목차

SDK API > BS2_EnableDeviceLicense

BS2_EnableDeviceLicense

[+ 2.9.1] 라이선스를 활성화 시킬 장치들과, 라이선스 파일을 지정하여 장치 라이선스를 활성화 시킵니다.
마스터 장치 및 그와 연결된 슬레이브 장치들에 대한 일괄 지정이 가능하며, 활성화에 대한 장치별 결과는 outResultObj 및 outNumOfResult를 통해 반환됩니다.
이 기능은 장치가 라이선스 활성화 기능을 지원해야 하며, 현재 기능을 지원하는 장치 정보는 아래와 같습니다.

지원장치	펌웨어
XS2-Finger	V1.2.0
XS2-Card	V1.2.0
BS3	V1.1.0

함수

```
#include "BS_API.h"

int BS2_EnableDeviceLicense(void* context, uint32_t deviceId, const
BS2LicenseBlob* licenseBlob, BS2LicenseResult** outResultObj, uint32_t*
outNumOfResult);
```

> BS2LicenseBlob 구조체 보기
> BS2LicenseResult 구조체 보기

파라미터

- [In] *context* : Context
- [In] *deviceId* : 장치 식별자
- [In] *licenseBlob* : 장치 라이선스 정보 구조체 포인터
- [Out] *outResultObj* : 장치 라이선스 활성화 결과를 돌려받을 구조체의 포인터
- [Out] *outNumOfResult* : 장치 라이선스 활성화 결과 구조체의 갯수

> **참고**
> outResultObj 변수는 사용 후 BS2_ReleaseObject 함수를 통해 시스템에 메모리를 반환해야 합니다.

반환값

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

샘플코드(C++)

sample_bs2_enabledevicelicense.cpp

```
int setDeviceLicense(void* context, BS2_DEVICE_ID id)
{
    DeviceControl dc(context);
    BS2LicenseBlob licenseBlob = { , };
    vector<BS2_DEVICE_ID> deviceIDs;
    vector<BS2LicenseResult> licenseResult;
    int sdkResult = BS_SDK_SUCCESS;

    licenseBlob.licenseType =
(BS2_LICENSE_TYPE)Utility::getInput<uint32_t>("Enter the license type.
(0: None, 1: Visual QR)");
    licenseBlob.numOfDevices =
(uint16_t)Utility::getInput<uint32_t>("How many devices do you want to
register?");
    if ( < licenseBlob.numOfDevices)
    {
        // Device ID
        for (uint16_t idx = ; idx < licenseBlob.numOfDevices; idx++)
        {
            BS2_DEVICE_ID deviceID =
(BS2_DEVICE_ID)Utility::getInput<uint32_t>("Enter a device ID:");
            deviceIDs.push_back(deviceID);
        }

        licenseBlob.deviceIDObjs = deviceIDs.data();

        string pathName = Utility::getLine("Enter the path and name of
license.");
        licenseBlob.licenseLen = Utility::getResourceSize(pathName);
        shared_ptr<uint8_t> buffer(new uint8_t[licenseBlob.licenseLen],
ArrayDeleter<uint8_t>());
        if ( < licenseBlob.licenseLen &&
Utility::getResourceFromFile(pathName, buffer, licenseBlob.licenseLen))
        {
            licenseBlob.licenseObj = buffer.get();

            sdkResult = dc.enableDeviceLicense(id, &licenseBlob,
licenseResult);
            if (BS_SDK_SUCCESS == sdkResult)
                DeviceControl::print(licenseResult);
```

```
        }
    }

    return sdkResult;
}

int DeviceControl::enableDeviceLicense(BS2_DEVICE_ID id, const
BS2LicenseBlob* licenseBlob, vector<BS2LicenseResult>& licenseResult)
{
    BS2LicenseResult* result = NULL;
    uint32_t numOfResult = ;
    int sdkResult = BS2_EnableDeviceLicense(context_, id, licenseBlob,
&result, &numOfResult);
    if (BS_SDK_SUCCESS != sdkResult)
    {
        TRACE("BS2_EnableDeviceLicense call failed: %d", sdkResult);
        return sdkResult;
    }

    licenseResult.clear();
    for (uint32_t idx = ; idx < numOfResult; idx++)
    {
        licenseResult.push_back(result[idx]);
    }

    return sdkResult;
}
```

샘플코드(C#)

sample_setdebugfilelogex.cs

```
const string CURRENT_DIR = ".";
const int MAX_SIZE_LOG_FILE = 100;  // 100MB
IntPtr ptrDir = Marshal.StringToHGlobalAnsi(CURRENT_DIR);
result =
(BS2ErrorCode)API.BS2_SetDebugFileLogEx(Constants.DEBUG_LOG_OPERATION_A
LL, Constants.DEBUG_MODULE_ALL, ptrDir, MAX_SIZE_LOG_FILE);
Marshal.FreeHGlobal(ptrDir);
if (result != BS2ErrorCode.BS_SDK_SUCCESS)
{
    Console.WriteLine("Got error({0}).", result);
    return;
}
```