Audio Device Plugin Development

Swyx Solutions AG
Joseph-von-Fraunhofer-Str. 13a
44227 Dortmund (Germany)

Status: Document in process, unfinished

Page Count:	52		
Version:	1		
Created:	08.08.11	by:	pd
Modified:	12.12.11 14:22	by:	pd

Change Log

No	Date	Who	Description
1	08.08.11	PD	Created.
2	24.08.11	TA	Modified.
3	12.12.11	PD	Modified (embedded sequence diagrams)

1 Introduction

1.1 The purpose of this document

Since version 2011 R2 of SwyxWare, Swyx is providing a new interface to create your own device plugin for SwyxIt!. That enables Manufacturers of audio devices (and of course other interested software developers) to integrate their own devices easily with SwyxIt!.

It is the task of this document to give a firm introduction in the possibilities and a documentation for the development of your own Audio Device Plugin.

1.2 Target audience

This document is intended to all software developers who need to create their own plugin for a particular audio device in SwyxIt!. It is designed to be your guideline throughout the development process.

To take full advantage of that document, you should be familiar with C++ / COM (ATL) and optional: .Net / C# / VB.Net.

2 Overview

2.1 What is a device Plug-In?

Technically spoken a device plug-in is a COM server (or a managed .Net assembly) which provides a specific interface for use with the Swyx Client Line Manager application. A device plug-in communicates with one or more hardware audio devices and acts as a translator between Client Line Manager and these devices. The only constraint for the device you want to support with your own Audio Device Plug-In: It must be visible to the Windows operating system, so it must provide the default Windows Audio Device interfaces and sound channel ID's.

This documentation is about implementing your own device Plug-In for Client Line Manager and SwyxIt!.

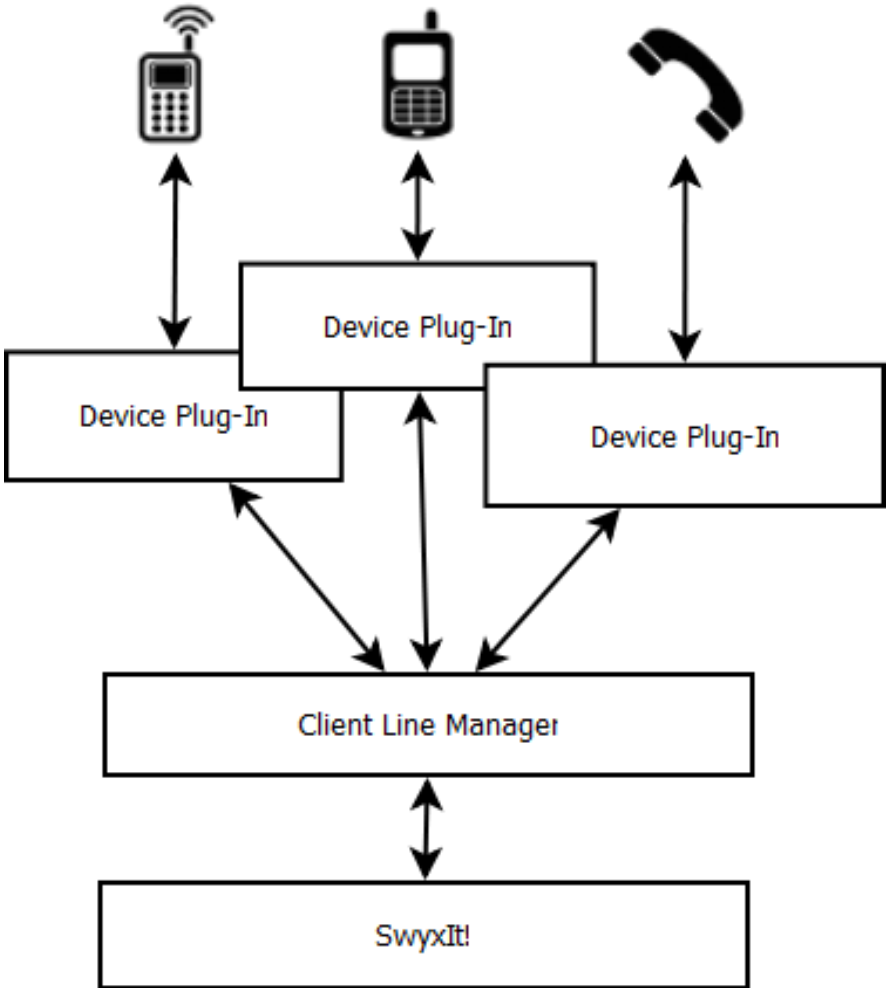
2.2 System Description

In order to create your own plugin, you should know about the context of its instantiation and where it is called from. So let's take a closer look at the system architecture.

Which software components are involved when dealing with Audio Device Plug-Ins?

- SwyxIt! (User Interface)
- Client Line Manager (Management of Communication Lines, Devices and Audio Mode Handling)
- Device Plug-Ins (providing device specific logic to Client Line Manager through a well-defined interface)

This graphic shows the relationship of the components in more detail:



2.3 SwyxIt!

SwyxIt! is the application where the User interacts with the system – this is the place where the User manages his communication and where he configures his communication devices and Audio Mode settings.

SwyxIt! is communicating with the Client Line Manager component via COM. The Client Line Manager is started / stopped by SwyxIt!.

SwyxIt! enables the User to select and configure the attached devices (e.g. Plug-In provided devices). The User is also able to assign a specific device to a dedicated Audio Mode. These settings are done via the “Local Settings” Dialog which is discussed throroughly in the SwyxIt! User Manual.

2.4 Client Line Manager

The Client Line Manager process supports SwyxIt! in background. The Client Line Manager loads and manages the registry configured Audio Device Plug-Ins and handles the real connected devices properly. It is also running Swyx defined algorithms bound to specific Audio Modes.

The lifetime of an Audio Device Plug-In instance is bound to the lifetime of the Client Line Manager process.

2.4.1 Registry settings of Client Line Manager (regarding Plug-Ins)

How is the Client Line Manager knowing about available plug-ins?

First, all COM Device Plug-Ins need to be registered to the Windows operating system.

The Client Line Manager reads the available plug-ins by enumerating the subkeys of

HKEY_LOCAL_MACHINE\SOFTWARE\[Wow6432Node]\Swyx\Client Line Manager\CurrentVersion\Options\AudioDevicePlugIns

To enable your plug-in you have to add a subkey with the version independent ProgID of your implementation of IAudioDeviceCollection as keyname. The Client Line Manager instantiates your collection-object via CoCreateInstance and this keyname as ClassID.

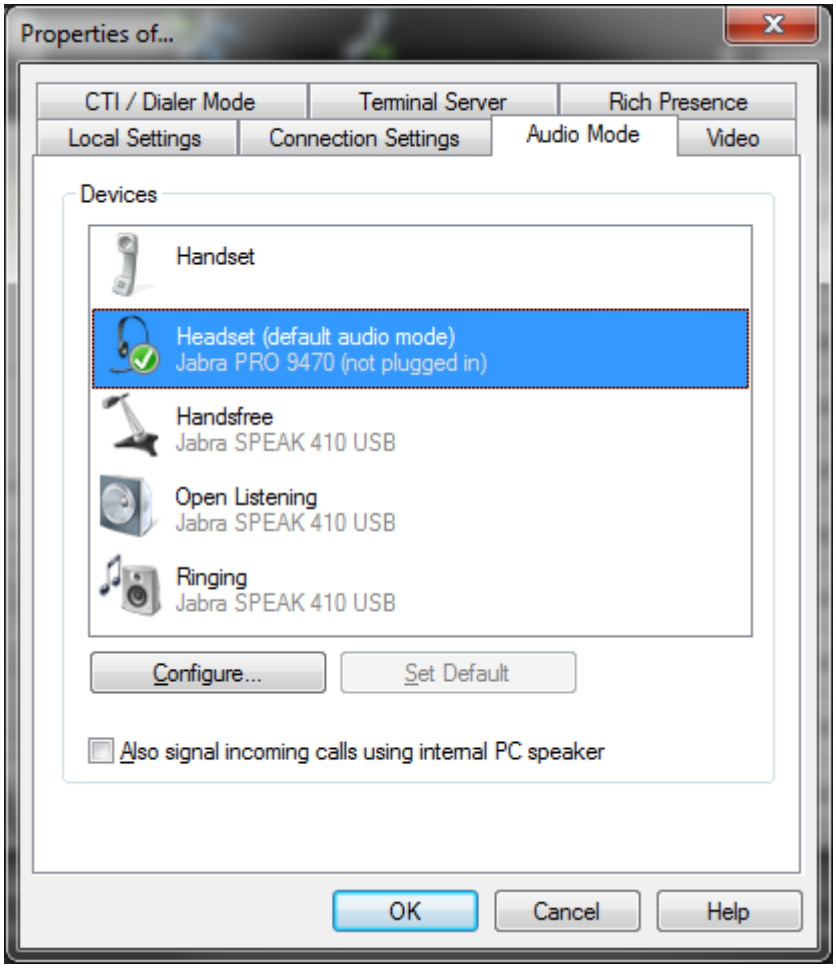
Important information: In each device sub key you can manually enable or disable the plug-in with a new Value “Enabled” of type DWORD. If the value of key “Enabled” equals **0**, the plug-in will not be loaded by Client Line Manager. If it is not **0** (or if the value is not existing) the plug-in is enabled and will be loaded when Client Line Manager starts.

2.5 Description of AudioModes

An Audio Mode is (simply spoken) the current communication and audio device configuration of SwyxIt!. It combines the used devices with specific settings and a dedicated workflow.
The user can configure the audio devices for use with the following audio modes:

Handset	Earphone device, Speaker and Mic
Headset	Headset device, Speaker and Mic
Handsfree	Speaker and Handsfree Mic
Open listening	Additional Speaker
Ringing	Speaker or Ringer

This audio modi are for GUI purposes only!



1 SwyxIt! AudioMode Device configuration

For developing purposes the AudioDevicePlugin-Library defines the enumeration *AudioMode*:

Handset	Earphone, render and capture. Best matching device formfactor: Handset
Headset	Headset, render and capture Best matching device formfactor: Headset
Handsfree	Handsfree, render and capture Best matching formfactor: Speaker and Sp. Mic
Generic	Used, if *Handset*, *Headset* or *Handsfree* does not match or don't know.
Idle	No audio mode active, but be ready to play ring-ingtones!

2.6 Description of SoundChannels

During the development of your own Plug-In, you will get in touch with the term „SoundChannel“.

First you have to distinguish between logical soundchannels as defined by the enumeration *Sound-Channel* of the Typelibrary and physical soundchannels as provided by your audio device.

Second you have to map each logical soundchannel to the best matching physical soundchannel of your device. It is up to you and your plugin, to make this decision!

Once the Client Line Manager has loaded your plugin, your plugin will be asked for the supported logical soundchannels frequently in accordance to the users's configuration.

Logical soundchannel	**Physical soundchannel**
SoundChannelCapture	Input channel. Map to physical soundchannel: • Handset mic, if AudioMode Handset is requested • Headset mic, if AudioMode Headset is requested • Handsfree mic, if AudioMode Handsfree is requested • Any mic, if AudioMode Generic is requested • No soundchannel, if AudioMode Idle is requested
SoundChannelRender	Output channel. Map to physical soundchannel: • Handset speaker, if AudioMode Handset is requested • Headset speaker, if AudioMode Headset is requested • Handsfree speaker, if AudioMode Handsfree is requested • Any speaker, if AudioMode Generic is requested • No soundchannel, if AudioMode Idle is requested
SoundChannelSpeaker	Output channel for open listening. Map to physical soundchannel: • No soundchannel, if AudioMode Handset is requested • No soundchannel, if AudioMode Headset is requested • Handsfree speaker, if AudioMode Handsfree is requested • Any speaker, if AudioMode Generic is requested • No soundchannel, if AudioMode Idle is requested
SoundChannelRinging	Output channel for ringing sound. Map to physical soundchannel: • Speaker

For your convenient you may use the AudioVolumeControl.DLL in order to adjust the volume of the desired soundchannels or you may use your own implementation of volume control.

3 Starting your own Audio Device Plug-In

3.1 Overview

Your audiodevice plugin must be a COM-Class-Library with a GUID and have to contain the following COM-Objects:

MyAudioDeviceCollection

- GUID
- Version independent ProgID
- Singleton
- Creatable
- Implements Interface: IAudioDeviceCollection

MyAudioDevice

- GUID
- NON creatable, accessible only via IAudioDeviceCollection
- Create one for each physical Audio Device
- Implements Interface: IAudioDevice
- Source interface: IAudioDeviceEvents

When starting to develop your own Audio Device Plug-In you need to decide which road you want to go. You can select between three major ways:

1. Creating a native COM Audio Device Plug-In using the COM interfaces described in this document.
2. Creating a managed Audio Device Plug-In using the Interop assemblies providing the COM interfaces as managed interfaces.
3. Creating a managed Audio Device Plug-In using the IpPbxAudioDevicePluginAPI assembly.

Let's have a closer look at these three possibilities and their pro and contra points.

Native COM Audio Device Plug-In

This is the hardest but most flexible way to create your own plug-in. Just create a COM server implementing the needed interfaces and handle all the calls by yourself. You need to have deep knowledge in COM and COM connection points.

You will find all needed components (e.g. idl files, tlb files and headers) in the AudioDevicePluginSDK solution delivered by Swyx. The documentation for the needed interfaces can be found in chapter 3 of this document.

3.2 Managed Audio Device Plug-In using the Interop assemblies

You can create a managed assembly which provides an implementation of the needed COM interfaces. Swyx delivers two Interop assemblies which will make your life easier when choosing this approach.

You will find the interfaces described in chapter 3 of this document inside the `Interop.AudioDevicePlugin.dll` and `Interop.AudioVolumeControl.dll` assemblies delivered with the AudioDevicePluginSDK solution. Just reference them in your class library project and start coding.

3.3 Managed Audio Device Plug-In using the IpPbxAudioDevicePluginAPI

This is the easiest way to create your own Audio Device plug-in. Swyx delivers a managed assembly which provides two classes which you can derive from to create your own Plug-In logic:

- AudioDeviceCollection
- AudioDevice

These two classes provide everything you need to get started. The AudioDevice class provides a default implementation of an Audio Device, providing one Render and one Capture sound channel. It is also handling the volume level control between Windows and your device (Windows XP and >= Vista only). You just have to implement a few abstract methods to provide device specific information.

If you don't want to use the internal volume control of the AudioDevice class, just override the virtual SetVolume method and handle the call yourself.

You just need to reference IpPbxAudioDevicePluginAPI.dll assembly to your class library project to get started. The classes a throroughly documented in the file "`IpPbxAudioDevicePluginAPI Document.chm`" – in combination with this file you should get fast results.

4 Interfaces

Let's start with a development documentation of the interfaces you need to implement if you want to create your own Audio Device Plug-In for SwyxIt!.

4.1 IAudioDeviceCollection Interface

Every Audio Device Plug-In needs to implement **IAudioDeviceCollection**. The Client Line Manager instantiates this class to get the instances of IAudioDevice for each connected device. The **IAudioDeviceCollection** provides a simple collection interface.

The **IAudioDeviceCollection** interface inherits from IDispatch interface. **IAudioDeviceCollection** also has the following Properties and Methods:

4.1.1 Properties and Methods

4.1.1.1 IAudioDeviceCollection::_NewEnum Method

The **_NewEnum** method returns an collection of IAudioDevice instances as an IEnumVARIANT.

Syntax

```
[propget, restricted, id(DISPID_NEWENUM)]
HRESULT _NewEnum(
        [out, retval] IUnknown** pVal
);
```

Return Value

If the method succeeds, it returns S_OK.

4.1.1.2 IAudioDeviceCollection::Item Method

The **Item** method returns the IAudioDevice instance with the specified index.

Syntax

```
[id(DISPID_VALUE)]
HRESULT Item(
        [in] VARIANT index,
        [out, retval] VARIANT* pVariant
);
```

Parameters

index [in]

The index of the requested item in collection.

pVariant [out, retval]

The instance of IAudioDevice with the specified index.

Return Value

If the method succeeds, it returns S_OK.

Remarks

If parameter index is greater as or equal to the value given by Count() method, the method will return an error.

4.1.1.3 IAudioDeviceCollection::Count Method

The **Count** method returns the current count of items in the IAudioDeviceCollection.

Syntax

```
[propget, id(1)]
HRESULT Count(
        [out, retval] long *pVal
);
```

Parameters

pVal [out, retval]

The current count of items in the IAudioDeviceCollection.

Return Value

If the method succeeds, it returns S_OK.

4.1.1.4 IAudioDeviceCollection::PluginVersion Property

The **PluginVersion** property returns the version of the plug-in as a string.

Syntax

```
[propget, id(2), helpstring("property PluginVersion")]
HRESULT PluginVersion(
	[out, retval] BSTR* pVal
);
```

Parameters

pVal [out, retval]

The version of the plug-in.

Return Value

If the method succeeds, it returns S_OK.

4.1.1.5 IAudioDeviceCollection::PluginFriendlyName Property

The PluginFriendlyName property returns a friendly name of the plug-in.

Syntax

```
[propget, id(3), helpstring("property PluginFriendlyName")]
HRESULT PluginFriendlyName(
	[out, retval] BSTR* pVal
);
```

Parameters

pVal [out, retval]

The version of the plug-in.

Return Value

If the method succeeds, it returns S_OK.

4.1.1.6 IAudioDeviceCollection::Initialize Method

The **Initialize** method is called by Client Line Manager after the plug-in was loaded. The method is called before any enumeration of your IAudioDevice instances starts, so you can initialize everything you need for your Plug-In's environment in the implementation of that function.

Syntax

```
[id(4), helpstring("Initialize")]
HRESULT Initialize(
	[in] BSTR bstrRegKey
);
```

Parameters

bstrRegKey [in]

This is the registry key of the plug-in. It can be used to read out plug-in specific information.

Return Value

If the method succeeds, it returns S_OK.

Remarks

The parameter "bstrRegKey" is the key to your device Plug-In Registry entry, so you are able to read out your own configuration for the Plug-In from the corresponding registry key here. See chapter 2.4.1 for the registry settings Client Line Manager is using – the setup of your Audio Device Plug-In can write configuration data used by your plug-in to that key.

4.1.1.7 IAudioDeviceCollection::DeviceChangedEvent Method

The **DeviceChangedEvent** method is called by Client Line Manager whenever the Windows defined WM_DEVICE_CHANGED occurs, so a Plug-In is able to handle a change in the known device list of Windows.

Syntax

```
[id(5), helpstring("DeviceChangedEvent")]
HRESULT DeviceChangedEvent();
```

Return Value

If the method succeeds, it returns S_OK.

4.1.1.8 IAudioDeviceCollection::UnInitialize Method

The UnInitialize method is called by Client Line Manager before the plug-in is freed. Within this method call you should clean up every resource used by your plug-in.

Syntax

```
[id(6), helpstring("UnInitialize")]
HRESULT UnInitialize();
```

Return Value

If the method succeeds, it returns S_OK.

4.2 IAudioDevice Interface

Provides information and methods to the Client Line Manager describing the Audio Device you want to integrate.

The **IAudioDevice** interface inherits from IDispatch interface. **IAudioDevice** also has the following Properties and Methods:

4.2.1 Properties and Methods

4.2.1.1 IAudioDevice::IsOpen Method

The **IsOpen** property tells the Client Line Manager if the Audio Device Plug-In is open (and ready) or not.

Syntax

```
[propget, id(1), helpstring("property IsOpen")]
HRESULT IsOpen(
	[out, retval] BOOL* pVal
);
```

Return Value

TRUE if the Audio Device Plug-In is ready to use, FALSE otherwise.

4.2.1.2 IAudioDevice::Open Method

The **Open** method is called by Client Line Manager to open the specific Audio Device. It is also called when the current AudioMode changes.

Syntax

```
[id(2), helpstring("method Open")]
HRESULT Open();
```

Return Value

If the method succeeds, it returns S_OK.

4.2.1.3 IAudioDevice::Close Method

The **Close** method is called by Client Line Manager to close the specific Audio Device.

Syntax

```
[id(3), helpstring("method Close")]
HRESULT Close();
```

Return Value

If the method succeeds, it returns S_OK.

Remarks

After the **Close** Method has been called, no events should be fired by Audio Device Plug-In until it is opened via the Open Method again.

4.2.1.4 IAudioDevice::VendorID Property

The **VendorID** property returns the device specific Vendor ID.

Syntax

```
[propget, id(6), helpstring("property VendorID")]
HRESULT VendorID([out, retval] ULONG* pVal);
```

Return Value

The device specific VendorID.

Remarks

If you do not want to provide a Vendor ID you can safely return 0 here.

4.2.1.5 IAudioDevice::ProductID Property

The **ProductID** property returns the device specific Product ID.

Syntax

```
[propget, id(7), helpstring("property ProductID")]
HRESULT ProductID([out, retval] ULONG* pVal);
```

Return Value

The device specific ProductID.

Remarks

If you do not want to provide a Product ID you can safely return 0 here.

4.2.1.6 IAudioDevice::FirmwareVersion Property

The **FirmwareVersion** property returns the current Version of the Firmware installed on the device.

Syntax

```
[propget, id(8), helpstring("property FirmwareVersion")]
HRESULT FirmwareVersion([out, retval] BSTR* pVal);
```

Return Value

The version of the installed Firmware on the device.

Remarks

If you do not want to provide a Firmware version you can safely return an empty string here.

4.2.1.7 IAudioDevice::FriendlyName Property

The **FriendlyName** property returns a friendly name of the device.

Syntax

```
[propget, id(9), helpstring("property FriendlyName")]
HRESULT FriendlyName([out, retval] BSTR* pVal);
```

Return Value

A friendly name of the device. This name is used on the SwyxIt! UI for your device.

4.2.1.8 IAudioDevice::GetIsAudiomodeSupported Method

The **GetIsAudiomodeSupported** method is called by Client Line Manager to determine which Audio-modes are actually supported by your device.

Syntax

```
[id(10), helpstring("GetIsAudiomodeSupported")]
HRESULT GetIsAudiomodeSupported(
       [in] AudioMode am,
       [out, retval] BOOL* pVal
);
```

Parameters

am [in]

The requested AudioMode.

pVal [out]

TRUE if AudioMode am is supported by your device, FALSE otherwise.

Return Value

If the method succeeds, it returns S_OK.

4.2.1.9 IAudioDevice::DefaultAudioMode Property

The **DefaultAudioMode** property returns a vector containing the preferred AudioModes of the de-vice.

Syntax

```
[propget, id(11), helpstring("property DefaultAudioMode")]
HRESULT DefaultAudioMode([out, retval] VARIANT* pVal);
```

Return Value

A vector with elements of type AudioMode the device is preferred for.

Remarks

This value is used by Client Line Manager to automatically configure your device.

4.2.1.10 IAudioDevice::SetAudioMode Method

The **SetAudioMode** method is called by Client Line Manager when the current AudioMode changes.

Syntax

```
[id(12), helpstring("SetAudioMode")]
HRESULT SetAudioMode(
        [in] AudioMode newVal,
        [out, retval] BOOL* pReEstablishMedia
);
```

Parameters

newVal [in]

The newly set AudioMode value.

pReEstablishMedia [out]

TRUE if the SoundChannel ID's are changing for your device regarding the new AudioMode, FALSE otherwise.

Return Value

If the method succeeds, it returns S_OK.

Remarks

Return TRUE in *pReEstablishMedia* if the new OpenListening state is changing the result of a call to GetSoundChannelID. If the new state of OpenListening is not changing anything for your device (regarding SoundChannels) return FALSE.

4.2.1.11 IAudioDevice::SetDisplayHookState Method

The **SetDisplayHookState** method is called by Client Line Manager when the device should show a specific hook state.

Syntax

```
[id(13), helpstring("SetDisplayHookState")]
HRESULT SetDisplayHookState(
	[in] BOOL bHookedOff
);
```

Parameters

bHookedOff [in]

TRUE if the device should show a hooked off state, FALSE if the device should show a hooked on state.

Return Value

If the method succeeds, it returns S_OK.

Remarks

If your device is not able to visualize hook states you can safely ignore this call in your Plug-In implementation.

4.2.1.12 IAudioDevice::SetDisplayCallerInfo Method

The **SetDisplayCallerInfo** method is called by Client Line Manager when the device should show a Caller Id.

Syntax

```
[id(14), helpstring("SetDisplayCallerInfo")]
HRESULT SetDisplayCallerInfo(
	[in] BSTR bstrCallerInfo
);
```

Parameters

bstrCallerInfo [in]

The Caller Id which should be shown on the device.

Return Value

If the method succeeds, it returns S_OK.

Remarks

If your device is not able to visualize caller Id's you can safely ignore this call in your Plug-In implementation.

4.2.1.13 IAudioDevice::SetDisplayMuteState Method

The **SetDisplayMuteState** method is called by Client Line Manager whenever the device should show a specific mute state.

Syntax

```
[id(15), helpstring("SetDisplayMuteState")]
HRESULT SetDisplayMuteState(
        [in] BOOL bMuted
);
```

Parameters

bMuted [in]

TRUE if the device should display a muted state, FALSE if the device should display a not muted state.

Return Value

If the method succeeds, it returns S_OK.

Remarks

If your device is not able to visualize mute states you can safely ignore this call in your Plug-In implementation.

4.2.1.14 IAudioDevice::StartRinger Method

The **StartRinger** method is called by Client Line Manager whenever the device should start its additional ringer – this is the case if an incoming call is available.

Syntax

```
[id(16), helpstring("StartRinger")]
HRESULT StartRinger(
	[in] BOOL bRingingDevice,
	[in] BOOL bSilent
);
```

Parameters

bRingingDevice [in]

TRUE if the device is currently configured as a Ringing Device, FALSE if not.

bSilent[in]

TRUE if the acoustic ringer should be disabled starting an optical signal only, FALSE if not.

Return Value

If the method succeeds, it returns S_OK.

Remarks

If your device is not able to enable an additional ringer you can safely ignore this call in your Plug-In implementation.

If parameter bRingingDevice is TRUE, the device is configured as a Ringing Device in SwyxIt!. This can only happen, if you support a ringing channel on your implementation. This parameter indicates if you are really configured to use the ringing channel when the call to SetRinger happens. If you are not the configured ringing device, you must decide whether you start a maybe existing “additional” ringer (maybe because your device needs the “ringing” state for some internal reason) or to ignore this call to SetRinger.

4.2.1.15 IAudioDevice::StopRinger Method

The **StopRinger** method is called by Client Line Manager whenever the device should stop its additional ringer.

Syntax

```
[id(20), helpstring("StopRinger")]
HRESULT StopRinger();
```

Return Value

If the method succeeds, it returns S_OK.

4.2.1.16 IAudioDevice:: GetIsSoundChannelSupported Method

The **GetIsSoundChannelSupported** method is called by Client Line Manager to ask the Plug-In if a specific combination of AudioMode and SoundChannel is supported.

Syntax

```
[id(17), helpstring("GetIsSoundChannelSupported")]
HRESULT GetIsSoundChannelSupported(
        [in] AudioMode am,
        [in] SoundChannel sc,
        [out, retval] BOOL* pVal
);
```

Parameters

am [in]

The requested AudioMode value.

sc [in]

The requested SoundChannel value.

pVal [out, retval]

TRUE if the device is supporting the given am / sc combination, FALSE otherwise.

Return Value

If the method succeeds, it returns S_OK.

4.2.1.17 IAudioDevice::GetSoundChannelID Method

The **GetSoundChannelID** method is called by Client Line Manager to get the SoundChannelID of a specific SoundChannel this device owns.

Syntax

```
[id(18), helpstring("GetSoundChannelID")]
HRESULT GetSoundChannelID(
	[in] SoundChannel sc,
	[out, retval] BSTR* pVal
);
```

Parameters

sc [in]

The requested SoundChannel value.

pVal [out, retval]

The SoundChannel ID.

Return Value

If the method succeeds, it returns S_OK.

Remarks

The sound channel ID is the Windows OS specific device GUID in format "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx".

4.2.1.18 IAudioDevice:: GetDefaultVolume Method

The **GetDefaultVolume** method returns the default volume value for the given AudioMode / SoundChannel combination.

Syntax

```
[id(19), helpstring("GetDefaultVolume")]
HRESULT GetDefaultVolume(
	[in] AudioMode am,
	[in] SoundChannel sc,
	[out, retval] USHORT* pVal
);
```

Parameters

am [in]

The requested AudioMode value.

sc [in]

The requested SoundChannel value.

pVal [out, retval]

The absolute percentage volume value.

Return Value

If the method succeeds, it returns S_OK.

4.2.1.19 IAudioDevice::SetVolume Method

The **SetVolume** method is called by Client Line Manager to inform the Plug-In of a new volume value for the specified SoundChannel.

Syntax

```
[id(21), helpstring("SetVolume")]
HRESULT SetVolume(
       [in] SoundChannel sc,
       [in] USHORT Volume
);
```

Parameters

sc [in]

The SoundChannel value.

Volume [in]

The new absolute percentage volume value for the specified SoundChannel.

Return Value

If the method succeeds, it returns S_OK.

4.2.1.20 IAudioDevice::SetOpenListening Method

The SetOpenListening method is called by Client Line Manager to inform the Plug-In of a new Open-Listening state.

Syntax

```
[id(22), helpstring("SetOpenListening")]
HRESULT SetOpenListening(
	[in] BOOL bEnabled,
	[out, retval] BOOL* pReEstablishMedia
);
```

Parameters

bEnabled [in]

TRUE if OpenListening is enabled, FALSE if OpenListening is disabled.

pReEstablishMedia [out, retval]

TRUE if media should be re-established, FALSE otherwise.

Return Value

If the method succeeds, it returns S_OK.

Remarks

Return TRUE in pReEstablishMedia if the new OpenListening state is changing the result of a call to GetSoundChannelID. If the new state of OpenListening is not changing anything for your device (regarding SoundChannels) return FALSE.

4.2.1.21 IAudioDevice:: UseSoftwareEchoCancellation Method

The **UseSoftwareEchoCancellation** method is called by Client Line Manager to ask the Plug-In if echo cancellation should be used for a particular SoundChannel.

Syntax

```
[id(23), helpstring("UseSoftwareEchoCancellation")]
HRESULT UseSoftwareEchoCancellation(
	[in] SoundChannel sc,
	[out, retval] BOOL* pbUse
);
```

Parameters

sc [in]

The requested SoundChannel.

pbUse [out, retval]

TRUE if echo cancellation should be enabled for the SoundChannel sc, FALSE if it should not be used for SoundChannel sc.

Return Value

If the method succeeds, it returns S_OK.

4.2.1.22 IAudioDevice:: GetMediaBufferCount Method

The **GetMediaBufferCount** method is called by Client Line Manager to determine the buffer byte count for the media render buffer.

Syntax

```
[id(26), helpstring("GetMediaBufferCount")]
HRESULT GetMediaBufferCount(
        [in] AudioMode am,
        [in] SoundChannel sc,
        [out, retval] LONG* pVal
);
```

Parameters

am [in]

The requested AudioMode value.

sc [in]

The requested SoundChannel value.

pVal [out, retval]

The media buffer byte count for the given am / sc combination.

Return Value

If the method succeeds, it returns S_OK.

4.2.1.23 IAudioDevice:: EnumSoundChannelIDs Method

The **EnumSoundChannelIDs** method is called by Client Line Manager to get a list of all SoundChannel ID's used by your device.

Syntax

```
[id(27), helpstring("EnumSoundChannelIDs")]
HRESULT EnumSoundChannelIDs(
	[out, retval] VARIANT* pVal
);
```

Parameters

pVal [out, retval]

A safe array containing all SoundChannel ID's (strings) used by your device.

Return Value

If the method succeeds, it returns S_OK.

Remarks

You can use the IAudioVolumeControl COM Component made by Swyx Solutions to enumerate all SoundChannel(IDs) used by your device.

4.3 _IAudioDeviceEvents Interface

The **_IAudioDeviceEvents** interface is the event sink interface for IAudioDevice implementations. You need to fire the events defined in that interface to inform Client Line Manager whenever something changes in your device related to these events.

4.3.1.1 _IAudioDeviceEvents::OnHook Event

The **OnHook** event should be fired by Plug-In whenever the hook state changed on the device.

Syntax

```
[id(1), helpstring("HookChanged event from audio device.")]
HRESULT OnHook(
	BOOL bHookOff,
	AudioMode mode
);
```

Parameters

bHookOff

TRUE if the event is a hook off event, FALSE otherwise.

mode

The audio mode identifier for that hook event (describes the device which the hook event was meant for).

Return Value

If the method succeeds, it returns S_OK.

4.3.1.2 _IAudioDeviceEvents::OnDetached Event

The **OnDetached** event should be fired whenever the device is detached from system.

Syntax

```
[id(2), helpstring("Device detached from system.")]
HRESULT OnDetached();
```

Return Value

If the method succeeds, it returns S_OK.

4.3.1.3 _IAudioDeviceEvents::OnAudioEnabled Event

The **OnAudioEnabled** event should be fired whenever the audio link is enabled for your device.

Syntax

```
[id(3), helpstring("Audio enabled on or off.")]
HRESULT OnAudioEnabled(BOOL bAudioEnabled);
```

Parameters

bAudioEnabled

TRUE if the audio link is enabled, FALSE otherwise.

Return Value

If the method succeeds, it returns S_OK.

4.3.1.4 _IAudioDeviceEvents::OnMute Event

The **OnMute** event should be fired whenever the device is muted or unmuted.

Syntax

```
[id(4), helpstring("Microphone mute.")]
HRESULT OnMute(BOOL bMute);
```

Parameters

bMute

TRUE if the device is muted, FALSE otherwise.

Return Value

If the method succeeds, it returns S_OK.

4.3.1.5 _IAudioDeviceEvents::OnVolume Event

The **OnVolume** event should be fired whenever the volume level for your devices changes.

Syntax

```
[id(5), helpstring("Volume changed event.")]
HRESULT OnVolume(
        SoundChannel channel,
        SHORT Level,
        AudioLevelPercentType audioLevelPercentType
);
```

Parameters

channel

The SoundChannel for which the volume had changed.

Level

The new volume level in percentage. Depending on parameter *audioLevelPercentType* this value can be negative.

audioLevelPercentType

Indicates how the value of Level must be handled. If the value is “AudioLevelPercentAbso lute” Level must contain a positive percentage value. If the value is “AudioLevelPercentDelta” the Level value is handled as an increment (a decrement) value of the current volume level.

Return Value

If the method succeeds, it returns S_OK.

Remarks

With this event you can create a device specific volume handling. Example: If your device has its own volume buttons (increase volume and decrease volume) you can easily fire this event with a delta value of e.g. +5 (increase volume button) or -5 (decrease volume button) and provide the value AudioLevelPercentDelta with the call – so Client Line Manager knows what to do.

4.3.1.6 _IAudioDeviceEvents::OnHeadsetState Event

The **OnHeadsetState** event should be fired by your device whenever the user switches the communication actively to the headset of the device. This is the case for devices which owns a headset as an additional communication option.

Syntax

```
[id(6), helpstring("Headset weared.")]
HRESULT OnHeadsetState(
        BOOL bWeared
);
```

Parameters

bWeared

TRUE if the headset is weared and should be used, FALSE otherwise.

Return Value

If the method succeeds, it returns S_OK.

4.3.1.7 _IAudioDeviceEvents::OnReInitialisation Event

The **OnReInitialisation** event should be fired by your device whenever it needs to be initialized again.

Syntax

```
[id(7), helpstring("Reinitialisation of device requested.")]
HRESULT OnReInitialisation();
```

Return Value

If the method succeeds, it returns S_OK.

4.3.1.8 _IAudioDeviceEvents::OnSpeakerKey Event

The **OnSpeakerKey** event should be fired by your device whenever the speaker key (which enables an integrated speaker on your device) is pressed on your device.

Syntax

```
[id(8), helpstring("Speakerkey pressed or not.")]
HRESULT OnSpeakerKey(
        BOOL bDown
);
```

Parameters

bDown

TRUE if the speaker key is down, FALSE if it is up.

Return Value

If the method succeeds, it returns S_OK.

4.3.1.9 _IAudioDeviceEvents::OnTraceLine Event

The **OnTraceLine** event should be fired by your plug-in every time you need to trace a line. This event is very important for error and functionality tracing of your plug-in.

Syntax

```
[id(9), helpstring("Tracing purposes.")]
HRESULT OnTraceLine(
        BSTR bstrLine
);
```

Parameters

bstrLine

The string you want to trace.

Return Value

If the method succeeds, it returns S_OK.

Remarks

The traces will be written to the Client Line Manager Log file.

4.3.1.10 _IAudioDeviceEvents::OnRejectCall

The **OnRejectCall** event should be fired by your plug-in every time you want to reject an incoming call.

Syntax

```
[id(10), helpstring("Device rejected incoming call.")]
HRESULT OnRejectCall();
```

Return Value

If the method succeeds, it returns S_OK.

5 AudioVolumeControl COM-Component

The AudioVolumeControl COM component is a useful tool to handle Audio Devices and Sound Levels independent from operating system. Audio devices and their settings are handled differently between Windows XP and Windows Vista upwards. This control handles the differences transparently for you. Nevertheless under Windows XP it is possible to give the AudioVolumeControl hints how to treat the soundchannels for enhanded flexibility.

This documentation chapter describes the interfaces offered by AudioVolumeControl COM component.

5.1 IAudioVolumeControlCollection Interface

Provides methods to enumerate all Audio Devices known to the operating system.

The **IAudioVolumeControlCollection** interface inherits from IDispatch interface. **IAudioVolumeControlCollection** has the following Properties and Methods:

5.1.1 Properties and Methods

5.1.1.1 IAudioVolumeControlCollection::Initialize Method

Call this method to initialize the **IAudioVolumeControlCollection**. You can provide a device (friendly) name as a parameter so the collection will only contain items which matches the given name.

Syntax

```
[id(1), helpstring("Initialize")]
HRESULT Initialize(
        BSTR bstrRequestedDevice
);
```

Parameters

bstrRequestedDevice [in]

The requested device to get Sound Channels for. It can be a part of a name or the complete name. Provide an empty string to get all known Audio Devices from the system.

Return Value

If the method succeeds, it returns S_OK.

Remarks

The *bstrRequestedDevice* parameter can be used to filter the Audio Device list for a particular device name. The friendly name is used to determine the owner of an audio device. You can also provide a part of the name – the collection will contain all audio devices which matches the name or a part of it.

5.1.1.2 IAudioVolumeControlCollection::InitializeEx Method

Call this method to initialize the **IAudioVolumeControlCollection**. You can provide a device (friendly) name as a parameter so the collection will only contain items which matches the given name. Additionally you can specify the desired volume control.

Syntax

```
[id(1), helpstring("Initialize")]
HRESULT Initialize(
       [in] BSTR bstrRequestedDevice,
       [in] BOOL AdjustMasterVolume,
       [out, retval] BOOL *pbMixerAPI
);
```

Parameters

bstrRequestedDevice [in]

The requested device to get Sound Channels for. It can be a part of a name or the complete name. Provide an empty string to get all known Audio Devices from the system.

AdjustMasterVolume[in]

Applies only under Windows XP. If set, the AudioVolumeControl will use the Master Volume control to adjust the volume. Otherwise the Wave In Volume Control.

Return Value

TRUE, if under Windows XP, otherwise FALSE.

Remarks

See Initialize.

5.1.1.3 IAudioVolumeControlCollection::UnInitialize Method

Call this method to uninitialize the **IAudioVolumeControlCollection**.

Syntax

```
[id(2), helpstring("UnInitialize")]
HRESULT UnInitialize();
```

Return Value

If the method succeeds, it returns S_OK.

5.1.1.4 IAudioVolumeControlCollection::Count Property

The **Count** property returns the count of IAudioVolumeControl instances in the collection.

Syntax

```
[propget, id(3)]
HRESULT Count(
        [out, retval] long *pVal
);
```

Return Value

The count of IAudioVolumeControl instances in the collection.

5.1.1.5 IAudioVolumeControlCollection:: GetAudioVolumeControl Method

The **GetAudioVolumeControl** Method returns the IDispatch of an IAudioVolumeControl of the collection.

Syntax

```
[id(4)]
HRESULT GetAudioVolumeControl(
        [in] ULONG Index,
```

```
        [out, retval] IDispatch **pDisp
);
```

Parameters

Index [in]

The requested index of the collection item.

pDisp [out, retval]

The item with index *Index*.

Return Value

If the method succeeds, it returns S_OK.

5.2 IAudioVolumeControl Interface

The **IAudioVolumeControl** interface provides methods and properties to handle volume levels of a particular soundchannel of an audio device known by the system.
You can only access objects of this type via the method

IAudioVolumeControlCollection::GetAudioVolumeControl.

The **IAudioVolumeControl** interface inherits from IDispatch interface. **IAudioVolumeControl** has the following Properties and Methods:

5.2.1 Properties and Methods

5.2.1.1 IAudioVolumeControl::FriendlyName Property

The **FriendlyName** property contains the friendly name of the Windows audio device represented by that instance of IAudioVolumeControl.

Syntax

```
[propget, id(1), helpstring("property FriendlyName")]
HRESULT FriendlyName(
        [out, retval] BSTR* pVal
);
```

Return Value

The friendly Name of the audio device.

5.2.1.2 IAudioVolumeControl::GUID Property

The **GUID** property contains the GUID of the Windows audio device represented by that instance of IAudioVolumeControl.

Syntax

```
[propget, id(2), helpstring("property GUID")]
HRESULT GUID(
	[out, retval] GUID* pVal
);
```

Return Value

The GUID of the audio device.

5.2.1.3 IAudioVolumeControl::FormFactor Property

The **FormFactor** property contains the form factor of the Windows audio device represented by that instance of IAudioVolumeControl.

Syntax

```
[propget, id(3), helpstring("property FormFactor")]
HRESULT FormFactor(
	[out, retval] AudioDeviceFormFactor* pVal
);

[propput, id(3), helpstring("property FormFactor")]
HRESULT FormFactor(
	[in] AudioDeviceFormFactor Val
);
```

Return Value

The FormFactor of the audio device. See enum AudioDeviceFormFactor for more information.

5.2.1.4 IAudioVolumeControl::DataFlow Property

The **DataFlow** property contains the data flow of the Windows audio device represented by that instance of IAudioVolumeControl.

Syntax

```
[propget, id(4), helpstring("property DataFlow")]
HRESULT DataFlow(
	[out, retval] AudioDeviceDataFlow* pVal
);

[propput, id(4), helpstring("property DataFlow")]
HRESULT DataFlow(
	[in] AudioDeviceDataFlow Val
);
```

Return Value

The FormFactor of the audio device. See enum AudioDeviceFormFactor for more information.

5.2.1.5 IAudioVolumeControl::VolumePercent Property

The **VolumePercent** property allows you to get and set the volume level for that audio device in percentage values.

Syntax

```
[propget, id(5), helpstring("property VolumePercent")]
HRESULT VolumePercent(
	[out, retval] BYTE* pVal
);

[propput, id(5), helpstring("property VolumePercent")]
HRESULT VolumePercent(
	[in] BYTE newVal
);
```

Return Value

The percentage value of the volume level for that particular device.

5.2.1.6 IAudioVolumeControl::Volume Property

The **Volume** Property allows you to set and get the normalized volume level for that particular device.

Syntax

```
[propget, id(6), helpstring("property Volume")]
HRESULT Volume(
	[out, retval] FLOAT* pVal
);

[propput, id(6), helpstring("property Volume")]
HRESULT Volume(
	[in] FLOAT newVal
);
```

Return Value

The value of the normalized volume level for that particular device.

5.2.1.7 IAudioVolumeControl::Mute Property

The **Mute** property allows you to set and get the Mute state of the Windows audio device represented by that instance of IAudioVolumeControl.

Syntax

```
[propget, id(7), helpstring("property Mute")]
HRESULT Mute(
	[out, retval] BOOL* pVal
);

[propput, id(7), helpstring("property Mute")]
HRESULT Mute(
	[in] BOOL newVal
);
```

Return Value

The current Mute state for that particular device.

5.2.1.8 IAudioVolumeControl::SetMux Method

Applies only under Windows XP. The **SetMux** Method allows to switch between several microphones of the audio device.

Syntax

```
[id(8), helpstring("SetMux")]
HRESULT SetMux(
	[in] ULONG InputIndex
);
```

Parameters

ULONG [in]

The index of the microphone mixer line.

Return Value

If the method succeeds, it returns S_OK.

5.2.1.9 IAudioVolumeControl::ModuleName Property

The **ModuleName** property contains the module name of the Windows audio device represented by that instance of IAudioVolumeControl.

Syntax

```
[propget, id(9), helpstring("property ModuleName")]
HRESULT ModuleName(
	[out, retval] BSTR* pVal
);
```

Return Value

The module name of that audio device.

5.2.1.10 IAudioVolumeControl::GUIDAsString Property

The **GUIDAsString** property contains the GUID property value as a string.

Syntax

```
[propget, id(10), helpstring("property GUIDAsString")]
HRESULT GUIDAsString(
	[out, retval] BSTR* pVal
);
```

Return Value

The GUID property of that audio device as a string.

5.2.1.11 IAudioVolumeControl::Index Method

Applies only under Windows XP. The **Index** Method allows to query the index of the mixer line.

Syntax

```
[propget, id(11), helpstring("property Index")]
HRESULT Index(
	[out, retval] LONG* pVal
);
```

Parameters

Return Value

The index of the mixer line.

5.2.1.12 IAudioVolumeControl::SetSourceLine Method

Applies only under Windows XP. The **SetSourceLine** Method allows to set the desired mixer source line of the volume control.

Syntax

```
[id(12), helpstring("SetSourceLine")]
HRESULT SetSourceLine(
	[in] ULONG LineIndex
);
```

Parameters

ULONG [in]

The index of the mixer source line.

Return Value

If the method succeeds, it returns S_OK.

5.2.1.13 IAudioVolumeControl::SetSourceMute Method

Applies only under Windows XP. The **SetSourceMute** Method allows to set the desired mixer source line of the mute control.

Syntax

```
[id(13), helpstring("SetSourceMute")]
HRESULT SetSourceMute(
	[in] ULONG LineIndex
);
```

Parameters

ULONG [in]

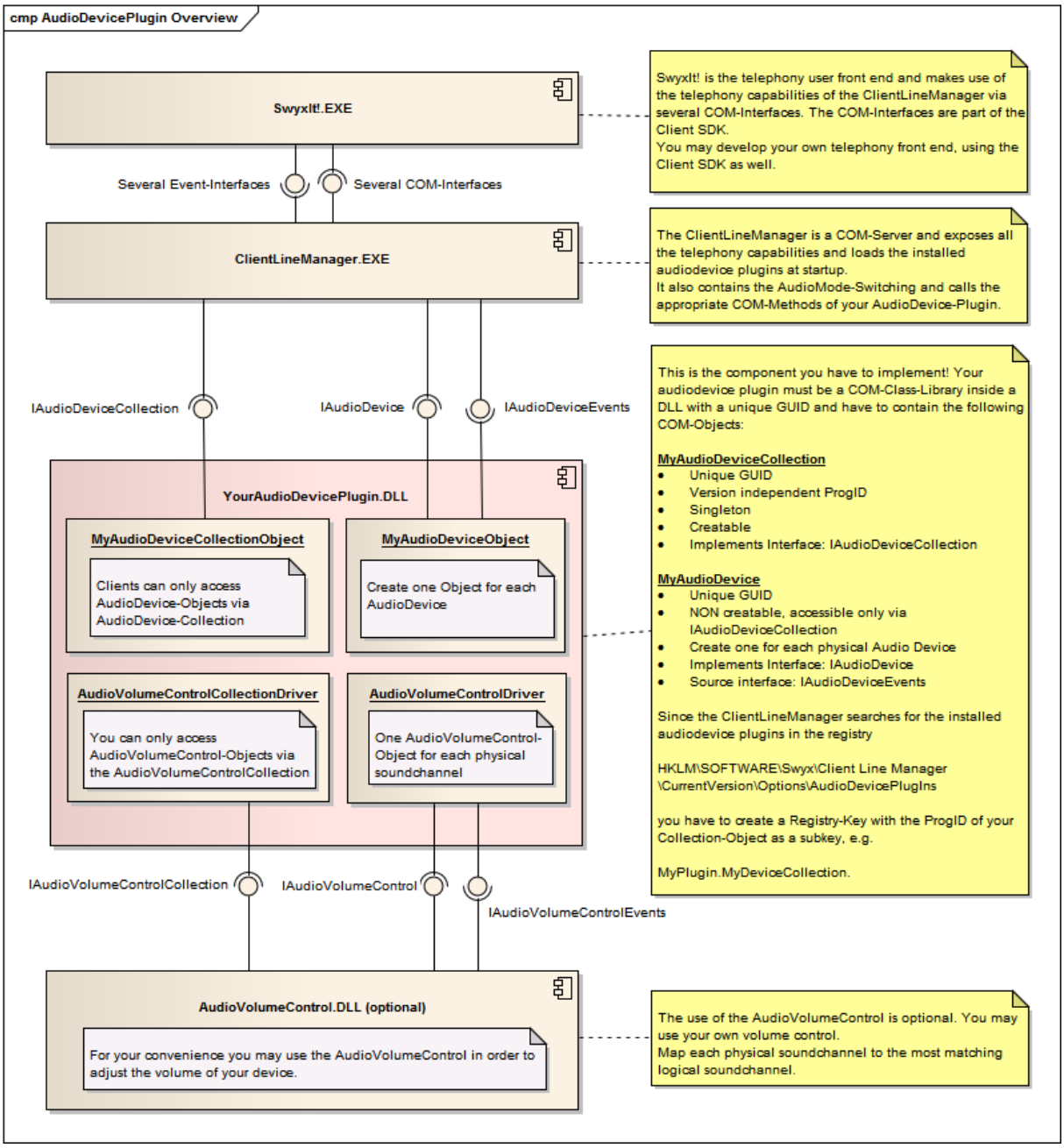
The index of the mixer source line.

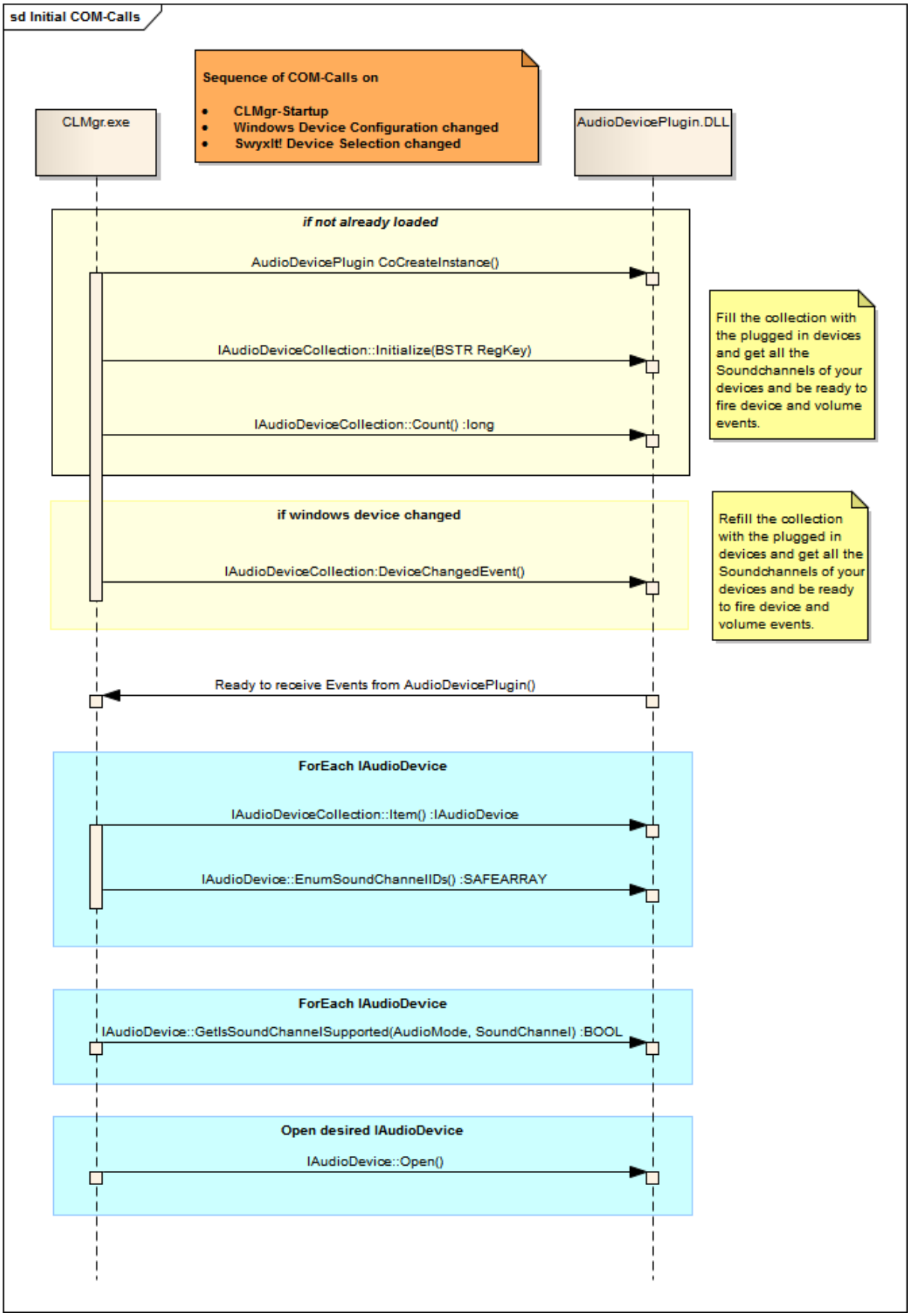
Return Value

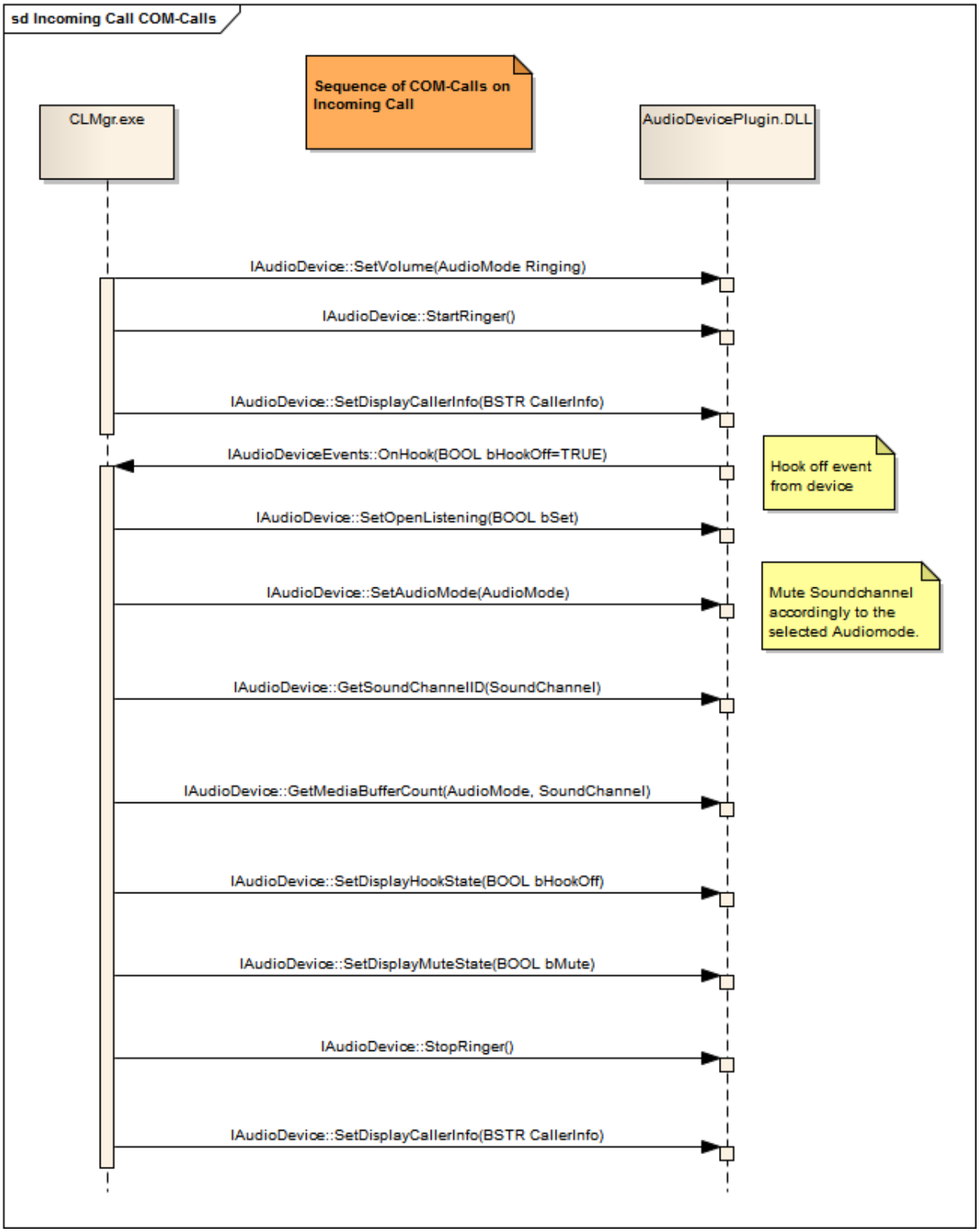
If the method succeeds, it returns S_OK.

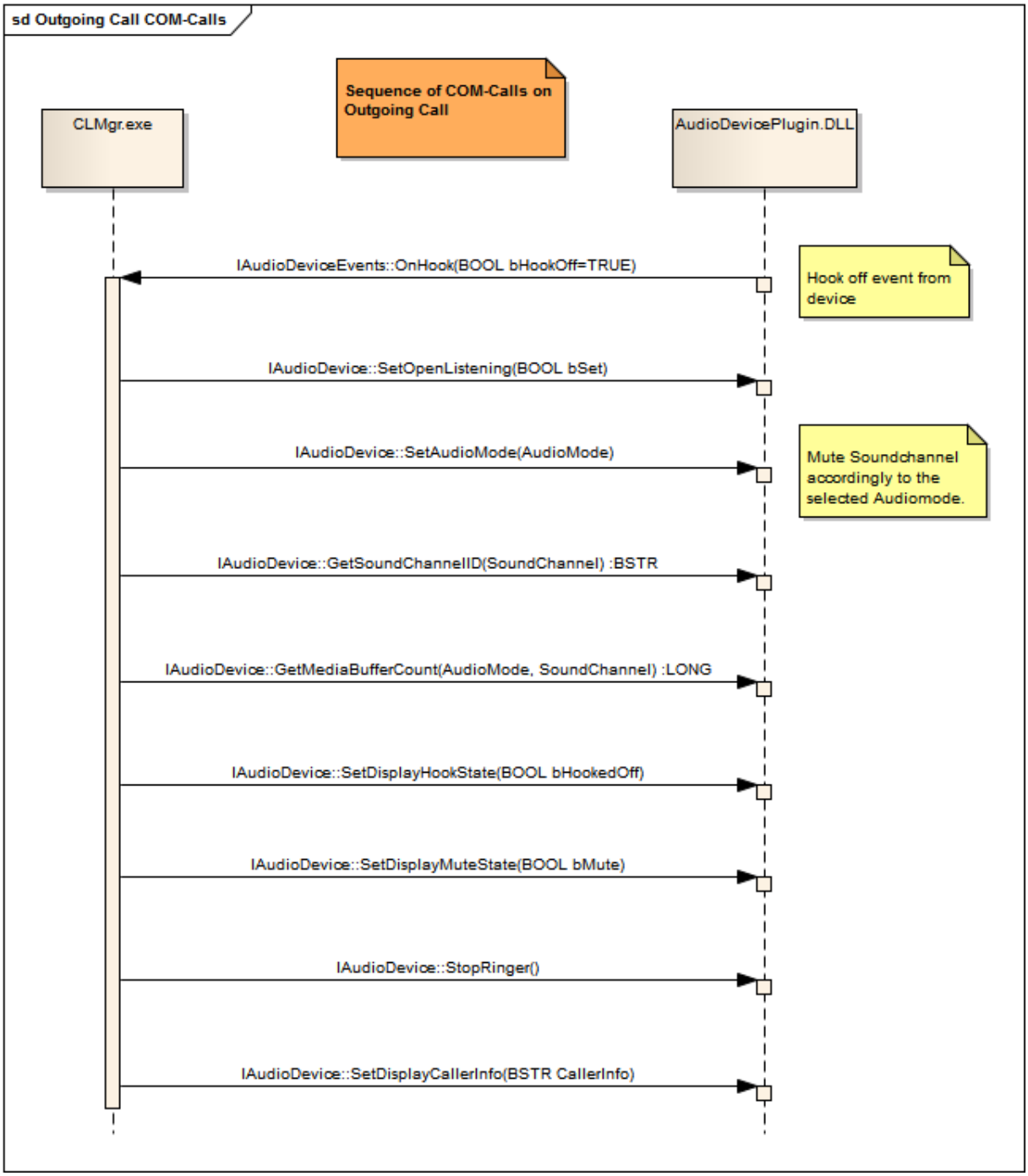
6 Appendix

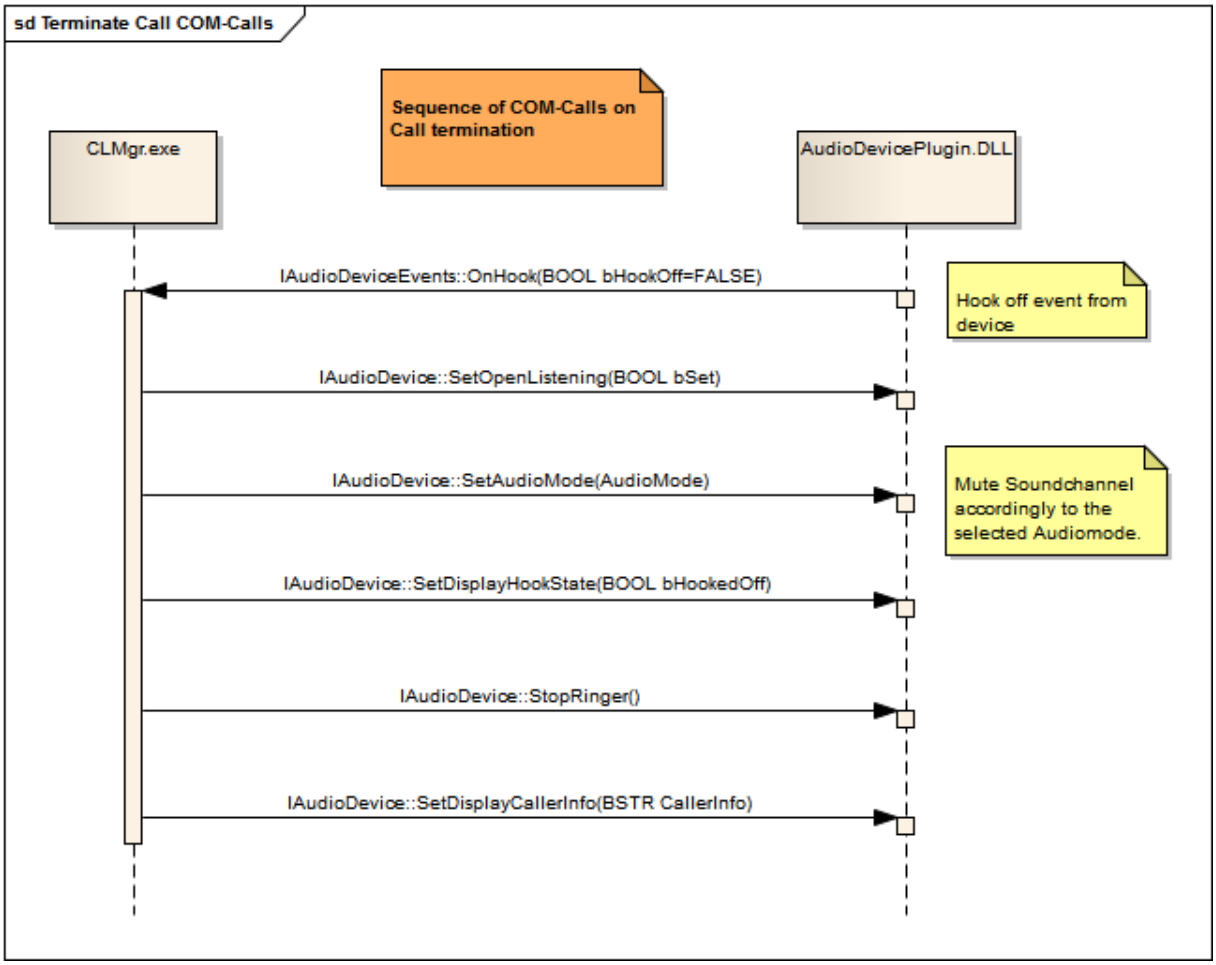
6.1 Sequence diagrams

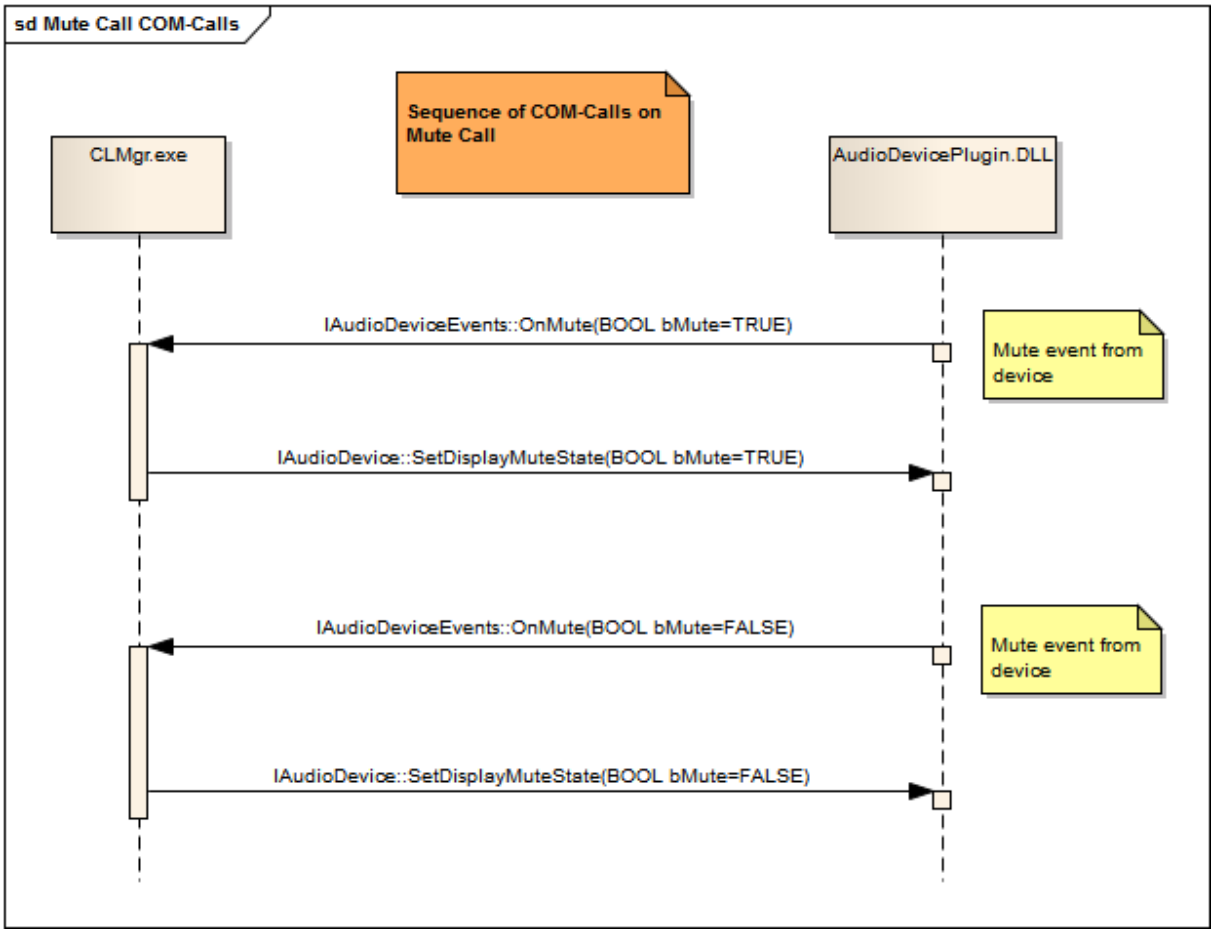


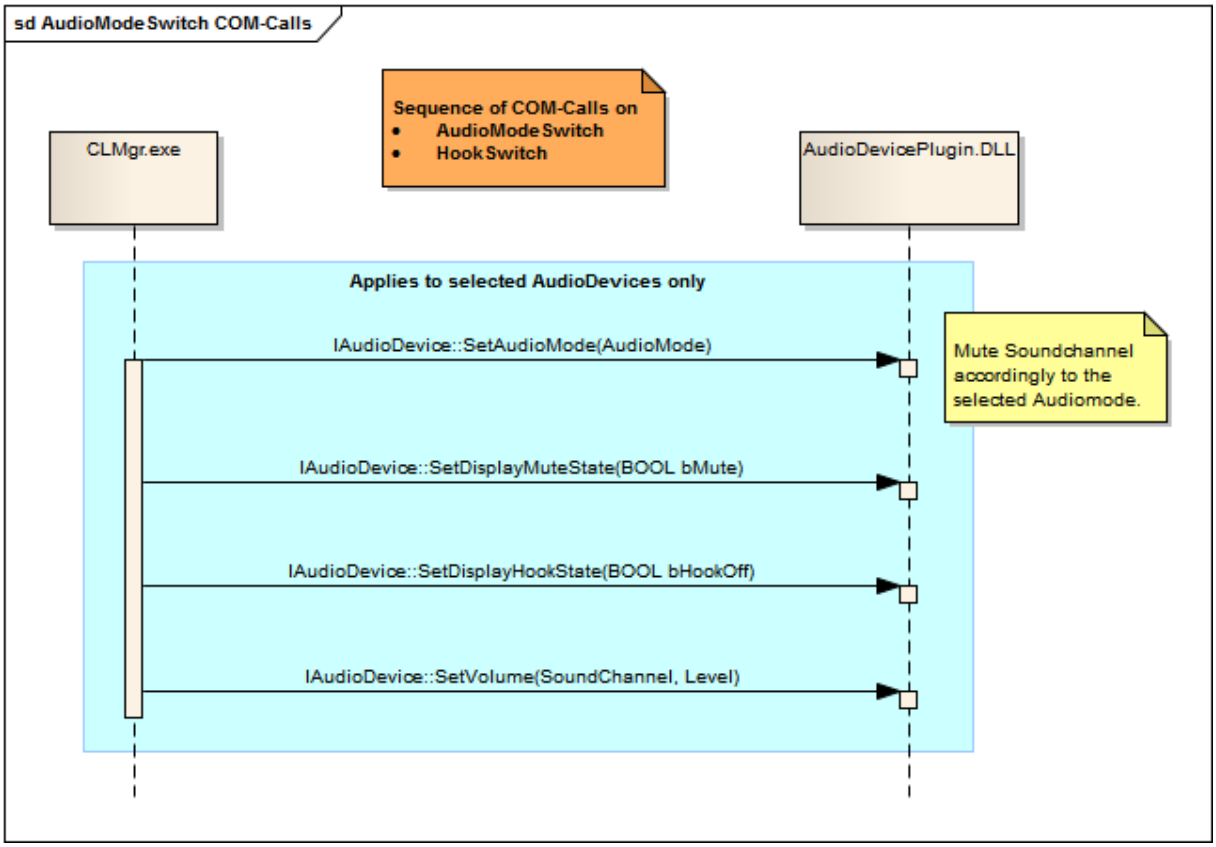

sd Initial COM-Calls
Sequence of COM-Calls on
CLMgr-Startup
Windows Device Configuration changed
SwyxIt! Device Selection changed
CLMgr.exe
AudioDevicePlugin.DLL
if not already loaded
AudioDevicePlugin CoCreateInstance()
IAudioDeviceCollection::Initialize(BSTR RegKey)
IAudioDeviceCollection::Count() :long
Fill the collection with the plugged in devices and get all the Soundchannels of your devices and be ready to fire device and volume events.
if windows device changed
IAudioDeviceCollection:DeviceChangedEvent()
Refill the collection with the plugged in devices and get all the Soundchannels of your devices and be ready to fire device and volume events.
Ready to receive Events from AudioDevicePlugin()
ForEach IAudioDevice
IAudioDeviceCollection::Item() :IAudioDevice
IAudioDevice::EnumSoundChannelIDs() :SAFEARRAY
ForEach IAudioDevice
IAudioDevice::GetIsSoundChannelSupported(AudioMode, SoundChannel) :BOOL
Open desired IAudioDevice
IAudioDevice::Open()


sd Incoming Call COM-Calls
Sequence of COM-Calls on Incoming Call
CLMgr.exe
AudioDevicePlugin.DLL
IAudioDevice::SetVolume(AudioMode Ringing)
IAudioDevice::StartRinger()
IAudioDevice::SetDisplayCallerInfo(BSTR CallerInfo)
IAudioDeviceEvents::OnHook(BOOL bHookOff=TRUE)
Hook off event from device
IAudioDevice::SetOpenListening(BOOL bSet)
IAudioDevice::SetAudioMode(AudioMode)
Mute Soundchannel accordingly to the selected Audiomode.
IAudioDevice::GetSoundChannelID(SoundChannel)
IAudioDevice::GetMediaBufferCount(AudioMode, SoundChannel)
IAudioDevice::SetDisplayHookState(BOOL bHookOff)
IAudioDevice::SetDisplayMuteState(BOOL bMute)
IAudioDevice::StopRinger()
IAudioDevice::SetDisplayCallerInfo(BSTR CallerInfo)


sd Outgoing Call COM-Calls
Sequence of COM-Calls on Outgoing Call
CLMgr.exe
AudioDevicePlugin.DLL
IAudioDeviceEvents::OnHook(BOOL bHookOff=TRUE)
Hook off event from device
IAudioDevice::SetOpenListening(BOOL bSet)
IAudioDevice::SetAudioMode(AudioMode)
Mute Soundchannel accordingly to the selected Audiomode.
IAudioDevice::GetSoundChannelID(SoundChannel) :BSTR
IAudioDevice::GetMediaBufferCount(AudioMode, SoundChannel) :LONG
IAudioDevice::SetDisplayHookState(BOOL bHookedOff)
IAudioDevice::SetDisplayMuteState(BOOL bMute)
IAudioDevice::StopRinger()
IAudioDevice::SetDisplayCallerInfo(BSTR CallerInfo)


sd Terminate Call COM-Calls
Sequence of COM-Calls on Call termination
CLMgr.exe
AudioDevicePlugin.DLL
IAudioDeviceEvents::OnHook(BOOL bHookOff=FALSE)
Hook off event from device
IAudioDevice::SetOpenListening(BOOL bSet)
IAudioDevice::SetAudioMode(AudioMode)
Mute Soundchannel accordingly to the selected Audiomode.
IAudioDevice::SetDisplayHookState(BOOL bHookedOff)
IAudioDevice::StopRinger()
IAudioDevice::SetDisplayCallerInfo(BSTR CallerInfo)


sd Mute Call COM-Calls
Sequence of COM-Calls on Mute Call
CLMgr.exe
AudioDevicePlugin.DLL
IAudioDeviceEvents::OnMute(BOOL bMute=TRUE)
Mute event from device
IAudioDevice::SetDisplayMuteState(BOOL bMute=TRUE)
IAudioDeviceEvents::OnMute(BOOL bMute=FALSE)
Mute event from device
IAudioDevice::SetDisplayMuteState(BOOL bMute=FALSE)


sd AudioModeSwitch COM-Calls
Sequence of COM-Calls on
AudioModeSwitch
HookSwitch
CLMgr.exe
AudioDevicePlugin.DLL
Applies to selected AudioDevices only
IAudioDevice::SetAudioMode(AudioMode)
Mute Soundchannel accordingly to the selected Audiomode.
IAudioDevice::SetDisplayMuteState(BOOL bMute)
IAudioDevice::SetDisplayHookState(BOOL bHookOff)
IAudioDevice::SetVolume(SoundChannel, Level)