Call Control - an Integrator’s Guide

How to implement efficient call control using Jabra solution components

Integration Guide

Revision history

Rev.	Date	Description
1.0	2020-12-22	The initial version of this document.
1.1	2021-03-15	*Updates in section 1.3: sections 1.3.1, 1.3.2, and 1.3.3 added to provide a better overview and more detailed explanation of terminology. *Updates in section 2 to clarify functionality. Section 2.1 with information on audio control added. *Rearrange and partly rewriting of section 4 to clarify subjects. Section 4.2 on 'synchronous/asynchronous function calls' added. *Initial section 5 moved to Appendix. To further clarify this subject a section on 'request-acknowledge interactions' and a section on 'receiving events from the device' has been added. *Initial sections 8, 9, and 10 are removed; the information is now included elsewhere. *Updates in section 7: section heading changed for clarification, description of multi-function button for subject clarification. *Appendix updated with a figure depicting the multi-function button on a headset.
1.2	2021-04-22	* Section 6.6 describing Call Lock added. Additionally, updates to section 6.3, Figure 4, Figure 5, Figure 6, and the Appendix to reflect this.
1.2.1	2021-06-28	* Minor changes to Call Lock text in Section 6.6. * Minor change to Figure 5. * Minor rewrite in section 2 and section 8 to clarify functionality.

Table of Contents

Table of Figures

1 Introduction

Integrating call control into your softphone application will improve the user experience for your customers. Users can control calls simply by pressing buttons on the Jabra device; this literally places control of calls at their fingertips.

1.1 Target audience

This guide is targeted at product managers, product owners, engineering managers, software developers, software architects, and other software solution professionals who need an overall understanding of how Jabra devices are integrated and deployed as active, value-adding components in software solutions.

1.2 About this guide

This guide is divided into sections, each with a specific topic. Each section can be read as a stand-alone text, but we encourage you to look through all sections, as these will provide you with useful information on how Jabra devices operate. When applicable, notes, examples, and tips will be provided.

Abbreviations and acronyms are used throughout the guide, you can find a list of them in the 'Abbreviations and Acronyms' section.

1.3 Terminology

Throughout this guide, we use terms that are closely related to telephony and call control functionality. Below you will find an explanation of some of the most common terms used in this guide.

1.3.1 Telephony

'On-hook' and 'off-hook' are telephony terms for a key concept used in Jabra call control.

The terms describe the 'call state' of a device; it can be 'True' or 'False' (represented as the value '1' or '0' respectively).
When initiating a call, the call state of the device is set to active (off-hook is 'True'), when ending a call, the call state of the device is set to inactive (off-hook is 'False').

This terminology is based on early telephony where you literally took the telephone receiver off-hook to make a call and put it back on-hook when the call was finished, see pictures below:

Figure 1 Old telephones with hook-switches.

With modern softphones you have far more options than you had with analog telephones in the early days; you can, for example, have multiple active lines at the same time. Nowadays you usually choose the call recipient from a list before activating the line to make the call, beforehand you had to activate the line before dialing numbers.

The interaction metaphor of the hook-switch allows Jabra devices to work with modern softphone systems as well as traditional desk phones using the same protocol.

Please note that while we use the term ‘on-hook’ to explain when the device is ‘not off-hook’ this term is generally not used in the APIs that handle call control functionality. Most APIs use the terms ‘OffHook(True)’ or ‘OffHook(False)’ to describe the call state of the device.

1.3.2 Call control/remote call control

‘Call control’ is used to describe the possibility of controlling call functionality from a Jabra device instead of using the softphone application user interface (GUI).

The ability to control this remotely from the device (as opposed to using the softphone user interface) can also be referred to as ‘remote call control’, however, for ease of understanding and wording, we only use the term ‘call control’ throughout this guide.

1.3.3 Jabra devices

In this guide, the term ‘Jabra device’ refers to a Jabra device that has call control functions, i.e. headsets and speakerphones. In contrast, some of the desk stands and cameras, in general, do not have call control functionality.

2 Call control

Digital headsets and speakerphones from Jabra provide the user with means of controlling the flow of voice and video calls directly from the device. This is call control.

The user can start and end calls, put calls on hold, adjust volume up/down, or mute the call from the device itself, without the need for a keyboard/mouse to interact with virtual buttons on a computer screen.
Providing the controls on the physical device gives an intimate level of control, making a daily routine filled with calls easier and faster for the end-user.

The basic call control functions available to the user are:

Function	Purpose
Answer/start call	To answer an incoming call or start a new call.
End call	To end the current active call.
Volume up/down	Adjust the sound volume during an active call.
Mute/unmute	Mute or unmute the microphone in an active call.
Hold and resume call	Put the current active call on hold (waiting), allowing it to be transferred, or allowing a temporary call to be made before resuming the original call.
Reject call	Decline an incoming call.

Table 1 Basic call control functions.

The call control functions on the devices are activated by interacting with buttons or tapping on:

- the device itself
- a controller attached to the USB cable
- a base station (for some wireless devices).

Some devices can activate the mute function by lifting the boom arm to a vertical position, on others you can answer a call by removing the headset from the base station - this depends on the device.
Consequently, the device may send e.g. an off-hook request as a result of one of several different actions. This means that an off-hook request represents a user intent, rather than a specific device interaction (e.g. button press).

2.1 Volume up/down

It is important to stress that volume up/down commands from the Jabra device are handled directly by the operating system - the softphone application should *not* react to this at all.
We do not recommend that softphone applications try to control volume up/down; if at all, the softphone should interact with the operating system APIs.

Volume buttons are not call control exclusive, they also work during non-call use of the devices, e.g. when listening to music or seeing an online presentation. Because of this, the device will send volume up/down signals even when the device is on-hook.

3 Platforms

Call control integration can be performed on the following platforms:

- Windows (C, C#)
- macOS (C, Objective-C)
- Linux (C).

Furthermore, integration into JavaScript environments is possible using the plug-ins and language wrappers of the Run-Time Environment adaptors for:

- Node.js
- Browser (Chromium-based).

Jabra SDKs are available on the Jabra Developer Zone.

Figure 2 below provides a quick overview of integration components:

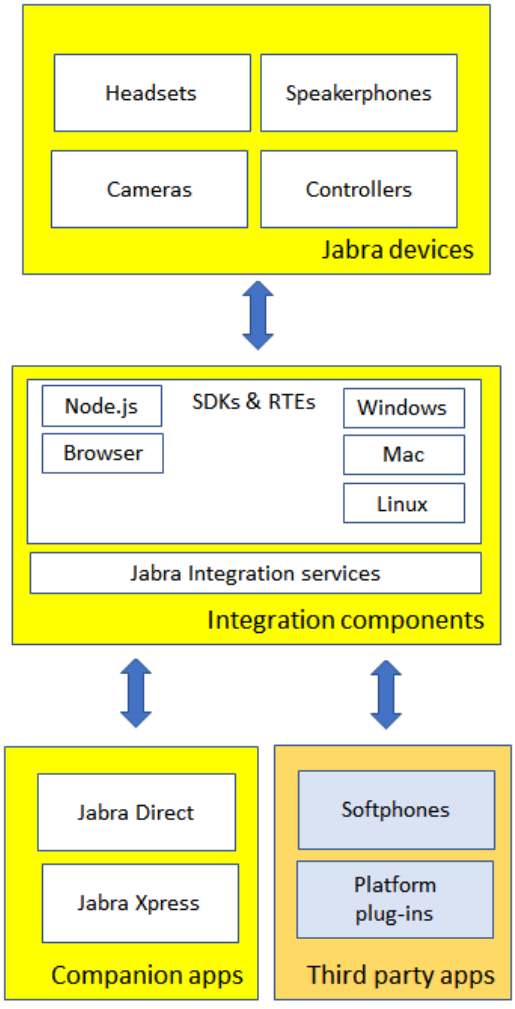

Figure 2 Overview of integration components.

4 The Software Development Kit

Software Development Kits (SDKs) enable the integration of digital functions such as call control in Jabra devices. The SDKs are available as:

- SDK packages for Windows, macOS, and Linux integration.
- NPM packages on NPM and GitHub for Node.js and browser integration.

The SDK packages are available on the Jabra Developer Zone where you also find links to NPM packages and GitHub.

For integration on Chromium-based browser please note, that your end-users will need to install the Jabra Browser Integration extension and the Jabra ChromeHost software for the SDK libraries to work; information on how to obtain these are given on the Browser SDK page on the Jabra Developer Zone.

The content of the SDKs varies depending on the platform you are using, but they all contain:

- A demo application and its source code
- Documentation
- Libraries.

The demo application serves as a simple example of how to use the SDK libraries. Turn to the documentation for more details, here you will find User Guides, API reference files, and release notes.

Further information on the SDK libraries is provided in the Appendix, see page 24.

4.1 The Application Programming Interface

The Application Programming Interface (API) for native applications, i.e. Windows, macOS, and Linux, consists mainly of:

- A range of direct function calls
- A set of callback services
- A set of data structures
- Enumerated values for parameters and commands.

The API is implemented by a library that can be distributed free of charge with an application.

4.2 Synchronous and asynchronous function calls

The functional interface of the API in the Jabra SDKs has both direct (synchronous) functions and callback (asynchronous) functions:

- Synchronous functions are generally used when communicating from the application to the device.
- Asynchronous functions are used for reporting signals from the device to the application.

For native application integrations (Windows, macOS, and Linux) it is important to note that callback functions are executed in a thread-context of the library rather than the main thread of the application; it is thus important to perform only light-weight activity in the callback functions.

The callback thread is essential to monitor the incoming signals and data from the devices, for this reason, the application must make sure that callback functions return the use of the thread as soon as possible. Spending too much time doing other processing in the context of the library thread may result in data and signals from the device being missed by the library.

Please note: this document will not elaborate on the platform-specific syntax and formats of the API, please refer to the platform-specific User Guides for details.

4.3 Initializing the SDK (Windows/macOS/Linux)

The SDK library must be initialized before it can be used. This will allocate and activate the necessary system resources required by the library to function.

During initialization, the library starts a dedicated thread on behalf of the client application to implement the exchange of data with the devices. This thread will monitor the connection and will call the callback functions when signals and data arrive from a device.

The initialization function is called only once and takes a set of callback functions and some configuration values as parameters.

Before process shutdown, remember to call ‘Jabra_Uninitialize’ to release the allocated resources properly.

4.4 Application key

Please note that you need a valid application ID key before you can initialize the SDK libraries. You can obtain one from the Jabra Developer Zone.

Go to API Keys and register your application. Observe that you need a unique key for each application.

5 Selecting the call control device

It is essential to choose the right Jabra device for call control. Below you will find guidelines on how to do this. Please note the details in ‘Bluetooth devices and dongles’ if you are using Bluetooth devices.

You can use a function call in the API to let the USB library provide you with a list of currently attached devices. Further, the USB library can call a callback function when devices are attached or removed, this lets you keep track of which device to use.

5.1 Getting a list of attached devices

Use the appropriate function to obtain a list of the currently attached devices e.g. ‘Jabra_GetAttachedJabraDevices’ if you are using the Jabra SDK for Windows.

You will get notifications from the ‘DeviceAttached’ and ‘DeviceRemoved’ callback functions when the list changes. Please refer to the platform-specific User Guide for further information.

The device list includes all Jabra devices, audio devices as well as non-audio devices. Non-audio devices such as desk stands do not support call control. Real audio devices are headsets, speakerphones, and dongles for Bluetooth headsets.

5.2 Bluetooth devices and dongles

Jabra Bluetooth devices must be connected to the accompanying USB dongle for call control to be supported. Note that the Bluetooth headset and the dongle are shown as separate devices in the above-mentioned list.

Call control functions and events are *handled by the USB dongle* connected with the Bluetooth headset, not the headset itself.

Please also note that Bluetooth headsets connected directly via a built-in Bluetooth module in a laptop will not appear in the list of attached Jabra devices.

5.3 Devices without Bluetooth

Headsets without Bluetooth, i.e. those that are connected directly via a USB cable, are identified by the headset itself.
DECT devices require a desk stand/base stand to operate, the stand and the headset are seen as one unit.

5.4 Filtering out non-audio devices

Jabra devices have a code that reflects the product variant, you can use this to determine which devices support call control. The codes for call control devices are:

- 01-xx (headsets)
- 04-xx (Bluetooth dongles)

- 08-xx (speakerphones).

To filter devices using the variant code you need to use the ‘VariantType’ property. You can also choose to do it the opposite way and remove irrelevant devices:

- FF-xx (non-Jabra devices)
- 02-xx (desk stands)
- 03-xx (desk stands).

5.5 Correlating with operating system audio devices

If you need to correlate the device list with devices reported by the OS you can use the vendor ID (VID) and a product ID (PID). Jabra device VID is 0x0B0E.

Note that headsets always have a corresponding microphone and speaker.

6 Interacting with devices

The API interaction between the softphone application and the Jabra device is generally based on:

- A request-acknowledge style exchange for the major functions
- Synchronous functions for minor operations
- Informational signals from the device.

The signals from the device are generally translated into ‘events’ in the SDK API, these are communicated to the application through callback functions.

The protocol format is derived from the USB standard HID protocol, which is the foundation of how Jabra (and other USB device vendors) handles call control in the devices.

Functionally, there is a close mapping between USB HID reading and writing and the API function calls, but the library encapsulates all the bit-field manipulation and timing management of the USB HID protocol.

6.1 Request-acknowledge interactions

Special attention must be directed to the request-acknowledge interactions. Signals/messages will be sent between the application and the device; these interactions will require a specific response in return to be completed successfully.

As an example, sending an off-hook message requires a similar message in return, otherwise, the off-hook message is ignored:

a) An off-hook request sent from the headset will require an acknowledge command from the application in return.
b) Any off-hook command sent from the application causes an acknowledge message from the headset in return.

Please see section 6.4 for a special note on ‘off-hook’ functions.

6.2 Controlling the state of the device

It can be informative to look at the call control mechanism of the devices as state machines.

For the most part, the application is in full control of the state of the headset. Changing state, e.g. on-hook to off-hook or not-ringing to ringing, is generally controlled by the softphone application, not the Jabra device. In other terms: the softphone application is the boss; the device is the subordinate.

Let’s take a simplified example using off-hook/on-hook to illustrate this:

A softphone gets an incoming call, sends a ring signal to the headset, the user then presses the ‘answer’ button on the headset to take the call. Following a conversation, the user ends the call by pressing the ‘end’ button on the headset.
This is illustrated in the sequence diagram below, please note*, that the example is for illustrative purposes only*, it does not depict the full wording nor the full amount of message exchange between a device and a softphone application.

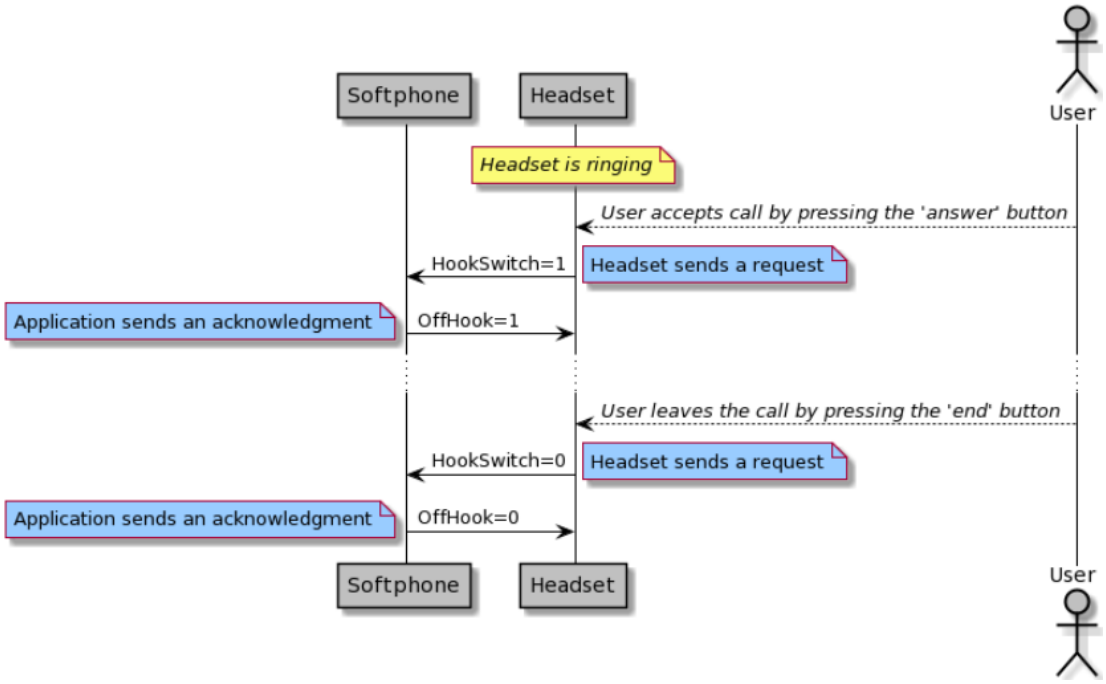

Figure 3 Example of simplified off-hook/on-hook sequence diagram (illustrative purpose only).

From a device integration point of view, this happens after the softphone has sent a ring signal to the headset:

1. The headset sends a 'HookSwitch=1' (i.e. off-hook) event to the application and waits for the application to respond with instructions on what to do.
   This is a request.

2. The application sends an 'OffHook=1' command to the headset to acknowledge the event.
   This is the acknowledge command.

3. The call is active - the user has a conversation.
   This is the result of the acknowledgment.

4. The headset sends a 'HookSwitch=0' (i.e. on-hook) event to the application and waits for instruction.
   Again, this is a request.

5. The application sends an 'OffHook=0' command to acknowledge the event.
   This is the acknowledge command to the new request.

6. The call state is inactive - the call is terminated.
   This is the result of the acknowledgment.

Refer to section 7 on page 18 for correct sequence diagram examples, a state diagram showing off-hook/on-hook is available in the Appendix, see page 27 for reference.

If this sequence was the other way round, i.e. the user answers the call by pressing a virtual button on the GUI, the softphone application will send an off-hook command to the headset, the headset will then return an acknowledgment. See section 7.2 for a detailed sequence diagram.

It is important to observe the ‘request-acknowledge’ and ‘command-acknowledge’ orders, failing to do so can lead to misinterpretations.
Consider a softphone application sending two commands in a row without waiting for the response to the first command; the acknowledgment to the first command may be seen as a stray or unexpected event or perceived as something else when it is received, this might lead to misinterpretations and possible errors.

Please see section 6.4 for a special note on ‘off-hook’ functions.

6.3 Messages must be seen in context to the current operational state

It is important to understand that your application not necessarily has to react to all messages received by the device; messages that are ‘out of context’ from the situation at hand should be ignored.
Events from an attached USB device, e.g. a headset, are visible/sent to all applications that connect to the headset if they have registered a callback function. Your softphone application should therefore only react to events that you are expecting, ignoring events that do not fit into the context of the current operational state of the softphone. Let us clarify this with an example:

If your softphone has an incoming call and sends a ring signal to a headset, then the expected message from the headset is an acceptance, a rejection, or simply a ‘no answer’. Here, either of the three is an expected return message, however, if your softphone does not have an incoming call, nor has it sent a ring signal to the headset, then it should not react to any ‘off-hook’ messages from the headset.

Situations with ‘out of context’ messages can happen for various reasons; one might be interference from other softphones. We strongly recommend against reacting to unexpected off-hook events, despite this being fully supported and permitted, it is likely to cause interference with other softphones or UC clients on the client PC.

We have seen an example where a softphone application reacts to an unexpected off-hook event from a device; interpreting the off-hook message as a desire by the user to make an outgoing call, the application presents a dial-pad on the screen and a dial-tone in the headset - however, the message was sent as an acknowledgment from the device to a different application. Reacting to an unexpected off-hook message in this way will most likely annoy the end-user; it will be a nuisance if it happens again and again...

To avoid clashing with other softphones use the Call Lock functionality described in section 6.6 on page 17.

Please also refer to page 25 in the Appendix; here you can find more information on how to reduce interference between multiple applications.

6.4 A special note on receiving events from the device

To make a correct integration of the call control API, it is important to understand a key item regarding events:

Important note
The events sent from the device to the application are not simple keypresses. There is *no* one-to-one relation between pressing a button on the device and the application receiving an event.

Instead, the events sent from the device are signals notifying the application that something has happened. It could be the user expressing an *intent* to do something by interacting with the device (pressing a button), it could be an *acknowledgment* to a request, or even an *error condition* (headset out of range from the base station), hence they cannot simply be interpreted as button presses.

Unfortunately, the current Jabra core library implementation contains an inconsistency in terminology based on events being button presses. The names of the callback functions registered during initialization (see section 4) can give reason to believe that they represent button presses, however, this is not the case.
Callback function names such as 'ButtonInDataTranslatedFunc' (C) or 'TranslatedButtonInput' (C#) or function parameters such as 'buttonInData' (C) and 'TranslatedButtonInputEventArgs' (C#) might emphasize the misunderstanding that an event corresponds to a button press. While this may not have a major impact on e.g. mute events, it does affect the request-acknowledge style interactions described earlier in this chapter. Both the request and the acknowledge events are the same; *this is especially important for the off-hook function.*

If the application is written with a generic keypress handler, then ending a call from the application by sending an off-hook message to the device will result in a 'ghost keypress' arriving from the device a little later (up to two seconds). This, however, is *not* a keypress, but rather the acknowledgment from the device to the off-hook command/request from the application.

Some applications merely ignore out of context 'off-hook keypresses', this means that everything more or less continues to work. But for other applications, e.g. contact-center clients where calls may be answered back-to-back, this may result in the application accidentally ending the call on a customer, who may have been waiting in a queue.

Note: At USB level, the signal is not referred to as 'off-hook', it is called 'HookSwitch', indicating the hook switch is either on (switch closed = handset lifted from hook) or off (switch open = handset on hook).

6.5 General advice on device interaction

Our general advice on interaction and device state is that the application should dictate what state the device is in at any given time.
The USB HID protocol, unfortunately, does not support reading the state of the headset regarding parameters such as line busy or mute, it is therefore not possible to predict with 100% certainty what state the headset is in.

Because of this, the recommended approach is to let the softphone application tell the device which state to be in when this makes sense for the flow of operation.

It could, for example, be that the softphone application always sets the mute state as ‘unmuted’ when an audio connection is opened, or maybe always joins a conference call with more than five participants in a ‘muted’ state.

6.6 Call Lock

Important note
We strongly recommend implementing this functionality to avoid interfering with other softphones.

The Call Lock functionality ensures that two (or more) softphones do not use the same Jabra device at the same time - it does so by controlling access to the device.

Please note that Call Lock is not an absolute lock; it is a semaphore to assist softphones using the same device without interfering with each other.

You acquire a ‘Call Lock’ before you perform any form of call control. The Call Lock informs other applications that use the Jabra SDK libraries to “keep hands-off”. The lock is kept for the duration of the call, once the call is finished, you release the lock to allow other softphones to use the device.

By acquiring a lock before using the call control functions softphones can co-exist without interfering with each other - if acquiring the lock fails another softphone is currently using the device.
Note that interference may still occur if the Call Lock functionality is not implemented by all softphones.

Call Lock consists of three API functions:

- Jabra_GetLock()
- Jabra_ReleaseLock()
- Jabra_IsLocked().

Use ‘Jabra_GetLock’ to acquire the lock, use ‘Jabra_ReleaseLock’ to release the lock.

‘Jabra_IsLocked’ reads the current state of the lock within your application. This can be used to check if your application already has obtained the lock; using ‘Jabra_IsLocked’ does not affect performance and has been provided as a convenient way of checking this without using ‘Jabra_GetLock’.

Please note that ‘Jabra_IsLocked’ does not check if another softphone has obtained a lock on the device. You must use ‘Jabra_GetLock’ to see if you can obtain the lock or not.

If an application crashes whilst having a call lock the lock will be released automatically allowing other softphones to use the device.

Refer to the platform-specific User Guide for further information.

7 Call control examples

This section will provide some examples of call control sequence diagrams, for more examples and details please also refer to the 'Softphone meta code' document and User Guides for each platform; all of this is available on the Jabra Developer Zone.

The sequence diagrams depict the user interaction with a device and hence the communication between the Jabra device and the application. The examples are described using a Jabra Evolve2 65 headset with a 'multi-function' button; see Appendix on page 28.

7.1 Example 1: Incoming call accepted and ended on device

This sequence diagram illustrates a situation where an incoming call is accepted by pressing the 'answer' button on the headset. The call is terminated by pressing the 'end' button on the headset. Note that 'answer' and 'end' functions are served by the same button (the 'multi-function' button), but the signal sent by the Jabra device depends on the call state of the device.

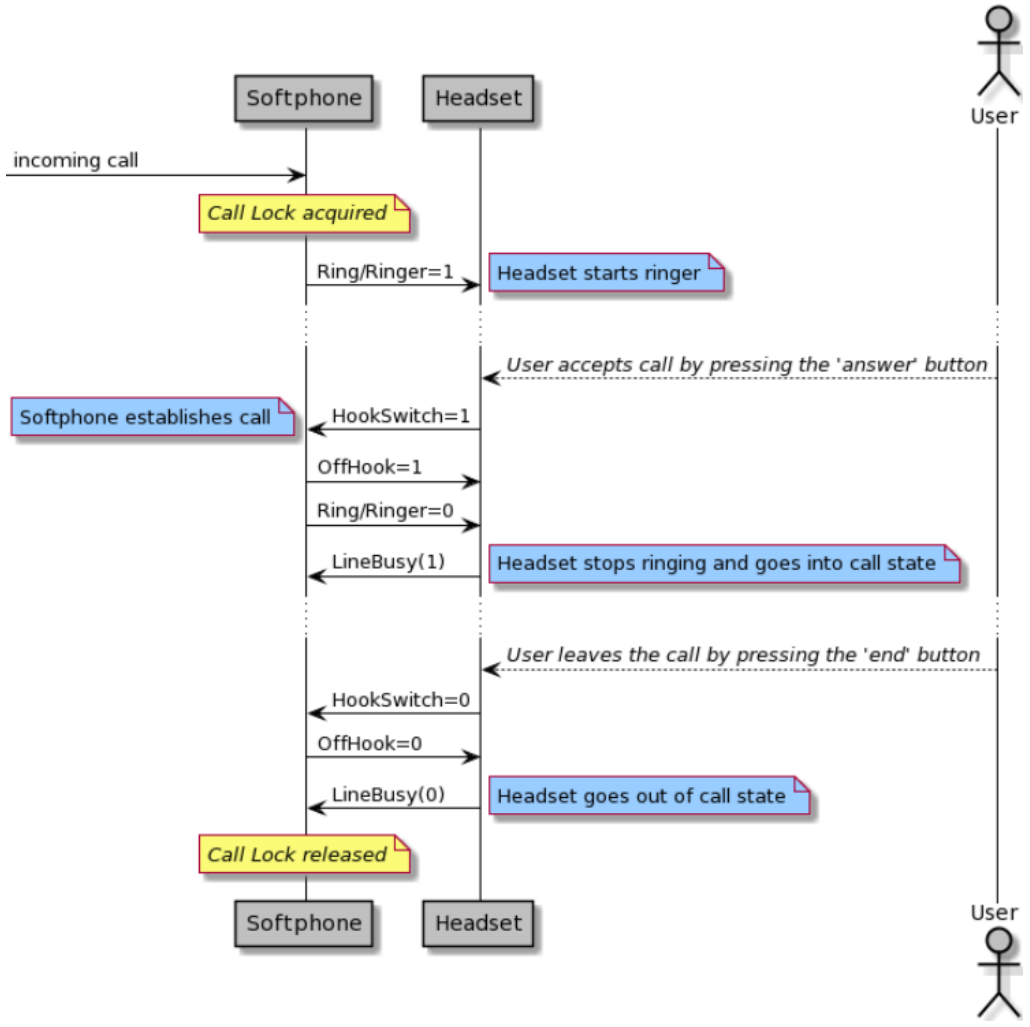

Figure 4 Sequence diagram: Incoming call accepted and ended on the device.

7.2 Example 2: Incoming call accepted on softphone GUI

This sequence diagram illustrates a situation where the user accepts an incoming call on the softphone GUI. This is similar to Example 1, but messages differ from those depicted in Figure 4 as the user accepts the call on the GUI instead of using headset buttons.

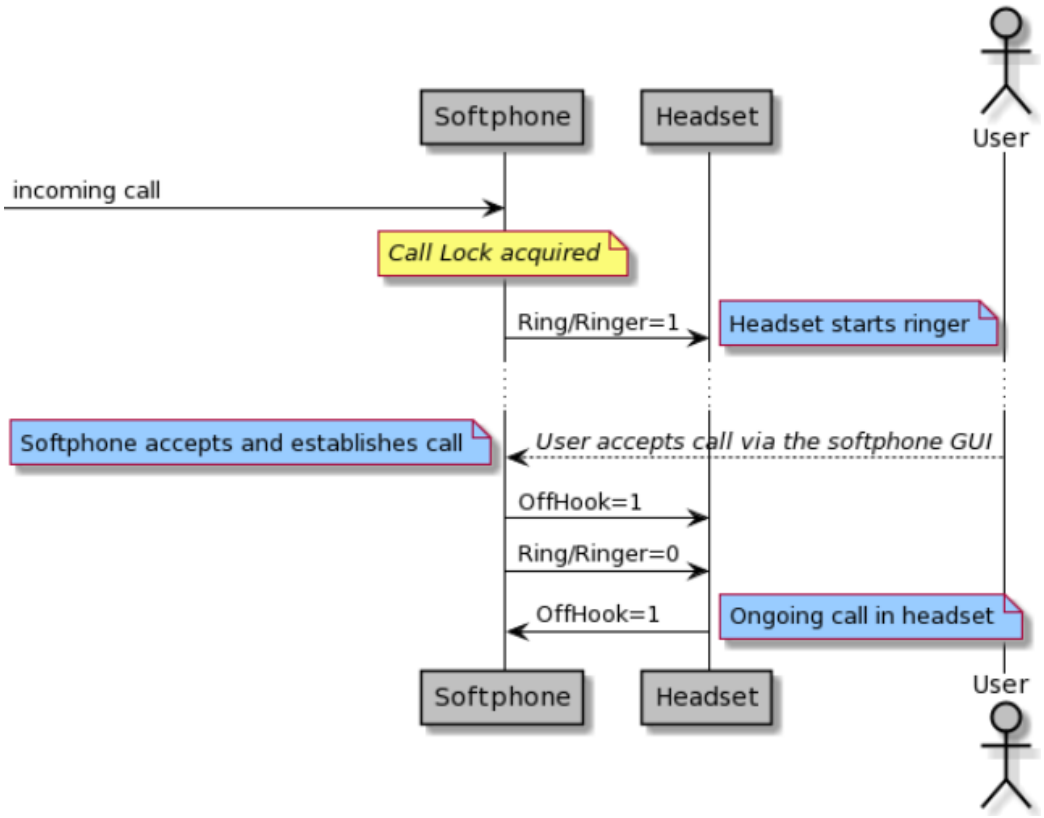

Figure 5 Sequence diagram: Incoming call accepted on softphone GUI.

7.3 Example 3: Incoming call rejected on device

This sequence diagram illustrates a situation where the user rejects a call by double-pressing the ‘multi-function’ button on the headset.

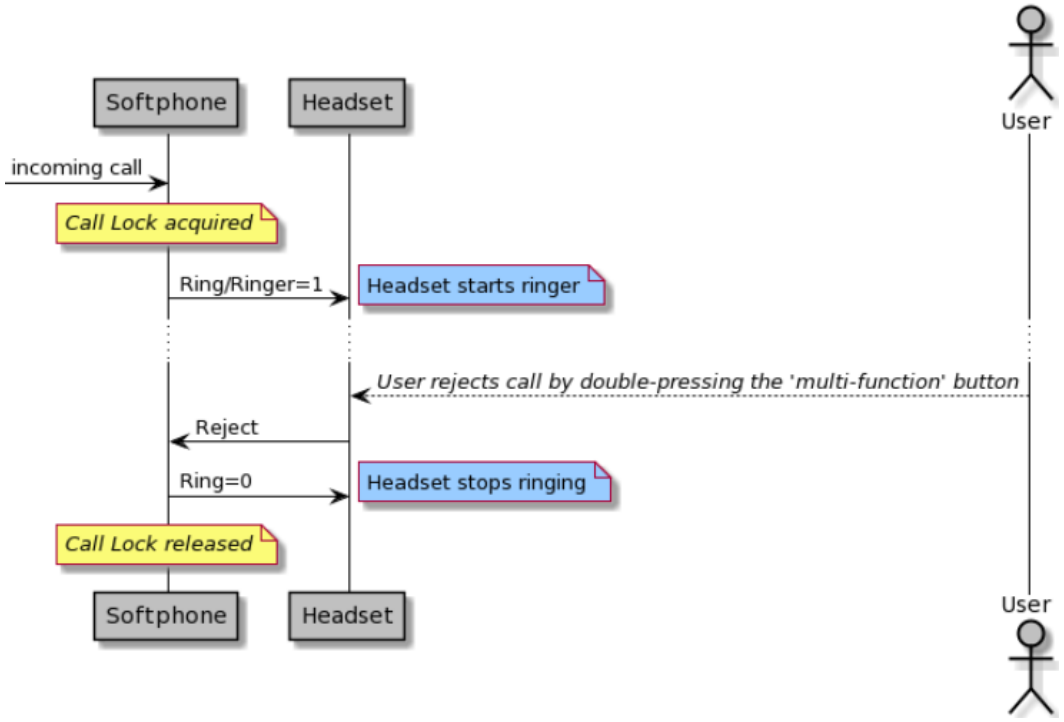

Figure 6 Sequence diagram: Incoming call rejected on the device.

8 Device-specific features

Depending on your Jabra device you can deploy features that can enhance the customer experience.
Some headsets allow you to set your own specific ringtone, some headsets have one button, some have two buttons; on some headsets you can set different busy light colors, and some headsets are just plain headsets without any fancy features.

Due to the variety of features, you must pay attention to the device-specific features of the headset(s) that you integrate into your solution.

If the headset has buttons pay attention to how many and how they work, buttons can be located either on the headset itself or on an attached control unit. If the headset enables you to turn on busy lights without being in a call you should also pay attention to this, and so forth.

The API includes all these features, it's the specific device that sets the limitations to the options you can deploy.

9 Advanced functionality and features

Basic call control is not the only functionality you can integrate, there's a lot more you can use to add value to your customers:

- Audio optimization
- Workflow support
- Real-time call-profiling data.

You can, for instance, support workflows by enabling programmable buttons and programmable lights or perform asset management such as FW updates.
Take for instance the Engage 50 headset, it has two programmable buttons: the '3 dots icon' button and the '4 dots icon' button. You can select which function you want each of these two buttons to perform when pressing them, hence you can offer your customers a customized solution that fits their specific needs.

The Engage 50 headset also supports setting different busy lights, you could for instance choose to deploy one color for "I'm in an active call" and another color for "I'm listening to the radio, but are free to take a call", or a third color for "Don't disturb, I'm busy".

You can also do telemetry; real-time audio telemetry such as talk ratio/crosstalk/silence, environmental telemetry such as measuring background noise or boom-arm position, as well as connection and device telemetry such as signal strength or battery status and level.

10 Abbreviations and Acronyms

API	*Application Programming Interface.* An application programming interface defines interactions between multiple software intermediaries. It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc.
FW	*Firmware.* Firmware is a specific class of software that provides low-level control for a device's specific hardware.
GUI	*Graphical User Interface.* A GUI is a system of interactive visual components for computer software. It displays objects that convey information and represent actions that can be taken by the user.
HID (USB HID)	*Human Interface Device.* The USB HID specifies a device class for human interface devices. For further information refer to https://www.usb.org/hid
OS	*Operating System.*
PID	*Product ID.*
SDK	*Software Development Kit.*
UC	*Unified Communication.*
VID	*Vendor ID.* Jabra device VID is 0x0B0E.

11 Appendix

Jabra SDK libraries

The purpose of the Jabra SDK library is to provide softphone applications with the ability to configure, activate, control, and monitor Jabra devices.

The main runtime component of the Jabra SDK is the core USB library. The library eases integration with Jabra devices by providing the following functional benefits:

- Handling the low-level communication of the USB protocol, hiding its complexity.
- An API at the application-programming level.
- Access to advanced digital functions not available through the standard USB HID protocol.
- A unifying layer managing differences in device protocols and capabilities.
- A mechanism for reducing the interference between multiple softphone applications.
- Logging of operation and any errors encountered at the library level.

The core Jabra USB library exposes this functionality through an API that allows interfacing to a native C/C++ application level on Windows, macOS, and Linux.

To make the functions available in other runtime environments such as node.js/Electron, or within a Browser JavaScript engine, library adapters are also provided. The runtime environment adapters are implemented as language wrappers, as well as helper libraries and gateway processes, and come packaged with example applications for the platforms.

Please see below for detailed information on each of these benefits.

Managing the complexity of USB HID-level communication

The USB HID-level interface as specified by usb.org operates on a fairly low level, where communication takes place by passing a number of bytes between the application and the device. Bit-fields within these bytes then carry messages, events, and data back and forth during an interaction between the application and the device.

The mapping of commands to bytes and bit-fields is somewhat cumbersome and may require quite extensive development and testing to cover all of the functions of a USB HID device correctly.

The Jabra core USB library builds upon many years of experience with USB HID interfacing, and significantly reduces the integration time for an application developer compared to writing low-level USB HID protocol management.

Application-level API

By encapsulating the low-level USB HID operations and providing them as named atomic function calls at the development language level, the library facilitates faster and more clear design and implementation of the application code.

This is likely to result in faster ‘time-to-code-complete’ and easier maintenance during the application's lifetime. Event-driven messages from the device are handled by callback functions, relieving the application of performing polling of USB connections.

Access to advanced digital functions

Several key digital features of Jabra devices are not accessible through the standard USB HID call control interface. This includes configuration settings such as ringer style and ringtones (on some devices) and enhanced busy-light handling.

Using the Jabra USB library also enables access to the programmable buttons on certain devices, allowing for unique integrations making special application features accessible at the touch of a button on a device.

Going beyond call control, the use of the Jabra USB libraries additionally opens the door to a wide range of telemetry data provided by newer Jabra devices. These allow for an even deeper level of device integration, providing the means for call profiling, as well as contributing to data sets for real-time analytics of workflows and device usage.

Managing differences in device protocols and capabilities

Although most Jabra devices are identical in their interface, a few minor differences do exist in cases of e.g. ringer behavior or device protocol signal structure.
The Jabra USB libraries support these device-specific differences by providing a unified API towards all supported Jabra devices.

The unified interface is implemented partly through actual code, and partly through a set of interface-definition files, or protocol manifests, that is handled by the core library.
To ensure that these files are up to date, the library will automatically try to update the protocol manifests from a network-based backend, either on-premises or from a secure server at jabra.com. As a result, updates to protocol formats of existing devices or adding new types of devices are supported without the need to update the Jabra USB library in the application.

Reducing interference between multiple applications

By default, the USB HID interface model provides non-exclusive access to a device from many applications on a client computer. This means that several applications can interact with the call control functions of a Jabra device concurrently. Unfortunately, this also leads to the risk of applications interfering with each other.
A typical example is when a user presses the ‘hook-switch’ button (answer/end button) to accept an incoming call on a softphone. The signal from the device can be interpreted by another softphone application; this application believes that the user wants to make a call. This brings the windows of the second application to the front of the GUI, blocking out the first application on which the actual call was being received.

The Jabra USB library contains functions that let a softphone application register as available and ready for telephony. Using the Jabra Direct companion application, the user can choose a ‘preferred softphone’.
By using this selection and checking if another application has already instructed the device to go ‘off-hook’, an application can respect that another application is already using the call control features of the device. The ‘Call Lock’ function described in section 6.6 on page 17 will also reduce the risk of multiple softphones using the device at the same time.

Logging of operation and any errors encountered

The Jabra SDK includes a logging mechanism that can be set to write runtime information to a set of log files. The level of information can be controlled through environment variables, enabling the change of detail level at end-user installations.

Please refer to the platform-specific User Guide on Jabra Developer Zone for details.

State diagram - on-hook/off-hook

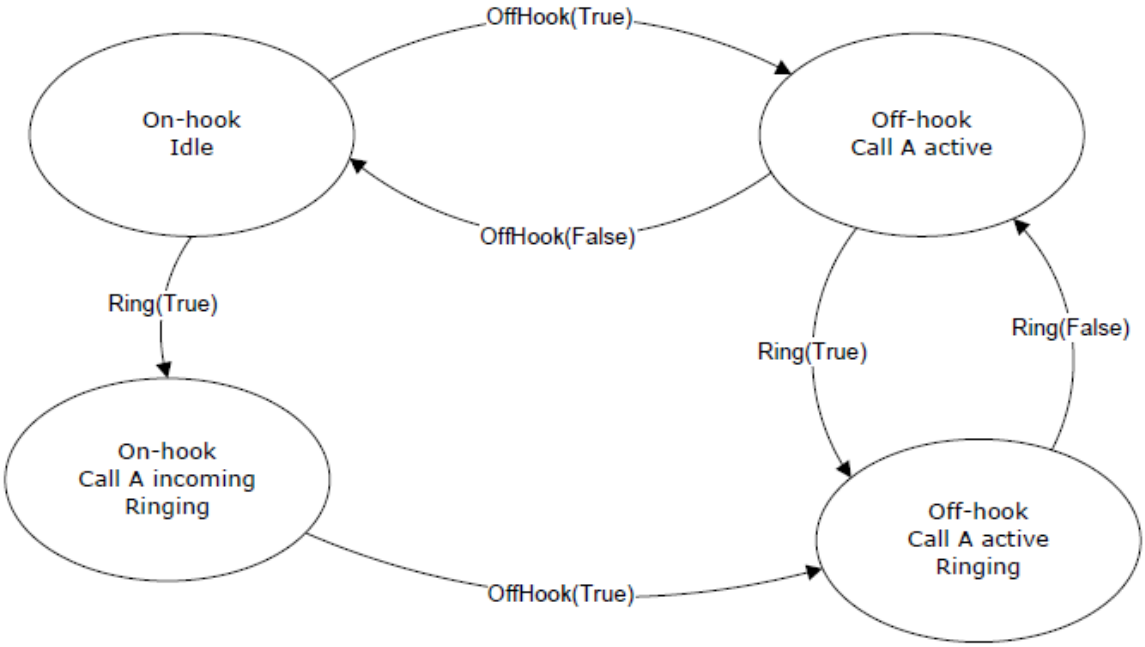

Figure 7 State diagram, on-hook/off-hook.

Multi-function button

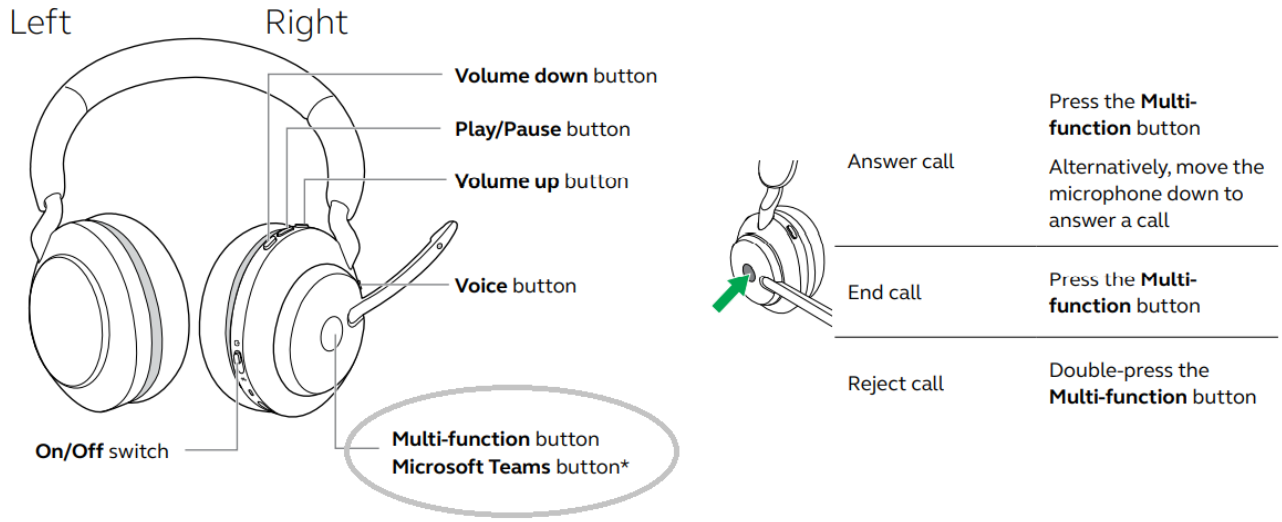

Figure 8 Multi-function button on headset (Jabra Evolve2 65 user manual)