

User Guide for DSEA Call Control Libraries

Table of Contents

Revision History	3
Introduction.....	4
Purpose of this document	4
Overview.....	4
Audience	4
Abbreviations	4
Architecture.....	5
Types of libraries	5
Call Control Mechanism	5
Deployment Scenarios.....	6
BL Device Functionality.....	7
Native Libraries.....	8
Supported Platforms	8
Dependencies	8
Program Structure	9
API Call Sequence	13
Initialization and termination sequence.....	13
Headset connection disconnection sequence	14
Call events sequence	15
API Description	15
JavaScript Library.....	16
Web Browser Support	16
Program Structure	16
Softphone to SDK events	17
SDK to Softphone events	19

Revision History

Version	Date	Change
1.0	December 2021	Initial release
1.1	April 2022	Added static runtime support for windows native library Added custom presence support for JS SDK
1.2	September 2022	Updated the document with new APIs added for BL device
1.3	August 2023	Update supported OS version and build tools
1.4	April 2026	Updated supported OS version

Introduction

Purpose of this document

This document provides a high-level description of the DSEA Call Control libraries and their API usage for integration with the softphone software.

Overview

Call Control refers to the use of buttons and indicators on the headset to control and view the state of the calls in the Softphone. The Busylight (BL) device is a display device that indicates the state of calls or user presence (available, busy, in a call etc.). More info on the Busylight can be seen here: [Link](#)

The DSEA Call Control libraries provide functionality to interface the softphone with EPOS Enterprise headsets for call control and Busylight devices.

Audience

This document has been written for software developers who intend to integrate call control functionality for EPOS enterprise headsets and Busylight devices in their softphones. The reader of this document is expected to know the basic concepts of call control including softphone and headset events.

Abbreviations

Abbreviation	Description
BL	Busylight
API	Application Programming Interface
BT	Bluetooth
CC	Call Control
SDK	Software Development Kit
USB	Universal Serial Bus

Architecture

Types of libraries

The DSEA Call Control Libraries are offered in two variants:

1. **Native C++ libraries:** It consists of compact platform-specific shared libraries that should be loaded dynamically by the softphone executable process.
2. **JavaScript library:** It is a lightweight JavaScript API that can be used within a browser context for web-based softphones.

The appropriate version of DSEA Call Control library must be loaded by the softphone and the relevant APIs should be invoked to enable the call control between the headset and the softphone.

Call Control Mechanism

Both the libraries offer similar APIs and call control functionalities. At a high level, the libraries provide the below functionalities:

- Enumerate the connected EPOS/DSEA USB Headsets, BT Headset/Speaker paired with USB Dongle and BL Device.
- Receive events like accept call, hold call, resume call etc. from the USB Headset and generate corresponding events to the softphone.
- Receive softphone events and update the corresponding headset and BL device state.

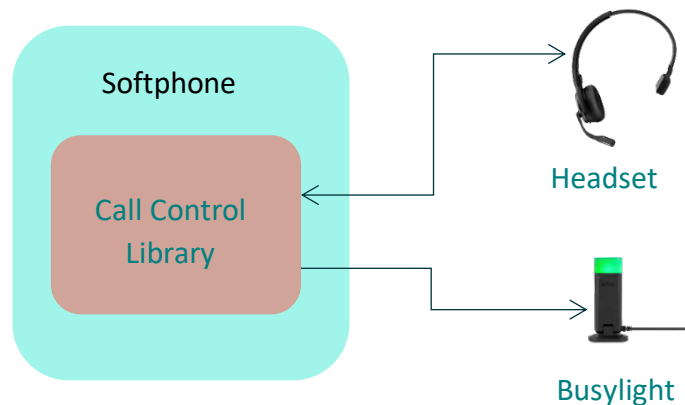
Note that all call events are asynchronous in nature and the call control library updates the device state only when the corresponding event/state of the call is updated by the softphone. This ensures that the headset state is always in agreement with the call state displayed on the softphone.

E.g., if there is an incoming call and the headset state is ringing, if the user presses the answer key on the headset, the corresponding answer call event is communicated by the call control library to the softphone and only after the softphone changes the state of the call to active, the device state is updated to active state from ringing state.

Deployment Scenarios

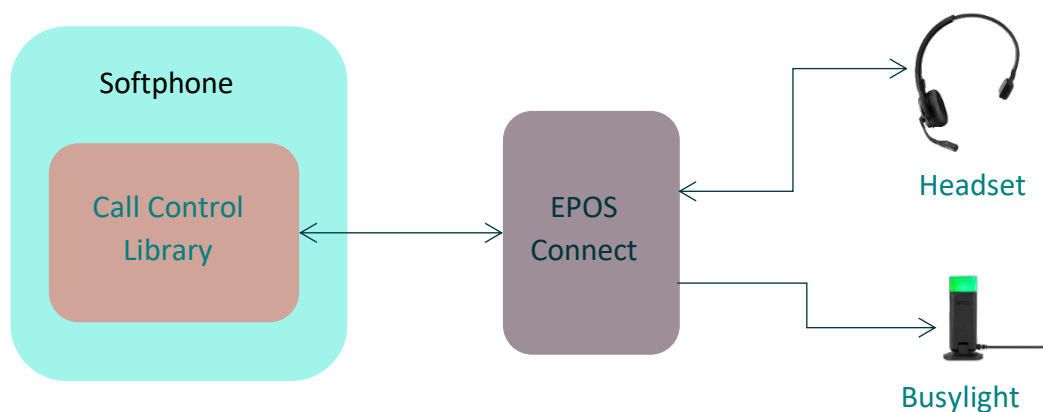
The DSEA Call Control libraries operate in one of the below two deployment scenarios:

- **Standalone without EPOS Connect application:** The library will directly communicate with the devices to provide call control functionality. The native libraries use USB HID protocol to communication directly with the devices whereas the JavaScript based library depends on the Web HID support provided by the browser for communication to the devices



- **With EPOS Connect application (available on Windows & Mac OS):** EPOS Connect is a desktop application for Windows and Mac OS that provides the latest firmware and personalization options for DSEA headsets and Busylight device. It also provides Enterprise Management of the devices along with call statistics reporting to ensure flawless operation and maximum productivity.

The call control libraries are designed to provide a seamless call control experience with EPOS connect application which leverages the extensive softphone and headset support features present in EPOS connect. You can download the latest version of the EPOS connect application here: [Link](#)



BL Device Functionality

The BL device can be controlled using the DSEA Call Control Libraries to indicate the below scenarios:

Scenario	Description	Default BL Indication [#]
Call States	Incoming Call	Long flash Red
	Outgoing Call	Long flash Red
	In a call	Solid Red
	In a call + Presenting	Solid Magenta
Predefined Presence States	Available	Solid Green
	Busy	Solid Red
	Away	Solid Yellow
	DoNotDisturb	Solid Blue
	Offline	Off
Custom Presence State	Any custom presence	Custom color and pattern
IM Notification	Used to notify new instant message (IM)	Blue Flash

#Notes:

1. The colors and patterns can be customized using EPOS Connect application
2. The default values may vary when used with different versions of EPOS Connect application

Native Libraries

Supported Platforms

The native version of the DSEA Call Control library is supported on multiple platforms as listed below:

OS Version	Compiler	Hardware
Windows 7, Windows 10 or later	MS Visual Studio 2022 ⁺	x86 and x64
macOS Sequoia (15.7.5), macOS Sonoma (26.3.1 or later)	Xcode 16 ⁺ or higher	x64 and ARM64
Linux Ubuntu 20.04 or later	GCC 7.5.0 ⁺ or higher	x64

⁺C++20 support option must be enabled on the compiler.

Important Note: On Linux, the library uses the hidraw Linux driver. Following commands needs to run on the Linux setup for the library to communicate with the headset.

1. Create file /etc/udev/rules.d/50-epos.rules with contents:

```
KERNEL=="hidraw*", ATTRS{idVendor}=="1395", MODE="0666"
```

You can modify the above command to allow access to a particular user as well.

2. Execute command "udevadm control --reload-rules && udevadm trigger"

Dependencies

The native version of the DSEA Call Control library requires the libraries below to be available.

OS	Dependent libraries
Windows	hid, cfgmgr32, bcrypt vcruntime.dll and msvcp140.dll must be included for dynamic runtime (drt) linked libraries
Mac OS	pthread, IOKit, Foundation
Linux	pthread, udev, crypto, rt, dl

Program Structure

To use the Call control library, the softphone must provide the implementation of the below:

SoftphoneInfoSt: This structure provides information regarding the softphone, and it is used for logging and identification purposes.

```
// Softphone info
SoftphoneInfoSt softphoneInfo("MySoftphoneName", "MyBrand", "1.0.0");
```

ICallControlListener: This interface provides the call back methods from the library to the softphone to notify headset related events and status changes in the library. This interface should be implemented by the softphone to receive events from the library.

```
#include "ICallControlListener.h"

using namespace dsea::callcontrol::sdk;

// Call control listener
class SoftphoneCallControlListener : public ICallControlListener
{
public:
    MyCallControlListener() = default;
    virtual ~MyCallControlListener () = default;

    // Override methods
    void LoginStatusChanged(LoginStatusValue f_bLoginStatus) override
    {
        switch(f_bLoginStatus)
        {
            // Add implementation here
        }
    }

    // override other methods here
};
```

ILogger: The ILogger interface provides the logging API and this needs to be implemented by the softphone. All the logs from the library will be directed to this logging API.

```
#include "ILogger.h"

using namespace dsea::callcontrol::sdk;

// Logger implementation
class SoftphoneLogger : public ILogger
{
public:
    MyLogger() = default;
    virtual ~MyLogger() = default;

    // Override methods
    void WriteLog(LogLevel f_ecLogLevel, const std::string& f_strLogMsg) override
    {
        switch (f_ecLogLevel)
        {
            // Add implementation here
        }
    }
}
```

The entry point for the client for the Call Control Library is the below exported function in the header file "ICallControlInterface.h".

```
// Create the call control interface instance
dsea::callcontrol::sdk::ICallControlInterface* CreateCallControlInstance(
    const dsea::callcontrol::sdk::SoftphoneInfoSt& f_softphoneInfo,
    dsea::callcontrol::sdk::ICallControlListener& f_callControlListener,
    dsea::callcontrol::sdk::ILogger& f_logger,
    bool f_bEnableVerboseLogging)
```

Using the **CreateCallControlInstance** function, the interfaces below can be obtained/used.

ICallControlInterface: This is the entry level class interface which provides API to retrieve the ISoftphoneEventHandler interface and version information.

```
#include "ICallControlInterface.h"
#include "MyCallControlListener.h"
#include "MyLogger.h"
using namespace dsea::callcontrol::sdk;

SoftphoneLogger logger;
SoftphoneCallControlListener listener;
SoftphoneInfoSt softphoneInfo("MySoftphoneName", "MyBrand", "1.0.0");
ICallControlInterface* callControlInstance = nullptr;
std::shared_ptr<ISoftphoneEventHandler> eventHandler;

void createMyCallControlInstance()
{
    callControlInstance = CreateCallControlInstance(softphoneInfo, listener, logger, true);
    assert(callControlInstance != nullptr);
    std::cout << "Call Control version : << callControlInstance->GetVersion();

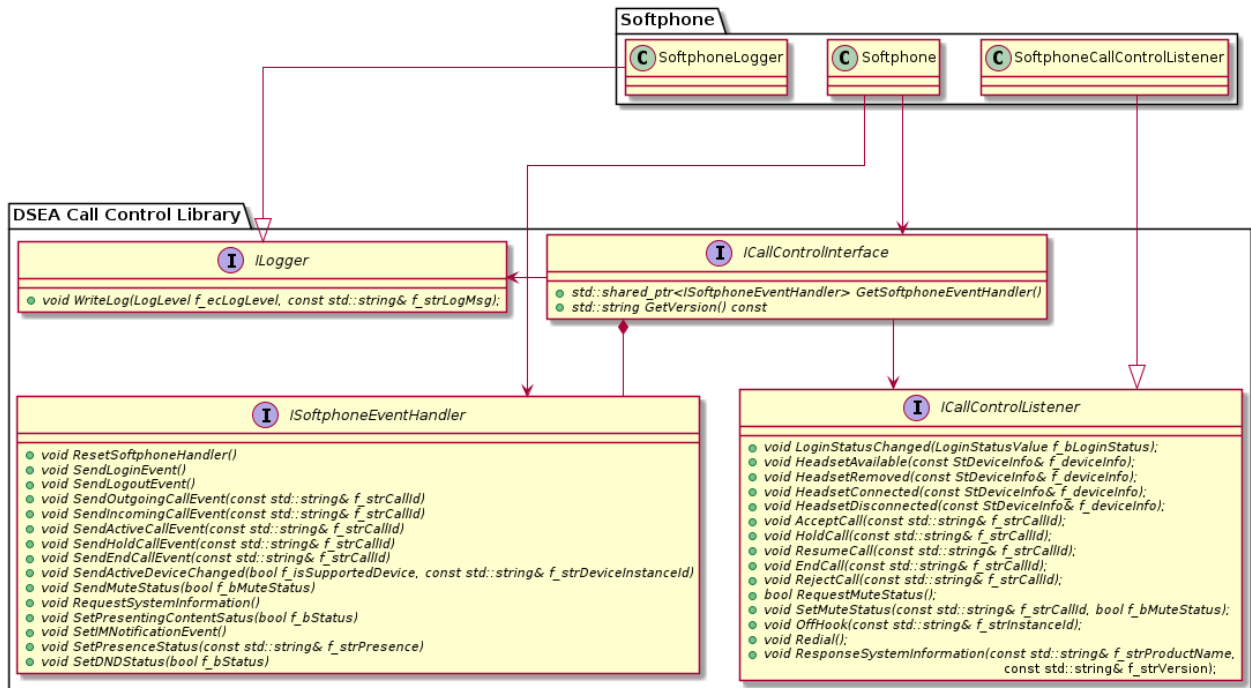
    // Send Login event
    eventHandler = callControlInstance->GetSoftphoneEventHandler();
    eventHandler->SendLoginEvent();
```

ISoftphoneEventHandler: This interface methods provide functionality to notify the call events from the softphone to the headset.

The softphone event handler instance can be used by the softphone to pass the call events to the library.

```
// Send incoming call event
eventHandler->SendIncomingCallEvent("CallId");
```

The below class diagram shows the relation between the classes.



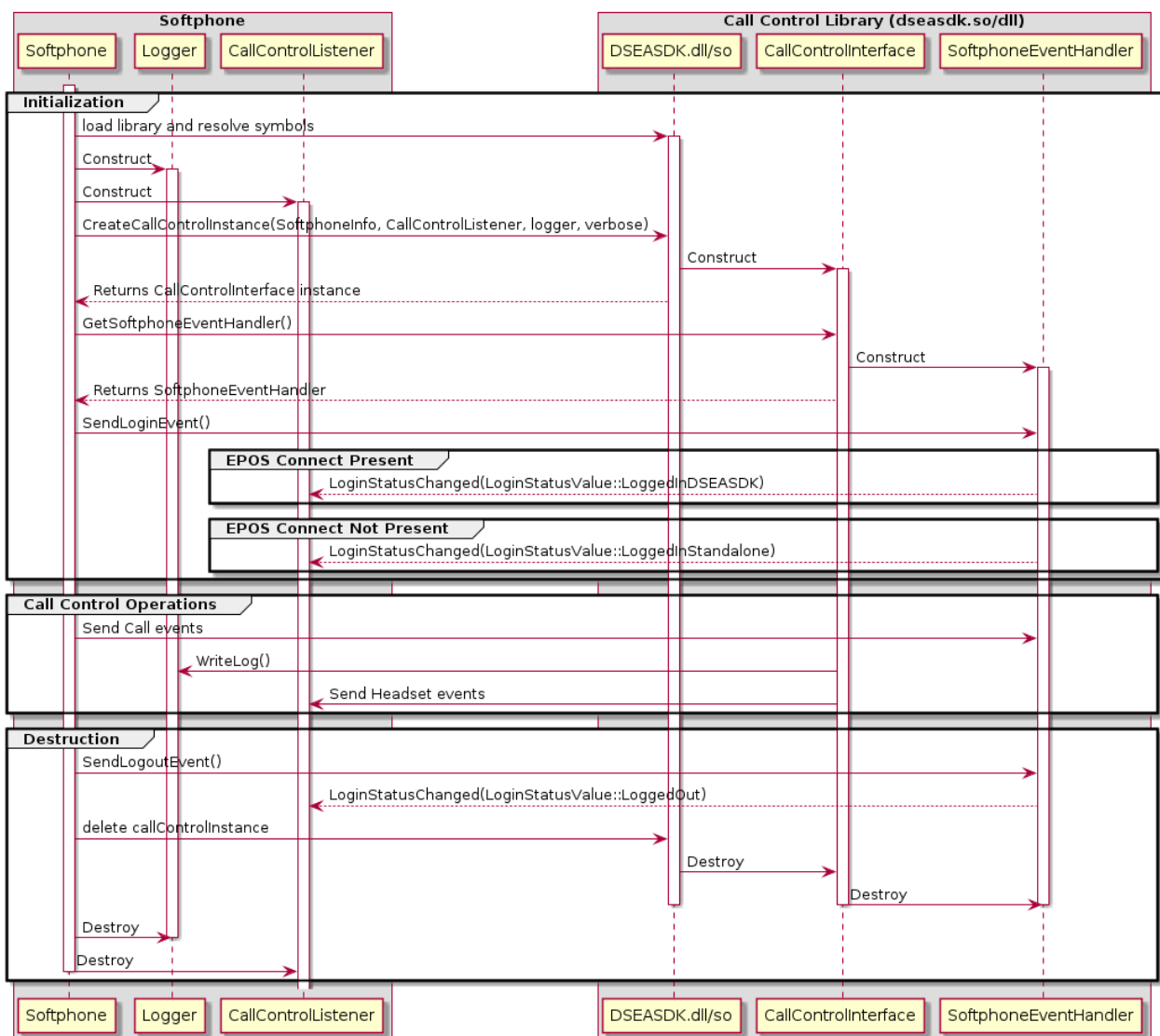
API Call Sequence

Initialization and termination sequence

The below sequence diagram shows the initialization and termination sequence for the call control library.

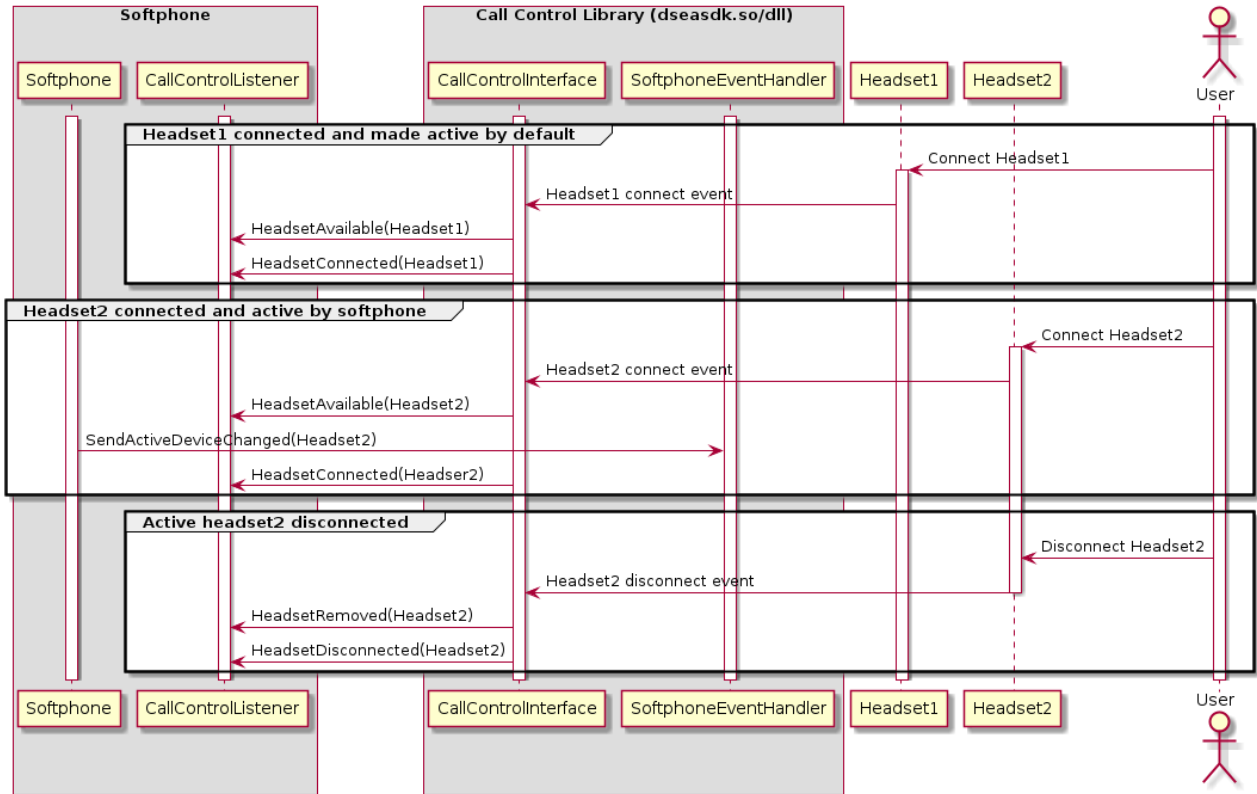
In the initialization step, the library is loaded, and the softphone event handler interface is obtained for communicating the softphone events to the library. The call control listener interface is implemented by the softphone and is provided to the library. A login event is sent from the softphone to the library, and the initialization is completed once the softphone receives the Login Status as Logged In from the library.

In the termination step, the softphone sends the logged-out event to the library and deletes the instance of call control interface created in the initialization step.



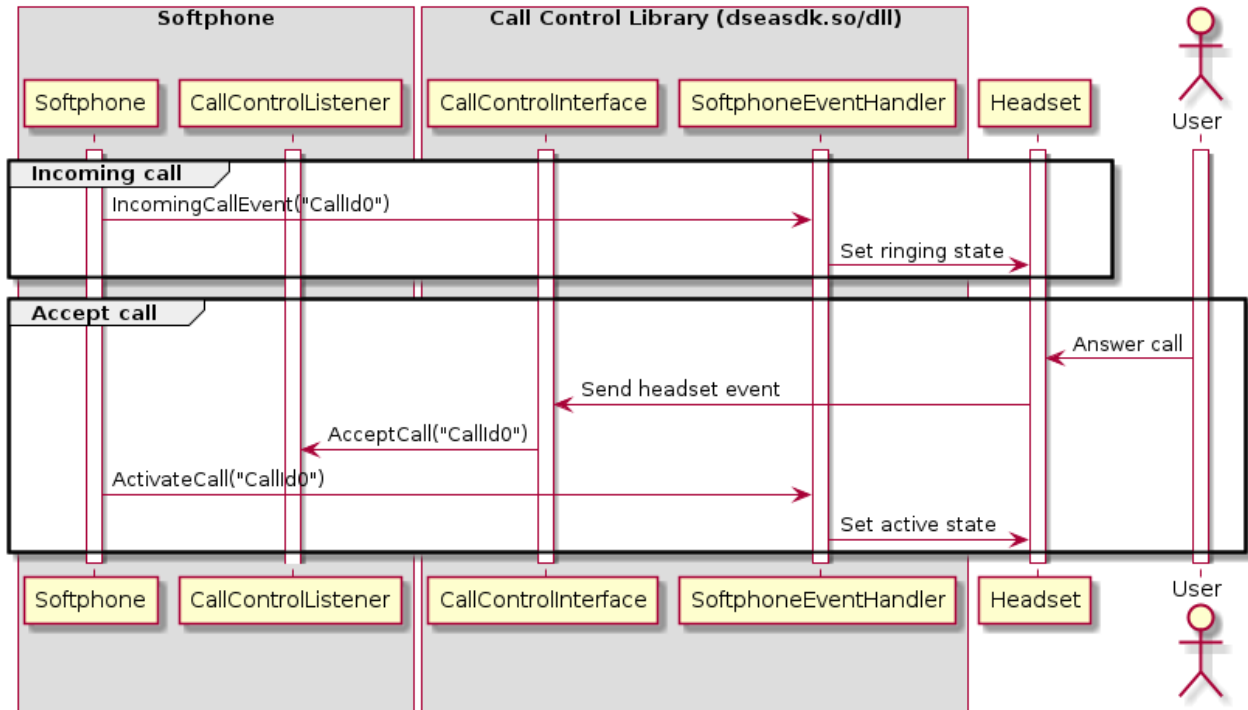
Headset connection disconnection sequence

The below sequence diagram shows the headset connection and disconnection sequence. If there are multiple headsets connected, the client should call the *SetActiveDeviceChanged()* API to set the active headset device which will be used for call control.



Call events sequence

The below diagram shows how the call events and headset events are communicated between the softphone and call control library.



API Description

The API details for the native DSEA Call Control Library can be found in a separate Doxygen generated document DSEA Call Control SDK - Library API.pdf.

A demo softphone demonstrating the use of the library is included in the distribution package.

JavaScript Library

Web Browser Support

The JavaScript library requires the below browser support based on the availability of EPOS connect application

- Without EPOS Connect: The browser should support Web HID protocol for communication with the devices.
- With EPOS Connect: The browser should support Websocket protocol for communication with EPOS Connect.

Program Structure

The JavaScript library can be loaded by referencing the below file

```
<script src="https://dseasdk.com/common/js/dseasdk.v1.js"></script>
```

The SDK object can be instantiated using the below method

```
// function dseacallcontrolsdk(brand="",name="",version="", forceHID = 0)  
mysdk = new dseacallcontrolsdk("MyBrandName","MySoftphoneName","MySoftphoneVersion")
```

Softphone to SDK events

The SDK provides the below methods for sending the call control events to the SDK:

Method	Arguments	Description
SendLoginEvent	None	Initiates the login operation on the SDK. The 'LoginStatusChanged' event indicates the result for this request.
SendLogoutEvent	None	Initiates the logout operation on the SDK. The 'LoginStatusChanged' event indicates the result for this request.
SendTerminateEvent	None	Terminates the SDK
SendMuteStatus	Mute (true/false)	Sends the mute status to the headset
SendIncomingCallEvent	Call id	Creates a new incoming call or sets state of call to incoming
SendOutgoingCallEvent	Call id	Creates a new outgoing call or sets state of call to outgoing. If there are any other existing outgoing or active calls, they end.
SendEndCallEvent	Call id	Ends the existing call
SendHoldCallEvent	Call id	Creates a new call which is in hold or sets state of existing call to hold.
SendActiveCallEvent	Call id	Creates a new call which is active or sets the state of existing call to active.
SendPresenceEvent	["Available", "Busy", "Away", "DoNotDisturb", "Offline"]	Sets the presence state on the Busylight device
SendCustomPresenceEvent	presenceName, presenceColor, presencePattern	Sets a User Defined Custom Presence State on the Busylight device. presenceName: Expected value is any string. This parameter has no validation; it can also be left empty. presenceColor: Expected value is of the format "#RRGGBB". presencePattern: Expected value is {0,1,2,3,4}: 0 -- Busylight will not Glow 1 -- Busylight will continuously Glow 2 -- Busylight will Flash 3 -- Busylight will Pulse (Breathing Light effect) 4 -- Busylight will glow with a Long Flash Pattern
SendIMEEvent	None	Sets the IM Notification light on Busylight device when device is in IDLE state or notifications are enabled during call state
SendPresentingEvent	Presenting (true/false)	Sets the Busylight device to Presenting state in an Active Call state if set to true.

SendActiveDeviceChanged	Supported – true if it is a DSEA device HS – headset name	Selects the specified headset as active. If call control should be disabled, the 'Supported' flag should be set to false.
AllowHS	None	Displays dialog to select headset device
AllowBL	None	Displays dialog to select Busylight device

SDK to Softphone events

The below call control events from the SDK can be received by registering event listener on the webpage document object.

Event	Arguments	Description
HeadsetAvailable	Headset object { "name": <name>, "id": <id>, "serial": <serial num>, "firmware":<firmware ver>, "vid": <vendor id>, "pid": <product id>, };	A new headset is connected event
HeadsetRemoved	Headset name	A headset is removed event
HeadsetConnected	Headset name	A headset is set as active
HeadsetDisconnected	Headset name	A headset is made inactive
LoginStatusChanged	["EstablishConnection", "SPLoggedIn", "SPLoggedInHID", "SPLoggedOut", "SPLoggedOutHID"]	Indicates the login connection status of the SDK
MuteMicrophone	muteStatus [true, false]	Mute event from headset
EndCall	Call id	End call event from headset
RejectCall	Call id	Reject call event from headset
HoldCall	Call id	Hold call event from headset
AcceptCall	Call id	Accept call event from headset
ResumeCall	Call id	Resume call event from headset
Redial	Call id	Redial event from headset
Offhook	Headset name	Offhook event from headset
BLsPresent	Number of BL devices	Number of BL devices connected

Example of registering for the 'HeadsetAvailable' event is shown below:

```
document.addEventListener('HeadsetAvailable', function (e) {
    console.log("HeadsetAvailable " + e.detail.name);
}, false);
```