Samsung / TizenRT Public

Code Issues 24 Pull requests 49 Actions Projects Wiki ...

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

```
/* *****************************************************************
 *
 * Copyright 2015 Samsung Electronics All Rights Reserved.
 *
 *
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * *****************************************************************/

#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>

#include "sqlite3.h"
#include "logger.h"
#include "oic_malloc.h"
```

```c
#include "provisioningdatabasemanager.h"
#include "pmutility.h"
#include "oic_string.h"
#include "utlist.h"


#define DB_FILE "PDM.db"

#define TAG "OIC_PDM"

#define PDM_FIRST_INDEX 0
#define PDM_SECOND_INDEX 1

#define PDM_BIND_INDEX_FIRST 1
#define PDM_BIND_INDEX_SECOND 2
#define PDM_BIND_INDEX_THIRD 3

#define PDM_CREATE_T_DEVICE_LIST "create table T_DEVICE_LIST(ID INTEGER PRIMARY KEY AUTOINCREMENT,
                                  UUID BLOB NOT NULL UNIQUE, STATE INT NOT NULL);"

#define PDM_CREATE_T_DEVICE_LINK  "create table T_DEVICE_LINK_STATE(ID INT NOT NULL, ID2 INT NOT \
                                   NULL,STATE INT NOT NULL, PRIMARY KEY (ID, ID2));"
/**
 * Macro to verify sqlite success.
 * eg: VERIFY_NON_NULL(TAG, ptrData, ERROR,OC_STACK_ERROR);
 */
#define PDM_VERIFY_SQLITE_OK(tag, arg, logLevel, retValue) do{ if (SQLITE_OK != (arg)) \
            { OIC_LOG_V((logLevel), tag, "Error in " #arg ", Error Message: %s", \
               sqlite3_errmsg(g_db)); return retValue; }}while(0)

#define PDM_SQLITE_TRANSACTION_BEGIN "BEGIN TRANSACTION;"
#define PDM_SQLITE_TRANSACTION_COMMIT "COMMIT;"
#define PDM_SQLITE_TRANSACTION_ROLLBACK "ROLLBACK;"
#define PDM_SQLITE_GET_STALE_INFO "SELECT ID,ID2 FROM T_DEVICE_LINK_STATE WHERE STATE = ?"
#define PDM_SQLITE_INSERT_T_DEVICE_LIST "INSERT INTO T_DEVICE_LIST VALUES(?,?,?)"
#define PDM_SQLITE_GET_ID "SELECT ID FROM T_DEVICE_LIST WHERE UUID like ?"
#define PDM_SQLITE_INSERT_LINK_DATA "INSERT INTO T_DEVICE_LINK_STATE VALUES(?,?,?)"
#define PDM_SQLITE_DELETE_LINK "DELETE FROM T_DEVICE_LINK_STATE WHERE ID = ? and ID2 = ?"
#define PDM_SQLITE_DELETE_DEVICE_LINK "DELETE FROM T_DEVICE_LINK_STATE WHERE ID = ? or ID2 = ?"
#define PDM_SQLITE_DELETE_DEVICE "DELETE FROM T_DEVICE_LIST  WHERE ID = ?"
#define PDM_SQLITE_DELETE_DEVICE_WITH_STATE "DELETE FROM T_DEVICE_LIST  WHERE STATE= ?"
#define PDM_SQLITE_UPDATE_LINK "UPDATE T_DEVICE_LINK_STATE SET STATE = ?  WHERE ID = ? and ID2 = ?
#define PDM_SQLITE_LIST_ALL_UUID "SELECT UUID FROM T_DEVICE_LIST WHERE STATE = 0"
#define PDM_SQLITE_GET_UUID "SELECT UUID,STATE FROM T_DEVICE_LIST WHERE ID = ?"
#define PDM_SQLITE_GET_LINKED_DEVICES "SELECT ID,ID2 FROM T_DEVICE_LINK_STATE WHERE \
                                           (ID = ? or ID2 = ?) and state = 0"
#define PDM_SQLITE_GET_DEVICE_LINKS "SELECT ID,ID2 FROM T_DEVICE_LINK_STATE WHERE \
                                          ID = ? and ID2 = ? and state = 0"
#define PDM_SQLITE_UPDATE_DEVICE "UPDATE T_DEVICE_LIST SET STATE = ?  WHERE UUID like ?"
```

```c
#define PDM_SQLITE_GET_DEVICE_STATUS "SELECT STATE FROM T_DEVICE_LIST WHERE UUID like ?"
#define PDM_SQLITE_UPDATE_LINK_STALE_FOR_STALE_DEVICE "UPDATE T_DEVICE_LINK_STATE SET STATE = 1\
                                                          WHERE ID = ? or ID2 = ?"

#define ASCENDING_ORDER(id1, id2) do{if( (id1) > (id2) )\
  { int temp; temp = id1; id1 = id2; id2 = temp; }}while(0)

#define CHECK_PDM_INIT(tag) do{if(true != gInit)\
  { OIC_LOG(ERROR, (tag), "PDB is not initialized"); \
    return OC_STACK_PDM_IS_NOT_INITIALIZED; }}while(0)

static sqlite3 *g_db = NULL;
static bool gInit = false;  /* Only if we can open sqlite db successfully, gInit is true. */

/**
 * function to create DB in case DB doesn't exists
 */
static OCStackResult createDB(const char* path)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    int result = 0;
    result = sqlite3_open_v2(path, &g_db, SQLITE_OPEN_READWRITE|SQLITE_OPEN_CREATE, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, result, ERROR, OC_STACK_ERROR);

    result = sqlite3_exec(g_db, PDM_CREATE_T_DEVICE_LIST, NULL, NULL, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, result, ERROR, OC_STACK_ERROR);

    OIC_LOG(INFO, TAG, "Created T_DEVICE_LIST");
    result = sqlite3_exec(g_db, PDM_CREATE_T_DEVICE_LINK, NULL, NULL, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, result, ERROR, OC_STACK_ERROR);

    OIC_LOG(INFO, TAG, "Created T_DEVICE_LINK_STATE");
    gInit = true;

    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);

    return OC_STACK_OK;
}


/**
 * Function to begin any transaction
 */
static OCStackResult begin()
{
    int res = 0;
    res = sqlite3_exec(g_db, PDM_SQLITE_TRANSACTION_BEGIN, NULL, NULL, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
    return OC_STACK_OK;
}

/**
 * Function to commit any transaction
 */
static OCStackResult commit()
{
    int res = 0;
    res = sqlite3_exec(g_db, PDM_SQLITE_TRANSACTION_COMMIT, NULL, NULL, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
    return OC_STACK_OK;
}

/**
 * Function to rollback any transaction
 */
static OCStackResult rollback()
{
    int res = 0;
    res = sqlite3_exec(g_db, PDM_SQLITE_TRANSACTION_ROLLBACK, NULL, NULL, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
    return OC_STACK_OK;
}

/**
 * Error log callback called by SQLite stack in case of error
 */
void errLogCallback(void *pArg, int iErrCode, const char *zMsg)
{
    (void) pArg;
    (void) iErrCode;
    (void) zMsg;
    OIC_LOG_V(DEBUG,TAG, "%s : (%d) %s", __func__, iErrCode, zMsg);
}

OCStackResult PDMInit(const char *path)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    int rc;
    const char *dbPath = NULL;
    if (SQLITE_OK !=  sqlite3_config(SQLITE_CONFIG_LOG, errLogCallback, NULL))
    {
        OIC_LOG(INFO, TAG, "Unable to enable debug log of sqlite");
    }

    if (NULL == path || !*path)
    {
        dbPath = DB_FILE;
    }
    else
    {
        dbPath = path;
    }
    rc = sqlite3_open_v2(dbPath, &g_db, SQLITE_OPEN_READWRITE, NULL);
    if (SQLITE_OK != rc)
    {
        OIC_LOG_V(INFO, TAG, "ERROR: Can't open database: %s", sqlite3_errmsg(g_db));
        sqlite3_close(g_db);
        OCStackResult ret = createDB(dbPath);
        if (OC_STACK_OK != ret)
        {
            sqlite3_close(g_db);
        }
        return ret;
    }
    gInit = true;

    /*
     * Remove PDM_DEVICE_INIT status devices.
     * PDM_DEVICE_INIT means that the OTM process is in progress.
     * PDM_DEVICE_INIT state device can be existed when the program is terminated during the OTM p
     * For this reason, PDM_DEVICE_INIT devices should be removed at PDM initialization time.
     */
    if(OC_STACK_OK != PDMDeleteDeviceWithState(PDM_DEVICE_INIT))
    {
        OIC_LOG_V(WARNING, TAG, "Failed to delete init state devices.");
    }

    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);

    return OC_STACK_OK;
}


OCStackResult PDMAddDevice(const OicUuid_t *UUID)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL == UUID)
    {
        return OC_STACK_INVALID_PARAM;
    }

    sqlite3_stmt *stmt = 0;
    int res =0;
```

```c
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_INSERT_T_DEVICE_LIST,
                              strlen(PDM_SQLITE_INSERT_T_DEVICE_LIST) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_SECOND, UUID, UUID_LENGTH, SQLITE_STATIC);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_THIRD, PDM_DEVICE_INIT);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_step(stmt);
    if (SQLITE_DONE != res)
    {
        if (SQLITE_CONSTRAINT == res)
        {
            //new OCStack result code
            OIC_LOG_V(ERROR, TAG, "Error Occured: %s",sqlite3_errmsg(g_db));
            sqlite3_finalize(stmt);
            return OC_STACK_DUPLICATE_UUID;
        }
        OIC_LOG_V(ERROR, TAG, "Error Occured: %s",sqlite3_errmsg(g_db));
        sqlite3_finalize(stmt);
        return OC_STACK_ERROR;
    }
    sqlite3_finalize(stmt);

    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

/**
 *function to get Id for given UUID
 */
static OCStackResult getIdForUUID(const OicUuid_t *UUID , int *id)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_ID, strlen(PDM_SQLITE_GET_ID) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_FIRST, UUID, UUID_LENGTH, SQLITE_STATIC);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    OIC_LOG(DEBUG, TAG, "Binding Done");
    while (SQLITE_ROW == sqlite3_step(stmt))
    {
        int tempId = sqlite3_column_int(stmt, PDM_FIRST_INDEX);
        OIC_LOG_V(DEBUG, TAG, "ID is %d", tempId);
        *id = tempId;
        sqlite3_finalize(stmt);
        OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
        return OC_STACK_OK;
    }
    sqlite3_finalize(stmt);
    return OC_STACK_INVALID_PARAM;
}

/**
 * Function to check duplication of device's Device ID.
 */
OCStackResult PDMIsDuplicateDevice(const OicUuid_t* UUID, bool *result)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL == UUID || NULL == result)
    {
        OIC_LOG(ERROR, TAG, "UUID or result is NULL");
        return OC_STACK_INVALID_PARAM;
    }
    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_ID, strlen(PDM_SQLITE_GET_ID) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_FIRST, UUID, UUID_LENGTH, SQLITE_STATIC);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    OIC_LOG(DEBUG, TAG, "Binding Done");
    bool retValue = false;
    while(SQLITE_ROW == sqlite3_step(stmt))
    {
        OIC_LOG(INFO, TAG, "Duplicated UUID");
        retValue = true;
    }

    sqlite3_finalize(stmt);
    *result = retValue;

    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

/**
 * Function to add link in sqlite
 */
```

```c
static OCStackResult addlink(int id1, int id2)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_INSERT_LINK_DATA,
                              strlen(PDM_SQLITE_INSERT_LINK_DATA) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id1);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id2);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_THIRD, PDM_DEVICE_ACTIVE);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    if (sqlite3_step(stmt) != SQLITE_DONE)
    {
        OIC_LOG_V(ERROR, TAG, "Error Occured: %s",sqlite3_errmsg(g_db));
        sqlite3_finalize(stmt);
        return OC_STACK_ERROR;
    }
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

OCStackResult PDMLinkDevices(const OicUuid_t *UUID1, const OicUuid_t *UUID2)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL == UUID1 || NULL == UUID2)
    {
        OIC_LOG(ERROR, TAG, "Invalid PARAM");
        return  OC_STACK_INVALID_PARAM;
    }

    PdmDeviceState_t state = PDM_DEVICE_UNKNOWN;
    if (OC_STACK_OK != PDMGetDeviceState(UUID1, &state))
    {
        OIC_LOG(ERROR, TAG, "Internal error occured");
        return OC_STACK_ERROR;
    }
    if (PDM_DEVICE_ACTIVE != state)
    {
        OIC_LOG_V(ERROR, TAG, "UUID1: Device state is not active : %d", state);
        return OC_STACK_INVALID_PARAM;
    }

    state = PDM_DEVICE_UNKNOWN;
    if (OC_STACK_OK != PDMGetDeviceState(UUID2, &state))
    {
        OIC_LOG(ERROR, TAG, "Internal error occured");
        return OC_STACK_ERROR;
    }
    if (PDM_DEVICE_ACTIVE != state)
    {
        OIC_LOG_V(ERROR, TAG, "UUID2: Device state is not active : %d", state);
        return OC_STACK_INVALID_PARAM;
    }

    int id1 = 0;
    if (OC_STACK_OK != getIdForUUID(UUID1, &id1))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }
    int id2 = 0;
    if (OC_STACK_OK != getIdForUUID(UUID2, &id2))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }

    ASCENDING_ORDER(id1, id2);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return addlink(id1, id2);
}

/**
 * Function to remove created link
 */
static OCStackResult removeLink(int id1, int id2)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    int res = 0;
    sqlite3_stmt *stmt = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_DELETE_LINK, strlen(PDM_SQLITE_DELETE_LINK) + 1, &st
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id1);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id2);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    if (SQLITE_DONE != sqlite3_step(stmt))
    {
        OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
        sqlite3_finalize(stmt);
        return OC_STACK_ERROR;
    }
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

OCStackResult PDMUnlinkDevices(const OicUuid_t *UUID1, const OicUuid_t *UUID2)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL == UUID1 || NULL == UUID2)
    {
        OIC_LOG(ERROR, TAG, "Invalid PARAM");
        return  OC_STACK_INVALID_PARAM;
    }

    int id1 = 0;
    if (OC_STACK_OK != getIdForUUID(UUID1, &id1))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }

    int id2 = 0;
    if (OC_STACK_OK != getIdForUUID(UUID2, &id2))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }
    ASCENDING_ORDER(id1, id2);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return removeLink(id1, id2);
}

static OCStackResult removeFromDeviceList(int id)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_DELETE_DEVICE,
                                  strlen(PDM_SQLITE_DELETE_DEVICE) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    if (sqlite3_step(stmt) != SQLITE_DONE)
    {
        OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
        sqlite3_finalize(stmt);
        return OC_STACK_ERROR;
    }
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

OCStackResult PDMDeleteDevice(const OicUuid_t *UUID)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL == UUID)
    {
        return OC_STACK_INVALID_PARAM;
    }
    int id = 0;
    if (OC_STACK_OK != getIdForUUID(UUID, &id))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }
    begin();
    if(OC_STACK_OK != removeFromDeviceList(id))
    {
        rollback();
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_ERROR;
    }
    commit();
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}


static OCStackResult updateLinkState(int id1, int id2, int state)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    sqlite3_stmt *stmt = 0;
    int res = 0 ;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_UPDATE_LINK,
                              strlen(PDM_SQLITE_UPDATE_LINK) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, state);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id1);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_THIRD, id2);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    if (SQLITE_DONE != sqlite3_step(stmt))
    {
        OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
        sqlite3_finalize(stmt);
        return OC_STACK_ERROR;
    }
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

OCStackResult PDMSetLinkStale(const OicUuid_t* uuidOfDevice1, const OicUuid_t* uuidOfDevice2)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL == uuidOfDevice1 || NULL == uuidOfDevice2)
    {
        OIC_LOG(ERROR, TAG, "Invalid PARAM");
        return  OC_STACK_INVALID_PARAM;
    }

    int id1 = 0;
    if (OC_STACK_OK != getIdForUUID(uuidOfDevice1, &id1))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }

    int id2 = 0;
    if (OC_STACK_OK != getIdForUUID(uuidOfDevice2, &id2))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }
    ASCENDING_ORDER(id1, id2);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return updateLinkState(id1, id2, PDM_DEVICE_STALE);
}

OCStackResult PDMGetOwnedDevices(OCUuidList_t **uuidList, size_t *numOfDevices)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL != *uuidList)
    {
        OIC_LOG(ERROR, TAG, "Not null list will cause memory leak");
        return OC_STACK_INVALID_PARAM;
    }
    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_LIST_ALL_UUID,
                              strlen(PDM_SQLITE_LIST_ALL_UUID) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    size_t counter  = 0;
    while (SQLITE_ROW == sqlite3_step(stmt))
    {
        const void *ptr = sqlite3_column_blob(stmt, PDM_FIRST_INDEX);
        OicUuid_t *uid = (OicUuid_t *)ptr;
        OCUuidList_t *temp = (OCUuidList_t *) OICCalloc(1,sizeof(OCUuidList_t));
        if (NULL == temp)
        {
            OIC_LOG_V(ERROR, TAG, "Memory allocation problem");
            sqlite3_finalize(stmt);
            return OC_STACK_NO_MEMORY;
        }
        memcpy(&temp->dev.id, uid->id, UUID_LENGTH);
        LL_PREPEND(*uuidList,temp);
        ++counter;
    }
    *numOfDevices = counter;
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

static OCStackResult getUUIDforId(int id, OicUuid_t *uid, bool *result)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_UUID,
                              strlen(PDM_SQLITE_GET_UUID) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    while (SQLITE_ROW == sqlite3_step(stmt))
    {
        const void *ptr = sqlite3_column_blob(stmt, PDM_FIRST_INDEX);
        memcpy(uid, ptr, sizeof(OicUuid_t));

        int temp = sqlite3_column_int(stmt, PDM_SECOND_INDEX);
        if(PDM_DEVICE_STALE == temp)
        {
            if(result)
            {
                *result = true;
            }
        }
        else
        {
            if(result)
            {
                *result = false;
            }
        }
        sqlite3_finalize(stmt);
        return OC_STACK_OK;
    }
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_INVALID_PARAM;
}

OCStackResult PDMGetLinkedDevices(const OicUuid_t *UUID, OCUuidList_t **UUIDLIST, size_t *numOfDev
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL == UUID || NULL == numOfDevices || !UUIDLIST)
    {
        return OC_STACK_INVALID_PARAM;
    }
    if (NULL != *UUIDLIST )
    {
        OIC_LOG(ERROR, TAG, "Not null list will cause memory leak");
```

```c
        return OC_STACK_INVALID_PARAM;
    }
    PdmDeviceState_t state = PDM_DEVICE_UNKNOWN;
    OCStackResult ret = PDMGetDeviceState(UUID, &state);
    if (OC_STACK_OK != ret)
    {
        OIC_LOG(ERROR, TAG, "Internal error occured");
        return OC_STACK_ERROR;
    }
    if (PDM_DEVICE_ACTIVE != state)
    {
        OIC_LOG_V(ERROR, TAG, "Device state is not active : %d", state);
        return OC_STACK_INVALID_PARAM;
    }
    int id = 0;
    if (OC_STACK_OK != getIdForUUID(UUID, &id))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }


    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_LINKED_DEVICES,
                              strlen(PDM_SQLITE_GET_LINKED_DEVICES) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    size_t counter  = 0;
    while (SQLITE_ROW == sqlite3_step(stmt))
    {
        int i1 = sqlite3_column_int(stmt, PDM_FIRST_INDEX);
        int i2 = sqlite3_column_int(stmt, PDM_SECOND_INDEX);

        OicUuid_t temp = {{0,}};
        if (i1 != id)
        {
            getUUIDforId(i1, &temp, NULL);
        }
        if (i2 != id)
        {
            getUUIDforId(i2, &temp, NULL);
        }

        OCUuidList_t *tempNode = (OCUuidList_t *) OICCalloc(1,sizeof(OCUuidList_t));
        if (NULL == tempNode)
        {
            OIC_LOG(ERROR, TAG, "No Memory");
            sqlite3_finalize(stmt);
            return OC_STACK_NO_MEMORY;
        }
        memcpy(&tempNode->dev.id, &temp.id, UUID_LENGTH);
        LL_PREPEND(*UUIDLIST,tempNode);
        ++counter;
    }
    *numOfDevices = counter;
     sqlite3_finalize(stmt);
     OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
     return OC_STACK_OK;
}

OCStackResult PDMGetToBeUnlinkedDevices(OCPairList_t **staleDevList, size_t *numOfDevices)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL != *staleDevList)
    {
        OIC_LOG(ERROR, TAG, "Not null list will cause memory leak");
        return OC_STACK_INVALID_PARAM;
    }

    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_STALE_INFO,
                              strlen(PDM_SQLITE_GET_STALE_INFO) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, PDM_DEVICE_STALE);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    size_t counter  = 0;
    while (SQLITE_ROW == sqlite3_step(stmt))
    {
        int i1 = sqlite3_column_int(stmt, PDM_FIRST_INDEX);
        int i2 = sqlite3_column_int(stmt, PDM_SECOND_INDEX);
        OicUuid_t temp1 = {{0,}};
        OicUuid_t temp2 = {{0,}};;
        getUUIDforId(i1, &temp1, NULL);
        getUUIDforId(i2, &temp2, NULL);

        OCPairList_t *tempNode = (OCPairList_t *) OICCalloc(1, sizeof(OCPairList_t));
        if (NULL == tempNode)
        {
            OIC_LOG(ERROR, TAG, "No Memory");
            sqlite3_finalize(stmt);
            return OC_STACK_NO_MEMORY;
        }
        memcpy(&tempNode->dev.id, &temp1.id, UUID_LENGTH);
        memcpy(&tempNode->dev2.id, &temp2.id, UUID_LENGTH);
        LL_PREPEND(*staleDevList, tempNode);
        ++counter;
    }
    *numOfDevices = counter;
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

OCStackResult PDMClose()
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    int res = 0;
    res = sqlite3_close(g_db);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

void PDMDestoryOicUuidLinkList(OCUuidList_t* ptr)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    if(ptr)
    {
        OCUuidList_t *tmp1 = NULL,*tmp2=NULL;
        LL_FOREACH_SAFE(ptr, tmp1, tmp2)
        {
            LL_DELETE(ptr, tmp1);
            OICFree(tmp1);
        }
    }

    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
}

void PDMDestoryStaleLinkList(OCPairList_t* ptr)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    if(ptr)
    {
        OCPairList_t *tmp1 = NULL,*tmp2=NULL;
        LL_FOREACH_SAFE(ptr, tmp1, tmp2)
        {
            LL_DELETE(ptr, tmp1);
            OICFree(tmp1);
        }
    }

    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
}

OCStackResult PDMIsLinkExists(const OicUuid_t* uuidOfDevice1, const OicUuid_t* uuidOfDevice2,
                               bool* result)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (NULL == uuidOfDevice1 || NULL == uuidOfDevice2 || NULL == result)
    {
        return OC_STACK_INVALID_PARAM;
    }
    int id1 = 0;
    int id2 = 0;
    if (OC_STACK_OK != getIdForUUID(uuidOfDevice1, &id1))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }

    if (OC_STACK_OK != getIdForUUID(uuidOfDevice2, &id2))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }

    PdmDeviceState_t state = PDM_DEVICE_UNKNOWN;
    if (OC_STACK_OK != PDMGetDeviceState(uuidOfDevice1, &state))
    {
        OIC_LOG(ERROR, TAG, "uuidOfDevice1:Internal error occured");
        return OC_STACK_ERROR;
    }
    if (PDM_DEVICE_ACTIVE != state)
    {
        OIC_LOG_V(ERROR, TAG, "uuidOfDevice1:Device state is not active : %d", state);
        return OC_STACK_INVALID_PARAM;
    }
```

```c

    state = PDM_DEVICE_UNKNOWN;
    if (OC_STACK_OK != PDMGetDeviceState(uuidOfDevice2, &state))
    {
        OIC_LOG(ERROR, TAG, "uuidOfDevice2:Internal error occured");
        return OC_STACK_ERROR;
    }
    if (PDM_DEVICE_ACTIVE != state)
    {
        OIC_LOG_V(ERROR, TAG, "uuidOfDevice2:Device state is not active : %d", state);
        return OC_STACK_INVALID_PARAM;
    }

    ASCENDING_ORDER(id1, id2);

    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_DEVICE_LINKS,
                              strlen(PDM_SQLITE_GET_DEVICE_LINKS) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id1);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id2);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    bool ret = false;
    while(SQLITE_ROW == sqlite3_step(stmt))
    {
        OIC_LOG(INFO, TAG, "Link already exists between devices");
        ret = true;
    }
    sqlite3_finalize(stmt);
    *result = ret;
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

static OCStackResult updateDeviceState(const OicUuid_t *uuid, PdmDeviceState_t state)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    sqlite3_stmt *stmt = 0;
    int res = 0 ;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_UPDATE_DEVICE,
                              strlen(PDM_SQLITE_UPDATE_DEVICE) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, state);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_SECOND, uuid, UUID_LENGTH, SQLITE_STATIC);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    if (SQLITE_DONE != sqlite3_step(stmt))
    {
        OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
        sqlite3_finalize(stmt);
        return OC_STACK_ERROR;
    }
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

static OCStackResult updateLinkForStaleDevice(const OicUuid_t *devUuid)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    sqlite3_stmt *stmt = 0;
    int res = 0 ;

    int id = 0;
    if (OC_STACK_OK != getIdForUUID(devUuid, &id))
    {
        OIC_LOG(ERROR, TAG, "Requested value not found");
        return OC_STACK_INVALID_PARAM;
    }

    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_UPDATE_LINK_STALE_FOR_STALE_DEVICE,
                              strlen(PDM_SQLITE_UPDATE_LINK_STALE_FOR_STALE_DEVICE) + 1,
                               &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, id);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_SECOND, id);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    if (SQLITE_DONE != sqlite3_step(stmt))
    {
        OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
        sqlite3_finalize(stmt);
        return OC_STACK_ERROR;
    }
    sqlite3_finalize(stmt);
```

```c
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

OCStackResult PDMSetDeviceState(const OicUuid_t* uuid, PdmDeviceState_t state)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    OCStackResult res = OC_STACK_ERROR;

    CHECK_PDM_INIT(TAG);
    if (NULL == uuid)
    {
        OIC_LOG(ERROR, TAG, "Invalid PARAM");
        return  OC_STACK_INVALID_PARAM;
    }
    begin();

    if(PDM_DEVICE_STALE == state)
    {
        res = updateLinkForStaleDevice(uuid);
        if (OC_STACK_OK != res)
        {
            rollback();
            OIC_LOG(ERROR, TAG, "unable to update links");
            return res;
        }
    }

    res = updateDeviceState(uuid, state);
    if (OC_STACK_OK != res)
    {
        rollback();
        OIC_LOG(ERROR, TAG, "unable to update device state");
        return res;
    }
    commit();
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

OCStackResult PDMGetDeviceState(const OicUuid_t *uuid, PdmDeviceState_t* result)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    if (NULL == uuid || NULL == result)
    {
        OIC_LOG(ERROR, TAG, "UUID or result is NULL");
        return OC_STACK_INVALID_PARAM;
    }

    sqlite3_stmt *stmt = 0;
    int res = 0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_GET_DEVICE_STATUS, strlen(PDM_SQLITE_GET_DEVICE_STAT
                              &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_blob(stmt, PDM_BIND_INDEX_FIRST, uuid, UUID_LENGTH, SQLITE_STATIC);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    *result = PDM_DEVICE_UNKNOWN;
    while(SQLITE_ROW == sqlite3_step(stmt))
    {
        int tempStaleStateFromDb = sqlite3_column_int(stmt, PDM_FIRST_INDEX);
        OIC_LOG_V(DEBUG, TAG, "Device state is %d", tempStaleStateFromDb);
        *result = (PdmDeviceState_t)tempStaleStateFromDb;
    }
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}

OCStackResult PDMDeleteDeviceWithState(const PdmDeviceState_t state)
{
    OIC_LOG_V(DEBUG, TAG, "IN %s", __func__);

    CHECK_PDM_INIT(TAG);
    if (PDM_DEVICE_ACTIVE != state && PDM_DEVICE_STALE != state &&
        PDM_DEVICE_INIT != state && PDM_DEVICE_UNKNOWN != state)
    {
        return OC_STACK_INVALID_PARAM;
    }

    sqlite3_stmt *stmt = 0;
    int res =0;
    res = sqlite3_prepare_v2(g_db, PDM_SQLITE_DELETE_DEVICE_WITH_STATE,
                              strlen(PDM_SQLITE_DELETE_DEVICE_WITH_STATE) + 1, &stmt, NULL);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    res = sqlite3_bind_int(stmt, PDM_BIND_INDEX_FIRST, state);
    PDM_VERIFY_SQLITE_OK(TAG, res, ERROR, OC_STACK_ERROR);

    if (SQLITE_DONE != sqlite3_step(stmt))
    {
        OIC_LOG_V(ERROR, TAG, "Error message: %s", sqlite3_errmsg(g_db));
        sqlite3_finalize(stmt);
        return OC_STACK_ERROR;
    }
    sqlite3_finalize(stmt);
    OIC_LOG_V(DEBUG, TAG, "OUT %s", __func__);
    return OC_STACK_OK;
}
```