```
// FeliCaLibSample08a.cpp : コンソール アプリケーションのエントリ ポイントを定義します。
// 鍵のないサービスデータへの書き込みと、鍵のないサービスデータの読み込み
//  （ポーリングと相互認証は行いません）データをキーボードから入力する

#include "stdafx.h"
#include <cstdio>
#include <cstdlib>
#include <conio.h>            // getch()の定義

#include "felica.h"

int main(void);
void error_routine(void);
void print_vector(char* title, unsigned char* vector, int length);
void print_vector2(char* title, unsigned char* vector, int length);

int main(void)
{
    fprintf(stdout, "鍵のないサービスデータへの書き込みと、鍵のないサービスデータの読み込み¥n¥n");

    /* ライブラリの初期化 */
    if (!initialize_library()) {
        fprintf(stderr, "ライブラリの初期化に失敗しました。¥n¥n");
        return EXIT_FAILURE;
    }

    /* リーダ・ライタの自動認識とオープン */
    if (!open_reader_writer_auto()) {
        fprintf(stderr, "リーダ・ライタのオープンに失敗しました。¥n¥n");
        return EXIT_FAILURE;
    }

    /* 変数と定数設定 */
    // 第1引数とする構造体、ポーリングをするために必要な情報
    structure_polling polling;
    unsigned char system_code[2] = {0x00, 0x00};
    polling.system_code = system_code;
    polling.time_slot = 0x00;

    // 第2引数とする自然数、カードの数
    unsigned char number_of_cards = 0;

    // 第3引数とする構造体、ポーリングの時に取得したカードの情報
    structure_card_information card_information;
    unsigned char card_idm[8];    // 製造IDブロックの製造ID(IDm)
    unsigned char card_pmm[8];    // 製造IDブロックの製造パラメータ(PMm)
    card_information.card_idm = card_idm;
    card_information.card_pmm = card_pmm;
```

```
/* ポーリングとカード情報の取得（Pollingコマンド） */
if (!polling_and_get_card_information(&polling, &number_of_cards, &card_information)) {
    fprintf(stderr, "FeliCaカードが見つかりません。¥n¥n");
    error_routine();
    return EXIT_FAILURE;
}

// 第1引数となる構造体、データを書き込むために必要な情報
input_structure_write_block_without_encryption input_write_block_without_encryption;
input_write_block_without_encryption.card_idm = card_idm;
input_write_block_without_encryption.number_of_services = 1;
unsigned char service_code_list[2] = {0x09, 0x10};
input_write_block_without_encryption.service_code_list = service_code_list;
input_write_block_without_encryption.number_of_blocks = 1;

// サービス定義ブロックの5,6番目の2バイトで指定するユーザブロックのサイズ
unsigned char block_list[2] = {0x80, 0x02};
input_write_block_without_encryption.block_list = block_list;

// 書き込むデータ、一度に書き込めるブロック数255まで
/* 直接指定する場合は以下のようにできる
unsigned char write_block_data[16] = {0x73, 0x74, 0x75, 0x64, 0x65, 0x6e, 0x74, 0x49,
                                      0x44, 0x3a, 0x20, 0x61, 0x35, 0x30, 0x30, 0x39};
unsigned char write_block_data[16] = {'s', 't', 'u', 'd', 'e', 'n', 't', 'I',
    'D', ':', ' ', 'a', '0', '0', '0', '9'};
*/
unsigned char write_block_data[16];
char str[80];    // gets()の引数用、ヌルコードの分+1のサイズを指定
size_t len;
fprintf(stdout, "書き込むデータを入力してください(16バイト以下)：");
gets(str);       // chrp=gets(strp)、文字列の最後はヌルコード
while (len = strlen(str) > 16) {
    fprintf(stdout, "文字数オーバー、再度入力してください：");
    gets(str);
}

for (int i = 0; str[i] != '¥0'; i++)
    write_block_data[i] = str[i];

while (i < 16)
    write_block_data[i++] = ' '; // 文字数が少ない場合ごみを表示しない

input_write_block_without_encryption.block_data = write_block_data;

// 第2引数となる構造体、データを書き込んだときの情報
output_structure_write_block_without_encryption output_write_block_without_encryption;
unsigned char status_flag_1 = 0x00;
unsigned char status_flag_2 = 0x00;
output_write_block_without_encryption.status_flag_1 = &status_flag_1;
output_write_block_without_encryption.status_flag_2 = &status_flag_2;
```

```
// 書き込みの確認
fprintf(stdout, "書き込みを行います、Enterを押してください。¥n");
getch();

/* 鍵のないサービスデータへの書き込み. ポーリング行わない. 一度に書き込めるブロック数255まで */
if (!write_block_without_encryption(&input_write_block_without_encryption,
                                    &output_write_block_without_encryption)) {
    fprintf(stderr, "鍵のないサービスデータへの書き込みに失敗しました¥n¥n");
    error_routine();
    return EXIT_FAILURE;
}

/* 書き込んだデータの確認 */
fprintf(stdout, "書き込んだブロック数     : %d¥n",
                                input_write_block_without_encryption.number_of_blocks);
print_vector("書き込んだブロックデータ:", write_block_data, sizeof(write_block_data));
fprintf(stdout, "¥n");

// 第1引数となる構造体、データを読み込むために必要な情報
input_structure_read_block_without_encryption input_read_block_without_encryption;
input_read_block_without_encryption.card_idm = card_idm;
input_read_block_without_encryption.number_of_services = 1;
input_read_block_without_encryption.service_code_list = service_code_list;
input_read_block_without_encryption.number_of_blocks = 1;
input_read_block_without_encryption.block_list = block_list;

// 第2引数となる構造体、データを読み込んだときの情報
output_structure_read_block_without_encryption output_read_block_without_encryption;
status_flag_1 = 0x00;
status_flag_2 = 0x00;
output_read_block_without_encryption.status_flag_1 = &status_flag_1;
output_read_block_without_encryption.status_flag_2 = &status_flag_2;
unsigned char result_number_of_blocks = 0;
output_read_block_without_encryption.result_number_of_blocks = &result_number_of_blocks;
unsigned char read_block_data[16];
output_read_block_without_encryption.block_data = read_block_data;

// 読み込みの確認
fprintf(stdout, "読み込みを行います、Enterを押してください。¥n");
getch();

/* 鍵のないサービスデータの読み込み. ポーリング行わない. 一度に読み込めるブロック数255まで */
if (!read_block_without_encryption(&input_read_block_without_encryption,
                                   &output_read_block_without_encryption)) {
    fprintf(stderr, "鍵のないサービスデータの読み込みに失敗しました¥n¥n");
    error_routine();
    return EXIT_FAILURE;
}
```

```
    /* 取得データの表示 */
    fprintf(stdout, "読み込んだブロック数     : %d¥n", result_number_of_blocks);
    print_vector("読み込んだブロックデータ:", read_block_data, sizeof(read_block_data));
    print_vector2("読み込んだブロックデータ:", read_block_data, sizeof(read_block_data));

    /* リーダ・ライタのクローズ */
    if (!close_reader_writer()) {
        fprintf(stderr, "リーダ・ライタのクローズに失敗しました。¥n¥n");
        return EXIT_FAILURE;
    }

    /* ライブラリの解放 */
    if (!dispose_library()) {
        fprintf(stderr, "ライブラリの解放に失敗しました。¥n¥n");
        return EXIT_FAILURE;
    }

    fprintf(stdout, "¥nプログラムの実行を終了します。¥n¥n");
    return EXIT_SUCCESS; // 正常終了を示す戻り値
}

void error_routine(void)
{
    enumernation_felica_error_type felica_error_type;
    enumernation_rw_error_type rw_error_type;
    get_last_error_types(&felica_error_type, &rw_error_type);
    printf("felica_error_type: %d¥n", felica_error_type);
    printf("rw_error_type: %d¥n¥n", rw_error_type);

    close_reader_writer();
    dispose_library();
}

void print_vector(char* title, unsigned char* vector, int length)
{
    if (title != NULL) {
        fprintf(stdout, "%s ", title);
    }

    int i;
    for (i = 0; i < length - 1; i++) {
        fprintf(stdout, "%02x ", vector[i]);
    }
    fprintf(stdout, "%02x", vector[length - 1]);
    fprintf(stdout, "¥n");
}
```

```
void print_vector2(char* title, unsigned char* vector, int length)
{
    if (title != NULL) {
        fprintf(stdout, "%s ", title);
    }

    int i;
    for (i = 0; i < length - 1; i++) {
        fprintf(stdout, "%c", vector[i]);
    }
    fprintf(stdout, "%c", vector[length - 1]);
    fprintf(stdout, "¥n");
}
```