Table of Contents

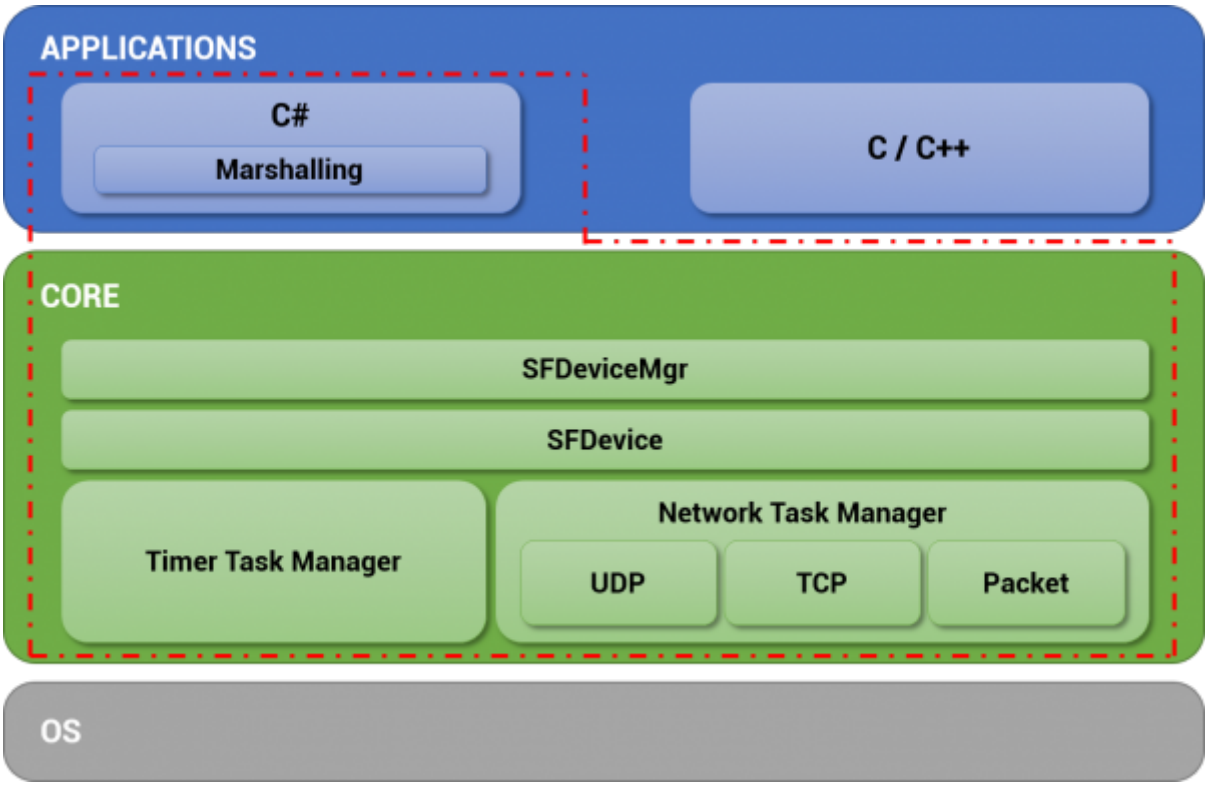
Getting Started

SDK Components

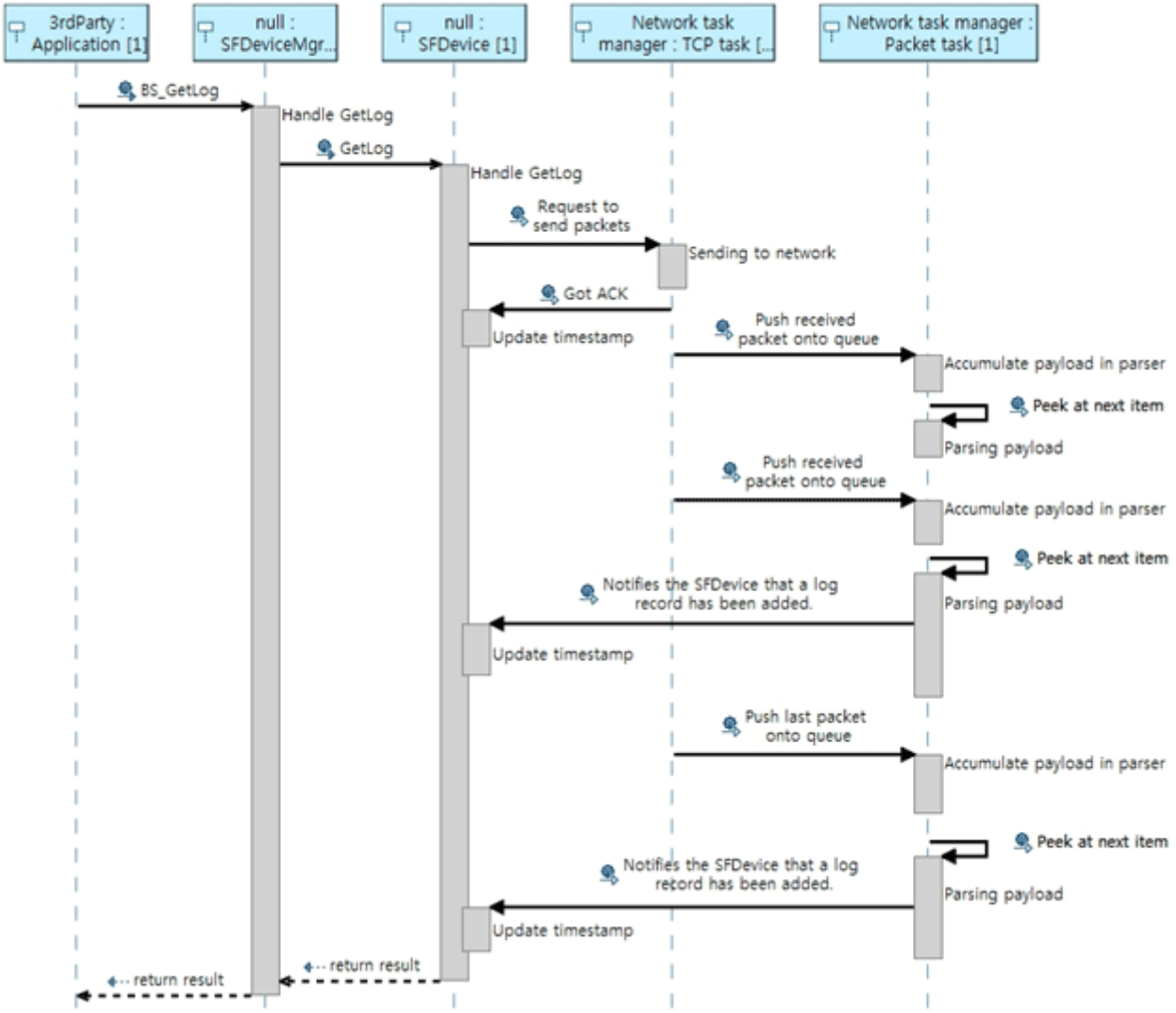
BioStar 2 Device SDK package is made of the following folders and files.

SDK	Document[1]				
	Include[2]				
	Lib	linux	lib	x86	BS_SDK_V2.so
				x64	BS_SDK_V2.so
		window	lib	x86	BS_SDK_V2.lib[3] BS_SDK_V2.dll
				x64	BS_SDK_V2.lib[4] BS_SDK_V2.dll
	Example[5]	C#			

Framework



Workflow


3rdParty : Application [1]
null : SFDeviceMgr...
null : SFDevice [1]
Network task manager : TCP task [...
Network task manager : Packet task [1]
BS_GetLog
Handle GetLog
GetLog
Handle GetLog
Request to send packets
Sending to network
Got ACK
Update timestamp
Push received packet onto queue
Accumulate payload in parser
Peek at next item
Parsing payload
Push received packet onto queue
Accumulate payload in parser
Peek at next item
Notifies the SFDevice that a log record has been added.
Parsing payload
Update timestamp
Push last packet onto queue
Accumulate payload in parser
Peek at next item
Notifies the SFDevice that a log record has been added.
Parsing payload
Update timestamp
return result
return result

Compatible device

You can use all devices which works with the BioStar 2.

Comparison with BioStar 1.x SDK

Consistency - Provides independent data structure and API

BioStar 1.x SDK had different structures and APIs for each type of devices. It was inconvenient, because making branching statements for every type was necessary to control several types of devices in one application.

```
if( m_DeviceType == BS_DEVICE_BIOENTRY_PLUS ||
    m_DeviceType == BS_DEVICE_BIOENTRY_W    ||
    m_DeviceType == BS_DEVICE_BIOLITE       ||
    m_DeviceType == BS_DEVICE_XPASS         ||
    m_DeviceType == BS_DEVICE_XPASS_SLIM    ||
    m_DeviceType == BS_DEVICE_XPASS_SLIM2)

{
    BEUserHdr userHdr;
    // Retrieve a user from the device
    BS_RET_CODE result = BS_GetUserBEPlus( m_Handle, m_UserID, &userHdr,
m_TemplateData );
    ...

    // Transfer the user to the device
    result = BS_EnrollUserBEPlus( m_Handle, &userHdr, m_TemplateData );
    ...
}
else if( m_DeviceType == BS_DEVICE_BIOSTATION )
{
    BSUserHdrEx userHdr;

    BS_RET_CODE result = BS_GetUserEx( m_Handle, m_UserID, &userHdr,
m_TemplateData );
    ...

    result = BS_EnrollUserEx( m_Handle, &userHdr, m_TemplateData );
    ...
}
else if( m_DeviceType == BS_DEVICE_DSTATION )
{
```

```
    DSUserHdr userHdr;
    ...

    BS_RET_CODE result = BS_GetUserDStation( m_Handle, m_UserID, &userHdr,
m_TemplateData, m_FaceTemplate_DST );
    ...

    result = BS_EnrollUserDStation( m_Handle, &userHdr, m_TemplateData,
m_FaceTemplate_DST );
}
else if( m_DeviceType == BS_DEVICE_XSTATION )
{
    XSUserHdr userHdr;
    ...

    BS_RET_CODE result = BS_GetUserXStation( m_Handle, m_UserID, &userHdr);
    ...

    result = BS_EnrollUserXStation( m_Handle, &userHdr );
}
else if( m_DeviceType == BS_DEVICE_BIOSTATION2 )
{
    BS2UserHdr userHdr;
    ...

    BS_RET_CODE result = BS_GetUserBioStation2( m_Handle, m_UserID,
&userHdr, m_TemplateData );
    ...

    result = BS_EnrollUserBioStation2( m_Handle, &userHdr, m_TemplateData );
}
else if( m_DeviceType == BS_DEVICE_FSTATION )
{
    FSUserHdr userHdr;
    ...

    BS_RET_CODE result = BS_GetUserFStation( m_Handle, m_UserID, &userHdr,
faceTemplate );
    ...

    result = BS_EnrollUserFStation( m_Handle, &userHdr, m_FaceTemplate_FST
);
}
```

BioStar 2.x SDK uses one structure and one API for all types of devices. The developer doesn't have to use complicated branching statements, which allows users to use simple codes.

```
BS2UserBlob userBlob =
(BS2UserBlob)Utils.AllocateStructure(typeof(BS2UserBlob));
```

```
...
int result = (BS2ErrorCode)API.BS2_EnrolUser(Program.sdkContext,
deviceHandle.info.id, ref userBlob);
...
```

Convenience - Automatic management for network interface

BioStar 1.x SDK has to get a handle(socket descriptor) on the device when connecting to it. Notifies which device to control by using the handle(socket descriptor) achieved when calling an API.

```
int handle = ;
uint deviceID = ;
int deviceType = ;

result = BS_OpenSocket( "192.168.0.5", 1471, &handle );
result = BS_GetDeviceID(handle, &deviceID, &deviceType);
```

BioStar 2.x SDK does not make the developer control the handle(socket descriptor). When the device ID is sent as a parameter, the BioStar 2.x SDK framework automatically finds the relevant device and controls it.

```
const char* deviceAddress = "192.168.1.2";
uint16_t devicePort = 51211;
uint32_t deviceId = ;
BS2SimpleDeviceInfo deviceInfo;

int result = BS2_ConnectDeviceViaIP(context, deviceAddress, devicePort,
&deviceId);
int result = BS2_GetDeviceInfo(context, deviceId, &deviceInfo);
```

Isolation - Thread Safe

BioStar 1.x SDK needs to have a 'lock mechanism' put in place by a developer to avoid a single API being called from multiple threads at the same time.
BioStar 2.x SDK is designed to allow you to call a single API from multiple threads at the same time.

Maintenance - Flexible Development

BioStar 1.x SDK required adding or modifying the UI/logic of the application when a new type of device has been added. However, **BioStar 2.x SDK** provides each device's information in a unified

structure. Therefore, modifying the existing application's UI/logic is not necessary when a new type of device is added.

For example, even though a new device supporting face recognition has been newly released, inconvenience of modifying the application can be avoided if the UI/logic has been designed to work based on the device's properties.

Building a Development Environment

Create a new project in Visual Studio

C/C++
Under construction

C#

1. Select the library directory from the SDK package and copy to the project directory.
2. Modification of the project properties is required to use the right DLL for the platform.
   Open the project properties page and enter as below on the 'Build Events' command line.

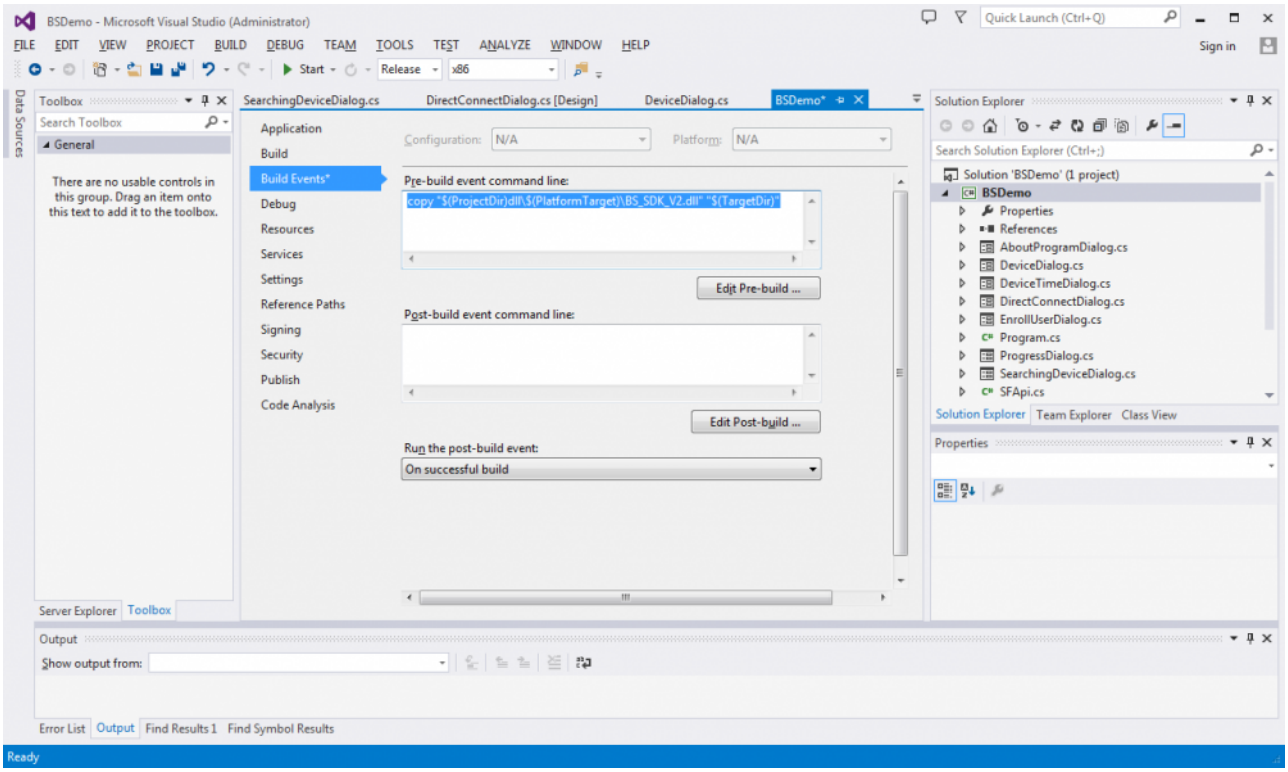
SDK Version	Platform	Input Information

Before V2.8.1	-	`copy "$(ProjectDir)lib\$(PlatformTarget)\BS_SDK_V2.dll" "$(TargetDir)"`
		`copy "$(ProjectDir)lib\$(PlatformTarget)\libeay32.dll" "$(TargetDir)"  // OpenSSL 1.0.2n`
		`copy "$(ProjectDir)lib\$(PlatformTarget)\libssl32.dll" "$(TargetDir)"  // OpenSSL 1.0.2n`
		`copy "$(ProjectDir)lib\$(PlatformTarget)\ssleay32.dll" "$(TargetDir)"  // OpenSSL 1.0.2n`

later V2.8.2	x86	`copy "$(ProjectDir)lib\$(PlatformTarget)\BS_SDK_V2.dll" "$(TargetDir)"`
		`copy "$(ProjectDir)lib\$(PlatformTarget)\libssl-1_1.dll" "$(TargetDir)"  // OpenSSL 1.1.1i`
		`copy "$(ProjectDir)lib\$(PlatformTarget)\libcrypto-1_1.dll" "$(TargetDir)"  // OpenSSL 1.1.1i`

later V2.8.2	x64	`copy "$(ProjectDir)lib\$(PlatformTarget)\BS_SDK_V2.dll" "$(TargetDir)"`
		`copy "$(ProjectDir)lib\$(PlatformTarget)\libssl-1_1-x64.dll" "$(TargetDir)"  // OpenSSL 1.1.1i`
		`copy "$(ProjectDir)lib\$(PlatformTarget)\libcrypto-1_1-x64.dll" "$(TargetDir)"  // OpenSSL 1.1.1i`

1. Copy **SFApi.cs**, **SFEnum.cs**, **SFStruct.cs** from the SDK package example codes.



1)

Document with instructions for the API provided by the SDK.

2)

Header files that defines the APIs and structures that are needed when developing C/C++ applications.

3) , 4)

Static library being imported by C/C++ projects.

5)

SDK Sample codes for different languages.