Avaya one-X® Communicator Headset Integration

This document describes how to integrate a Headset application to the Avaya one-X® (TM) Communicator. The Headset application can control any type of headset (Blue tooth, USB, etc.). The Headset application is expected to run on the same PC as the Avaya one-X® Communicator. The communication between the Headset application and the Avaya one-X® Communicator will be done through the HeadsetInterface Apis which are described in this document.

Assumptions:

- Both the Avaya one-X® Communicator and the Headset application reside on the same PC.
- Avaya one-X® Communicator will only allow connections coming from the local host (127.0.0.1).
- The Avaya one-X® Communicator is in Road Warrior mode (otherwise the voice path does not go through the Avaya one-X® Communicator).
- The HeadsetInterface is a COM component with the following set of Apis exposed.
- All Apis return a value of HRESULT. S_OK on success, E_FAIL on error
- The Headset application needs to maintain the logic for handling multiple calls.

Introduction

The HeadsetInterface is a COM component object that the Headset Application can use to originate and control the calls using the Avaya one-X® Communicator.
The COM component provides a set of Apis and events by which the headset application can control the calls.

HeadsetInterface Apis:

Following is the list of Apis provided in the HeadsetInterface component

1. HRESULT Register()

[For registration]
The first Api the Headset application must call in order to integrate with the Avaya one-X® Communicator is Register() method.

2. HRESULT Unregister()

[For un-registration]
When the Headset application shuts down, or for any other reason needs to disconnect from the Avaya one-X® Communicator, it should call the Unregister() Api.

Any subsequent Api calls from the Headset application would fail. The only exception is a Register()Api.

3. HRESULT BlacklistEvent()

[For blacklisting optional events]
The Headset Application can request the Avaya one-X® Communicator not to send it some optional events. This is done by calling the BlackListEvent() Api with the desired event name.

The EventName is taken as a parameter. For example, if the application does not want to receive SessionUpdatedEvent, call BlacklistEvent("SessionUpdatedEvent");

Only the following events can be blacklisted:

- CallStateActiveEvent
- CallStateIdleEvent
- SessionUpdatedEvent
- SessionCreatedEvent
- ServiceInitiatedEvent

4. HRESULT MakeCall (BSTR sNumber, out short nConnectionId)

[Placing a call from the Headset application]
Using the MakeCall() Api method, the Headset application can call using the Avaya one-X® Communicator. This Api returns the unique connection-id for the new call made. This connection-id is used to identify the call.

For example : MakeCall("4452", out nConnectionId) :- This will call 4452 and store the connection-id in "nConnectionId"

Note that how the Headset application gets the phone number to call (Speech recognition etc.) is outside the scope of HeadsetInterface.

5. HRESULT HangupCall(short nConnectionId)

[For hanging up a call from the Headset application]

To hang up an existing call, use HangupCall(nConnectionId) Api. Pass the connection-id for the call to be ended.

The connection-id can be received from MakeCall() or any of the session related events.

7. HRESULT AnswerCall(short nConnectionId)

[Answering an incoming call from the Headset application]

To answer an incoming call, use AnswerCall(nConnectionId). The connection-id can be received on the IncomingSessionEvent.

8. HRESULT IgnoreCall(short nConnectionId)

[Ignoring or rejecting an incoming call]

To ignore or reject an incoming call, use IgnoreCall(nConnectionId) Api. The connection-id can be received on the IncomingSessionEvent.

Currently – only DenialCode value of 0 (Ignore) is supported.

9. HRESULT HoldCall(short nConnectionId)

[Holding a Call]

To hold an active call, use HoldCall(nConnectionId) Api. The connection-id is received from MakeCall() or SessionCreatedEvent.

10. HRESULT UnholdCall(short nConnectionId)

[Resuming a held Call]

To resume a held call, use UnholdCall(nConnectionId) Api. The connection-id is received from MakeCall() or SessionCreatedEvent or HeldEvent.

11. HRESULT MuteCall(short nAudio, short nVideo)

[Muting a call]

To mute an active call, use MuteCall(nAudio, nVideo) Api.

Using nAudio, nVideo parameters, the Headset application can specify if it wants to mute Audio only, video only ,or both. Set value 1 for mute.

12. HRESULT UnmuteCall(nAudio, nVideo)

[Un Muting a call]

To mute an active call, use MuteCall(nAudio, nVideo) Api.

Using nAudio, nVideo parameters, the Headset application can specify if it wants to mute Audio only, video only ,or both. Set value 1 for un-mute.

AVAYA

HeadsetInterface Events:

HeadsetInterface component fires call control related events. The Headset Application can control the calls not originating from the application by handling these events. Also handling these events will be useful to handle the multiple call scenarios.

These events are received for all the calls through the Avaya one-X® Communicator whether originating from the Headset Application using the above Apis or directly from the Avaya one-X® Communicator. The connection-id received in the events can be used for further call control.

Following is the list of Events that the HeadsetInterface component fires.

1. **IncomingSessionEvent**

   This event is received when there is an incoming call on the Avaya one-X® Communicator.
   The connection-Id for the incoming call is available in this event. This connection-id can be used for further call control like answer, ignore etc.

2. **SessionCreatedEvent**

   This event is received when a call was created.
   The connection-Id for the call is available in this event. This connection-id can be used for further call control like answer, ignore etc.

   This event will be received even when a call is placed from Avaya one-X® Communicator.

3. **SessionEndedEvent**

   This event is received when a call is ended at near end or remote end
   The connection-Id for the call is available in this event.

   This event will be received even when a call is ended from Avaya one-X® Communicator.

4. **SessionUpdatedEvent**

   This event is received when there is change in session.
   The connection-Id for the incoming call is available in this event. This connection-id can be used for further call control like answer, ignore etc.

5. **EstablishedEvent**

   This event is received when call established with the far end.
   The connection-Id for the incoming call is available in this event. This connection-id can be used for further call control like answer, ignore etc.

6. **HeldEvent**

   This event is received when the call is put on hold.
   The connection-Id for the held call is available in this event. This connection-id can be used for further call control like answer, ignore etc.

   This event will be received even when a call is held from Avaya one-X® Communicator.

7. UnheldEvent

This event is received when the call is resumed (un-held) from hold
The connection-Id for the held call is available in this event. This connection-id can be used for further call control like answer, ignore etc.

This event will be received even when a call is un-held from Avaya one-X® Communicator.

8. MuteStateEvent

This event is received when the call is put on mute or un-mute.
The event provides the state of the audio call.

This event will be received even when a call is muted/un-muted from Avaya one-X® Communicator.

Handling multiple call scenarios:
The Headset application can handle multiple call scenarios using the HeadsetInterface Apis and events mentioned above.
Maintaining the various connection-ids received from the Apis and the events is the main part in handling multiple calls.

Note that the following examples are related to 2 call scenarios. Same logic can be used to handle more than 2 calls.

Following are some of the multiple call scenarios.
The assumption is that there is existing active call. Connection-id for this call is maintained in nActiveCall.

1. Placing a second call

To place a second call, use MakeCall() Api with the number to be called. Maintain the connection-id received to control this call.

The existing call will be automatically put to hold

2. Answering a second incoming call

To answer a second incoming call, involves processing the IncomingSessionEvent and using the AnswerCall() Api.

The connection-id for the second incoming call will be received in IncomingSessionEvent. Store this in nIncomingCall. Use this connection-id with AnswerCall() Api to answer this call.
The first call will be automatically put on hold.

3. Switching between two existing calls

The process of switching between existing calls includes putting the active call on hold, and un-holding (resuming) the held call.

It is assumed that there are two existing calls, out of which one is active and the other is held. The call with connection-id nCallid1 is the active call and call with connection-id nCallid2 is held call.

To switch to call with nCallid2, use HoldCall(nCallid1) to put active call on hold. And then use UnholdCall(nCallid2) to resume the held call.