목차

BS2_EnrolUser

새로운 사용자를 등록합니다.

> **도움말**
> **FaceStation F2**로 사용자 전송을 위해서는 BS2_EnrollUserFaceEx를 사용해 주십시오.

함수

```
#include "BS_API.h"

int BS2_EnrolUser(void* context, uint32_t deviceId, BS2UserBlob* userBlob,
uint32_t userCount, uint8_t overwrite);
```

> BS2UserBlob 구조체 보기

파라미터

- [In] *context* : Context
- [In] *deviceId* : 장치 식별자
- [In] *userBlob* : 등록하고자 하는 사용자 정보
- [In] *userCount* : 사용자 개수
- [In] *overwrite* : 중복된 사용자 덮어쓰기 여부

반환값

성공적으로 수행될 경우 BS_SDK_SUCCESS를 반환하고, 에러가 발생할 경우 상응하는 에러 코드를 반환합니다.

C++

```
int enrollUser(BS2_DEVICE_ID id)
{
    BS2SimpleDeviceInfo deviceInfo = { , };
    BS2SimpleDeviceInfoEx deviceInfoEx = { , };
```

```
    int sdkResult = BS2_GetDeviceInfoEx(context_, id, &deviceInfo,
&deviceInfoEx);
    if (BS_SDK_SUCCESS != sdkResult)
    {
        TRACE("BS2_GetDeviceInfoEx call failed: %d", sdkResult);
        return sdkResult;
    }

    bool fingerScanSupported = (deviceInfoEx.supported &
BS2SimpleDeviceInfoEx::BS2_SUPPORT_FINGER_SCAN) ==
BS2SimpleDeviceInfoEx::BS2_SUPPORT_FINGER_SCAN;
    bool faceScanSupported = (deviceInfoEx.supported &
BS2SimpleDeviceInfoEx::BS2_SUPPORT_FACE_SCAN) ==
BS2SimpleDeviceInfoEx::BS2_SUPPORT_FACE_SCAN;
    bool qrSupported = (deviceInfoEx.supported &
BS2SimpleDeviceInfoEx::BS2_SUPPORT_QR) ==
BS2SimpleDeviceInfoEx::BS2_SUPPORT_QR;

    BS2UserBlob userBlob = { , };
    BS2User& user = userBlob.user;
    BS2UserSetting& setting = userBlob.setting;
    BS2UserPhoto& photo = userBlob.user_photo;
    stringstream msg;

    string uid = Utility::getInput<string>("Please enter a user ID:");
    if (BS2_USER_ID_SIZE < uid.size())
    {
        TRACE("User ID is too big.");
        return BS_SDK_ERROR_INVALID_PARAM;
    }
    strcpy(user.userID, uid.c_str());

    if (deviceInfo.userNameSupported)
    {
        string name = Utility::getInput<string>("Enter your name:");
        if (BS2_USER_NAME_SIZE < name.size())
        {
            TRACE("User name is too long.");
            return BS_SDK_ERROR_INVALID_PARAM;
        }
        strcpy(reinterpret_cast<char*>(userBlob.user_name), name.c_str());
    }

    {
        string inputTime = Utility::getLine("Please enter start time [YYYY-
MM-DD HH:MM:SS] ?");
        BS2_TIMESTAMP startTime = Utility::convertTimeString2UTC(inputTime);
        setting.startTime = startTime;

        inputTime = Utility::getLine("Please enter end time [YYYY-MM-DD
```

```
HH:MM:SS] ?");
        BS2_TIMESTAMP endTime = Utility::convertTimeString2UTC(inputTime);
        setting.endTime = endTime;
    }

    if (deviceInfo.pinSupported)
    {
        string pinString = Utility::getInput<string>("Enter the PIN code:");
        if (BS2_USER_PIN_SIZE < pinString.size())
        {
            TRACE("PIN code is too long");
            return BS_SDK_ERROR_INVALID_PARAM;
        }

        sdkResult = BS2_MakePinCode(context_,
const_cast<char*>(pinString.c_str()), userBlob.pin);
        if (BS_SDK_SUCCESS != sdkResult)
        {
            TRACE("BS2_MakePinCode call failed: %d", sdkResult);
            return sdkResult;
        }
    }

    setting.fingerAuthMode = BS2_AUTH_MODE_BIOMETRIC_ONLY;
    setting.cardAuthMode = BS2_AUTH_MODE_CARD_ONLY;
    setting.idAuthMode = BS2_AUTH_MODE_ID_PIN;
    setting.securityLevel = BS2_USER_SECURITY_LEVEL_DEFAULT;

    if (deviceInfo.userPhotoSupported)
    {
        if (Utility::isYes("Do you want to register a profile image?"))
        {
            string imagePath = Utility::getInput<string>("Enter the profile
image path and name:");
            uint32_t size = Utility::getResourceSize(imagePath);
            shared_ptr<uint8_t> buffer(new uint8_t[size],
ArrayDeleter<uint8_t>());

            while (BS2_USER_PHOTO_SIZE < size)
            {
                msg.str("");
                msg << "Image is to big.\n";
                msg << "Re-enter an image smaller than 16384 byte:";
                imagePath = Utility::getInput<string>(msg.str());
                size = Utility::getResourceSize(imagePath);
            }

            if (Utility::getResourceFromFile(imagePath, buffer, size))
            {
                photo.size = size;
                memcpy(photo.data, buffer.get(), size);
```

```
            }
        }
    }

    user.flag = BS2_USER_FLAG_CREATED | BS2_USER_FLAG_UPDATED;

    user.numFingers = ;
    user.numCards = ;
    user.numFaces = ;

    if (deviceInfo.cardSupported)
    {
        if (Utility::isYes("Do you want to scan card?"))
        {
            uint32_t numCard = Utility::getInput<uint32_t>("How many cards
would you like to register?");
            BS2CSNCard* ptrCard = new BS2CSNCard[numCard];
            if (ptrCard)
            {
                userBlob.cardObjs = ptrCard;
                for (uint32_t index = ; index < numCard;)
                {
                    BS2Card card = { , };
                    sdkResult = BS2_ScanCard(context_, id, &card,
onReadyToScanCard);
                    if (BS_SDK_SUCCESS != sdkResult)
                        TRACE("BS2_ScanCard call failed: %d", sdkResult);
                    else
                    {
                        if (card.isSmartCard)
                        {
                            TRACE("CSN card only supported.");
                            continue;
                        }

                        memcpy(&ptrCard[index], &card.card,
sizeof(BS2CSNCard));
                        user.numCards++;
                        index++;
                    }
                }
            }
        }
    }

    // +V2.8 XS2 QR support
    if (qrSupported)
    {
        if (Utility::isYes("Would you like to register the QR code string to
be used for authentication?"))
        {
```

```
            msg.str("");
            msg << "Enter the ASCII QR code." << endl;
            msg << "  [ASCII code consisting of values between 32 and
126].";
            string qrCode = Utility::getInput<string>(msg.str());

            BS2CSNCard qrCard = { , };
            sdkResult = BS2_WriteQRCode(qrCode.c_str(), &qrCard);
            if (BS_SDK_SUCCESS != sdkResult)
            {
                TRACE("BS2_WriteQRCode call failed: %d", sdkResult);
            }
            else
            {
                size_t numOfRealloc = user.numCards + 1;
                BS2CSNCard* ptrNewCard = new BS2CSNCard[numOfRealloc];
                memset(ptrNewCard, 0x0, sizeof(BS2CSNCard) * numOfRealloc);

                if ( < user.numCards && userBlob.cardObjs)
                {
                    memcpy(ptrNewCard, userBlob.cardObjs, sizeof(BS2CSNCard)
* user.numCards);
                    delete[] userBlob.cardObjs;
                    userBlob.cardObjs = NULL;
                }

                memcpy(ptrNewCard + user.numCards, &qrCard,
sizeof(BS2CSNCard));
                userBlob.cardObjs = ptrNewCard;
                user.numCards++;
            }
        }
    }
    // +V2.8 XS2 QR support

    if (fingerScanSupported)
    {
        if (Utility::isYes("Do you want to scan fingerprint?"))
        {
            uint32_t numFinger = Utility::getInput<uint32_t>("How many
fingers would you like to register?");
            BS2Fingerprint* ptrFinger = new BS2Fingerprint[numFinger];
            if (ptrFinger)
            {
                userBlob.fingerObjs = ptrFinger;
                for (uint32_t index = ; index < numFinger; index++)
                {
                    for (uint32_t templateIndex = ; templateIndex <
BS2_TEMPLATE_PER_FINGER;)
                    {
                        sdkResult = BS2_ScanFingerprint(context_, id,
```

```
&ptrFinger[index], templateIndex, BS2_FINGER_TEMPLATE_QUALITY_HIGHEST,
BS2_FINGER_TEMPLATE_FORMAT_SUPREMA, onReadyToScanFinger);
                        if (BS_SDK_SUCCESS != sdkResult)
                            TRACE("BS2_ScanFingerprint call failed: %d",
sdkResult);
                        else
                            templateIndex++;
                    }
                    user.numFingers++;
                }
            }
        }
    }

    if (faceScanSupported)
    {
        if (Utility::isYes("Do you want to scan face?"))
        {
            uint32_t numFace = Utility::getInput<uint32_t>("How many face
would you like to register?");
            BS2Face* ptrFace = new BS2Face[numFace];
            if (ptrFace)
            {
                userBlob.faceObjs = ptrFace;
                for (uint32_t index = ; index < numFace;)
                {
                    sdkResult = BS2_ScanFace(context_, id, &ptrFace[index],
BS2_FACE_ENROLL_THRESHOLD_DEFAULT, onReadyToScanFace);
                    if (BS_SDK_SUCCESS != sdkResult)
                        TRACE("BS2_ScanFace call failed: %d", sdkResult);
                    else
                    {
                        user.numFaces++;
                        index++;
                    }
                }
            }
        }
    }

    sdkResult = BS2_EnrolUser(context_, id, &userBlob, 1, 1);
    if (BS_SDK_SUCCESS != sdkResult)
        TRACE("BS2_EnrolUser call failed: %d", sdkResult);

    if (userBlob.cardObjs)
        delete[] userBlob.cardObjs;

    if (userBlob.fingerObjs)
        delete[] userBlob.fingerObjs;

    if (userBlob.faceObjs)
```

```
        delete[] userBlob.faceObjs;

    return sdkResult;
}
```

C#

```
public void insertUserIntoDevice(IntPtr sdkContext, UInt32 deviceID, bool
isMasterDevice)
{
    List<BS2User> userList = new List<BS2User>();
    if (dbHandler.GetUserList(ref deviceInfo, ref userList))
    {
        if (userList.Count > )
        {
            for (int idx = ; idx < userList.Count; ++idx)
            {
                Console.Write("[{0:000}] ==> ", idx);
                print(userList[idx]);
            }

            Int32 selection = Util.GetInput();
            if (selection >= )
            {
                if (selection < userList.Count)
                {
                    BS2User user = userList[selection];
                    BS2UserSmallBlob[] userBlob =
Util.AllocateStructureArray<BS2UserSmallBlob>(1);
                    if (dbHandler.GetUserBlob(ref deviceInfo, ref user, ref
userBlob[]))
                    {
                        Console.WriteLine("Trying to enroll user.");
                        //BS2ErrorCode result =
(BS2ErrorCode)API.BS2_EnrolUser(sdkContext, deviceID, userBlob, 1, 1);
                        BS2ErrorCode result =
(BS2ErrorCode)API.BS2_EnrollUserSmall(sdkContext, deviceID, userBlob, 1, 1);
                        if (result != BS2ErrorCode.BS_SDK_SUCCESS)
                        {
                            Console.WriteLine("Got error({0}).", result);
                        }

                        if (userBlob[].cardObjs != IntPtr.Zero)
                        {
                            Marshal.FreeHGlobal(userBlob[].cardObjs);
                        }

                        if (userBlob[].fingerObjs != IntPtr.Zero)
                        {
                            Marshal.FreeHGlobal(userBlob[].fingerObjs);
                        }
```

```
                        if (userBlob[].faceObjs != IntPtr.Zero)
                        {
                            Marshal.FreeHGlobal(userBlob[].faceObjs);
                        }

                        if (userBlob[].user_photo_obj != IntPtr.Zero)
                        {
                            Marshal.FreeHGlobal(userBlob[].user_photo_obj);
                        }
                    }
                }
                else
                {
                    Console.WriteLine("Invalid selection[{0}]", selection);
                }
            }
            else
            {
                Console.WriteLine("Invalid user index");
            }
        }
        else
        {
            Console.WriteLine("There is no user.");
        }
    }
    else
    {
        Console.WriteLine("An error occurred while attempting to retrieve
user list.");
    }
}
```

함께 보기

BS2_GetUserList
BS2_RemoveUser
BS2_RemoveAllUser
BS2_GetUserInfos
BS2_GetUserInfosEx
BS2_EnrolUser
BS2_EnrolUserEx
BS2_GetUserDatas
BS2_GetUserDatasEx