %d", state);
678         return OC_STACK_INVALID_PARAM;
679     }
680     int id = 0;
681     if (OC_STACK_OK != getIdForUUID(UUID, &id))
682     {
683         OIC_LOG(ERROR, TAG, "Requested value not found");
684         return OC_STACK_INVALID_PARAM;
685     }
686
687
688     sqlite3_stmt *stmt = 0;
689     int res = 0;
690     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_LINKED_DEVICES,
691                             strlen(PDM_SQLITE_GET_LINKED_DEVICES) + 1, &stmt, NULL);
692     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
693
694     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id);
695     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
696
697     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id);
698     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
699
700     size_t counter = 0;
701     while (SQLITE_ROW == sqlite3_step(stmt))
702     {
703         int i1 = sqlite3_column_int(stmt, PDM_FIRST_INDEX);
704         int i2 = sqlite3_column_int(stmt, PDM_SECOND_INDEX);
705
706         OicUuid_t temp = {{0,}};
707         if (i1 != id)
708         {
709             getUUIDforId(i1, &temp, NULL);
710         }
711         if (i2 != id)
712         {
713             getUUIDforId(i2, &temp, NULL);
714         }

```

```

715
716     OCUuidList_t *tempNode = (OCUuidList_t *) OIAlloc(1, sizeof(OCUuidList_t));
717     if (NULL == tempNode)
718     {
719         OIC_LOG(ERROR, TAG, "No Memory");
720         sqlite3_finalize(stmt);
721         return OC_STACK_NO_MEMORY;
722     }
723     memcpy(&tempNode->dev.id, &temp.id, UUID_LENGTH);
724     LL_PREPEND(*UUIDLIST, tempNode);
725     ++counter;
726 }
727 *numOfDevices = counter;
728 sqlite3_finalize(stmt);
729 OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
730 return OC_STACK_OK;
731 }
732
733 OCStackResult PDMGetToBeUnlinkedDevices(OCPairList_t **staleDevList, size_t *numOfDevices)
734 {
735     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
736
737     CHECK_PDM_INIT(TAG);
738     if (NULL != *staleDevList)
739     {
740         OIC_LOG(ERROR, TAG, "Not null list will cause memory leak");
741         return OC_STACK_INVALID_PARAM;
742     }
743
744     sqlite3_stmt *stmt = 0;
745     int res = 0;
746     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_STALE_INFO,
747                             strlen(PDM_SQLITE_GET_STALE_INFO) + 1, &stmt, NULL);
748     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
749
750     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, PDM_DEVICE_STALE);
751     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
752
753     size_t counter = 0;
754     while (SQLITE_ROW == sqlite3_step(stmt))
755     {
756         int i1 = sqlite3_column_int(stmt, PDM_FIRST_INDEX);
757         int i2 = sqlite3_column_int(stmt, PDM_SECOND_INDEX);
758         OicUuid_t temp1 = {{0,}};
759         OicUuid_t temp2 = {{0,}};
760         getUUIDforId(i1, &temp1, NULL);
761         getUUIDforId(i2, &temp2, NULL);
762
763         OCPairList_t *tempNode = (OCPairList_t *) OIAlloc(1, sizeof(OCPairList_t));

```



```

764         if (NULL == tempNode)
765         {
766             OIC_LOG(ERROR, TAG, "No Memory");
767             sqlite3_finalize(stmt);
768             return OC_STACK_NO_MEMORY;
769         }
770         memcpy(&tempNode->dev.id, &temp1.id, UUID_LENGTH);
771         memcpy(&tempNode->dev2.id, &temp2.id, UUID_LENGTH);
772         LL_PREPEND(*staleDevList, tempNode);
773         ++counter;
774     }
775     *numOfDevices = counter;
776     sqlite3_finalize(stmt);
777     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
778     return OC_STACK_OK;
779 }
780
781 OCStackResult PDMClose()
782 {
783     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
784
785     CHECK_PDM_INIT(TAG);
786     int res = 0;
787     res = sqlite3_close(g_db);
788     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
789     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
790     return OC_STACK_OK;
791 }
792
793 void PDMDestroyOicUuidLinkList(OCUuidList_t* ptr)
794 {
795     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
796
797     if(ptr)
798     {
799         OCUuidList_t *tmp1 = NULL, *tmp2=NULL;
800         LL_FOREACH_SAFE(ptr, tmp1, tmp2)
801         {
802             LL_DELETE(ptr, tmp1);
803             OICFree(tmp1);
804         }
805     }
806
807     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
808 }
809
810 void PDMDestroyStaleLinkList(OCPairList_t* ptr)
811 {
812     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

```

```

813
814     if(ptr)
815     {
816         OCPairList_t *tmp1 = NULL,*tmp2=NULL;
817         LL_FOREACH_SAFE(ptr, tmp1, tmp2)
818         {
819             LL_DELETE(ptr, tmp1);
820             OICFree(tmp1);
821         }
822     }
823
824     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
825 }
826
827 OCStackResult PDMIsLinkExists(const OicUuid_t* uuidOfDevice1, const OicUuid_t* uuidOfDevice2,
828                               bool* result)
829 {
830     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
831
832     CHECK_PDM_INIT(TAG);
833     if (NULL == uuidOfDevice1 || NULL == uuidOfDevice2 || NULL == result)
834     {
835         return OC_STACK_INVALID_PARAM;
836     }
837     int id1 = 0;
838     int id2 = 0;
839     if (OC_STACK_OK != getIdForUUID(uuidOfDevice1, &id1))
840     {
841         OIC_LOG(ERROR, TAG, "Requested value not found");
842         return OC_STACK_INVALID_PARAM;
843     }
844
845     if (OC_STACK_OK != getIdForUUID(uuidOfDevice2, &id2))
846     {
847         OIC_LOG(ERROR, TAG, "Requested value not found");
848         return OC_STACK_INVALID_PARAM;
849     }
850
851     PdmDeviceState_t state = PDM_DEVICE_UNKNOWN;
852     if (OC_STACK_OK != PDMGetDeviceState(uuidOfDevice1, &state))
853     {
854         OIC_LOG(ERROR, TAG, "uuidOfDevice1:Internal error occured");
855         return OC_STACK_ERROR;
856     }
857     if (PDM_DEVICE_ACTIVE != state)
858     {
859         OIC_LOG_V(ERROR, TAG, "uuidOfDevice1:Device state is not active : %d", state);
860         return OC_STACK_INVALID_PARAM;
861     }

```

```

862
863     state = PDM_DEVICE_UNKNOWN;
864     if (OC_STACK_OK != PDMGetDeviceState(uuidOfDevice2, &state))
865     {
866         OIC_LOG(ERROR, TAG, "uuidOfDevice2:Internal error occured");
867         return OC_STACK_ERROR;
868     }
869     if (PDM_DEVICE_ACTIVE != state)
870     {
871         OIC_LOG_V(ERROR, TAG, "uuidOfDevice2:Device state is not active : %d", state);
872         return OC_STACK_INVALID_PARAM;
873     }
874
875     ASCENDING_ORDER(id1, id2);
876
877     sqlite3_stmt *stmt = 0;
878     int res = 0;
879     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_DEVICE_LINKS,
880                             strlen(PDM_SQLITE_GET_DEVICE_LINKS) + 1, &stmt, NULL);
881     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
882
883     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id1);
884     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
885
886     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id2);
887     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
888
889     bool ret = false;
890     while(SQLITE_ROW == sqlite3_step(stmt))
891     {
892         OIC_LOG(INFO, TAG, "Link already exists between devices");
893         ret = true;
894     }
895     sqlite3_finalize(stmt);
896     *result = ret;
897     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
898     return OC_STACK_OK;
899 }
900
901 static OCStackResult updateDeviceState(const OicUuid_t *uuid, PdmDeviceState_t state)
902 {
903     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
904
905     sqlite3_stmt *stmt = 0;
906     int res = 0 ;
907     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_UPDATE_DEVICE,
908                             strlen(PDM_SQLITE_UPDATE_DEVICE) + 1, &stmt, NULL);
909     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
910

```

```

911     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, state);
912     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
913
914     res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_SECOND, uuid, UUID_LENGTH, SQLITE_STATIC);
915     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
916
917     if (SQLITE_DONE != sqlite3_step(stmt))
918     {
919         OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
920         sqlite3_finalize(stmt);
921         return OC_STACK_ERROR;
922     }
923     sqlite3_finalize(stmt);
924     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
925     return OC_STACK_OK;
926 }
927
928 static OCStackResult updateLinkForStaleDevice(const OicUuid_t *devUuid)
929 {
930     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
931
932     sqlite3_stmt *stmt = 0;
933     int res = 0 ;
934
935     int id = 0;
936     if (OC_STACK_OK != getIdForUUID(devUuid, &id))
937     {
938         OIC_LOG(ERROR, TAG, "Requested value not found");
939         return OC_STACK_INVALID_PARAM;
940     }
941
942     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_UPDATE_LINK_STALE_FOR_STALE_DEVICE,
943                             strlen(PDM_SQLITE_UPDATE_LINK_STALE_FOR_STALE_DEVICE) + 1,
944                             &stmt, NULL);
945     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
946
947     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id);
948     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
949
950     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id);
951     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
952
953     if (SQLITE_DONE != sqlite3_step(stmt))
954     {
955         OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
956         sqlite3_finalize(stmt);
957         return OC_STACK_ERROR;
958     }
959     sqlite3_finalize(stmt);

```

```

960     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
961     return OC_STACK_OK;
962 }
963
964 OCStackResult PDMSetDeviceState(const OicUuid_t* uuid, PdmDeviceState_t state)
965 {
966     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
967
968     OCStackResult res = OC_STACK_ERROR;
969
970     CHECK_PDM_INIT(TAG);
971     if (NULL == uuid)
972     {
973         OIC_LOG(ERROR, TAG, "Invalid PARAM");
974         return OC_STACK_INVALID_PARAM;
975     }
976     begin();
977
978     if(PDM_DEVICE_STALE == state)
979     {
980         res = updateLinkForStaleDevice(uuid);
981         if (OC_STACK_OK != res)
982         {
983             rollback();
984             OIC_LOG(ERROR, TAG, "unable to update links");
985             return res;
986         }
987     }
988
989     res = updateDeviceState(uuid, state);
990     if (OC_STACK_OK != res)
991     {
992         rollback();
993         OIC_LOG(ERROR, TAG, "unable to update device state");
994         return res;
995     }
996     commit();
997     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
998     return OC_STACK_OK;
999 }
1000
1001 OCStackResult PDMGetDeviceState(const OicUuid_t *uuid, PdmDeviceState_t* result)
1002 {
1003     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
1004
1005     if (NULL == uuid || NULL == result)
1006     {
1007         OIC_LOG(ERROR, TAG, "UUID or result is NULL");
1008         return OC_STACK_INVALID_PARAM;

```

```

1009     }
1010
1011     sqlite3_stmt *stmt = 0;
1012     int res = 0;
1013     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_DEVICE_STATUS, strlen(PDM_SQLITE_GET_DEVICE_STAT
1014                               &stmt, NULL);
1015     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
1016
1017     res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_FIRST, uuid, UUID_LENGTH, SQLITE_STATIC);
1018     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
1019
1020     *result = PDM_DEVICE_UNKNOWN;
1021     while(SQLITE_ROW == sqlite3_step(stmt))
1022     {
1023         int tempStaleStateFromDb = sqlite3_column_int(stmt, PDM_FIRST_INDEX);
1024         OIC_LOG_V(DEBUG, TAG, "Device state is %d", tempStaleStateFromDb);
1025         *result = (PdmDeviceState_t)tempStaleStateFromDb;
1026     }
1027     sqlite3_finalize(stmt);
1028     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
1029     return OC_STACK_OK;
1030 }
1031
1032 OCStackResult PDMDeleteDeviceWithState(const PdmDeviceState_t state)
1033 {
1034     OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);
1035
1036     CHECK_PDM_INIT(TAG);
1037     if (PDM_DEVICE_ACTIVE != state && PDM_DEVICE_STALE != state &&
1038         PDM_DEVICE_INIT != state && PDM_DEVICE_UNKNOWN != state)
1039     {
1040         return OC_STACK_INVALID_PARAM;
1041     }
1042
1043     sqlite3_stmt *stmt = 0;
1044     int res = 0;
1045     res = sqlite3_prepare_v2(g_db, PDM_SQLITE_DELETE_DEVICE_WITH_STATE,
1046                               strlen(PDM_SQLITE_DELETE_DEVICE_WITH_STATE) + 1, &stmt, NULL);
1047     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
1048
1049     res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, state);
1050     PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
1051
1052     if (SQLITE_DONE != sqlite3_step(stmt))
1053     {
1054         OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
1055         sqlite3_finalize(stmt);
1056         return OC_STACK_ERROR;
1057     }

```

```
1058     sqlite3_finalize(stmt);
1059     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
1060     return OC_STACK_OK;
1061 }
```