**设备拓扑Device Topologies**

# 1.什么时候用设备拓扑API

**设备拓扑API是最基础的轮子, 而端点设备模型(Endpoint Device Model)则封装的很简便**

设备拓扑API([DeviceTopology API](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/devicetopology-api))能够控制音频适配器内部的各种功能，这些功能没法用[MMDevice API](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/mmdevice-api), [WASAPI](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/wasapi)或 [EndpointVolume API](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/endpointvolume-api)访问。

[MMDevice API](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/mmdevice-api), [WASAPI](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/wasapi), 和[EndpointVolume API](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/endpointvolume-api) 将麦克风、扬声器、耳机和其他音频输入输出设备封装为**音频端点设备**([audio endpoint devices](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/audio-endpoint-devices))。

**端点设备模型**大大简化了对音频设备中的音量和静音控件的访问。有了它, 对于这些简单控件, 无需遍历音频设备硬件的内部拓扑, 就可以实现访问。在Windows Vista中，音频引擎自动配置了音频设备的拓扑结构，供音频应用程序使用。因此，应用程序很少（如果有的话）需要为此而使用设备拓扑API。

*举个例子，假如:*

1. *音频适配器包含一个输入多路复用器(Multiplexer, Switcher)，它可以从线路输入或麦克风捕获流，但不能同时从两个端点设备捕获流。*
2. *在扬声器高级属性中的使能了独占模式(*[*Exclusive-Mode Streams*](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/exclusive-mode-streams)*)。*
3. *一个共享模式程序正在记录来自线路输入(Line Input)的流,*

*这时一个独占模式程序开始从麦克风记录流，则音频引擎自动将多路复用器从线路输入切换到麦克风。*

*在早期的Windows版本（包括Windows XP）中，就没有这么幸福了. 那时独占模式程序需要用Windows多媒体API中的mixerxxx函数遍历适配器设备的拓扑，发现多路复用器，并配置多路复用器以选择Microphone输入。在Windows Vista中，不再需要这些步骤。*

但是，有些音频控件(硬件) (audio hardware controls)无法通过mmdevice api、wasapi或endpointvolume api访问。则可以使用设备拓扑api遍历声适配器设备的拓扑，以发现和管理设备中的控件。使用设备拓扑API的应用程序必须小心设计，要避免干扰Windows音频策略, 也要避免动刀声卡里和其他程序共享的配置。有关Windows音频策略的详细信息，请参阅用户模式音频组件([User-Mode Audio Components](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/user-mode-audio-components))。

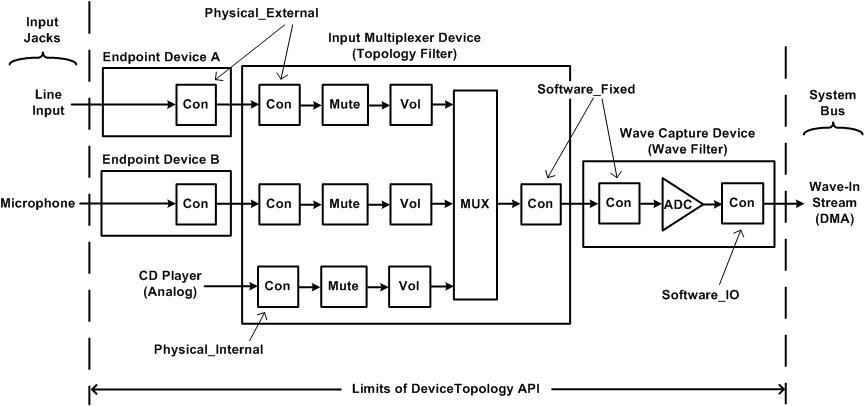
# 2.功能

设备拓扑API能在设备拓扑中发现和管理以下类型的音频控件：

* Automatic gain control 自动增益控制
* Loudness control响度控制
* Volume control音量控制
* Bass control 低音控制
* Midrange control中音控制
* Treble control高音控制
* Mute control静音控制
* Peak meter峰值计
* Input selector (multiplexer) 输入选择器（多路复用器）
* Output selector (demultiplexer) 输出选择器（多路分配器）

此外，设备拓扑api使客户机能够查询适配器设备，以获取它们支持的流格式的信息。头文件deviceopology.h定义了设备拓扑api中的接口。

下图显示了几个互相连接设备的拓扑的示例, 这是一个PCI适配器的一部分, 用于捕获来自麦克风，线路输入和CD播放器的音频。



上图展示了从模拟输入到系统中线的数据通路. 以下每个设备都作为一个设备拓扑对象(支持**[IDeviceTopology](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-idevicetopology)**接口)呈现:

* 波捕获(Wave capture device)
* 输入选择器(Input multiplexer device)
* 端点设备A(Endpoint device A)
* 端点设备B(Endpoint device B)

可以注意到，拓扑图包含了适配器设备（波捕获设备和输入输入选择器）和端点设备。 通过设备之间的连接，音频数据从一个设备传递到下一个设备。连接的每一侧都有一个连接器（图中标记为Con），数据通过该连接器进入或离开设备。

在图形的左边,信号从线路输入(line-input)和麦克风插孔(Jack)进入端点设备.

波捕获设备和输入选择器内部的流处理功能，在DeviceTopology API的术语中被称为子单元(subunits)。 以下类型的子单元显示在上图中：

* 音量控制 (Volume control, 标记为Vol)
* 静音控制(Mute control, 标记为Mute)
* 多路复用设备或输入选择器(Multiplexer or input selector, 标记为MUX)
* 模数转换器(Analog-to-digital converter, 标记为ADC)

以上子单元中前三个提供了控制接口, 可以通过DeviceTopology API访问. 模数转换器没有相关接口.

在DeviceTopology API中, 都归类为部分(Part). 无论是连接器还是子单元，所有部件都提供一组通用功能。 DeviceTopology API实现IPart接口，以表示连接器和子单元通用部分。 API实现了IConnector和ISubunit接口，以表示连接器和子单元的特定方面。

在和操作系统打交道的时候, 音频驱动将这些设备展示为内核流过滤器(KS过滤器, kernel-streaming (KS) filters), DeviceTopology API通过这些过滤器获取设备的拓扑结构。 （音频适配器驱动程序实现IMiniportWaveXxx和IMiniportTopology接口以表示这些过滤器的硬件相关部分;有关这些接口和KS过滤器的更多信息，请参阅Windows DDK文档。）

上图中DeviceTopology API构造了琐碎的拓扑，来表示的端点设备A和B. 端点设备的设备拓扑里只有单个连接器, 这里面没有包含用于处理音频数据的子单元, 只是一个占位符, 他的主要作用是作为探索设备拓扑的起点。

设备拓扑中两个部分之间的内部连接称为链接(links)。 DeviceTopology API提供了通过链接从一个部分方位下一个部分的方法。 API还提供了遍历设备拓扑之间连接的方法。

为了开始探索一组连接的设备拓扑，客户端应用程序需要先激活音频端点设备的IDeviceTopology接口。端点设备中的连接器连接到音频适配器中的连接器或网络。如果端点连接到音频适配器上的设备，则DeviceTopology API中的方法允许应用程序通过获取对另一端适配器设备的IDeviceTopology接口的引用来跨越从端点到适配器的连接。网络的情况不同, 网络没有设备拓扑。网络连接将音频流传输到远程访问系统的客户端。

The DeviceTopology API provides access only to the topologies of the hardware devices in an audio adapter. The external devices on the left edge of the diagram and the software components on the right edge are beyond the scope of the API. The dashed lines on either side of the diagram represent the limits of the DeviceTopology API. The client can use the API to explore a data path that stretches from the input jack to the system bus, but the API cannot penetrate beyond these boundaries.

Each connector in the preceding diagram has an associated connection type that indicates the type of connection that the connector makes. Thus, the connectors on the two sides of a connection always have identical connection types. The connection type is indicated by a **[ConnectorType](https://docs.microsoft.com/en-us/windows/win32/api/devicetopology/ne-devicetopology-connectortype)** enumeration value—Physical\_External, Physical\_Internal, Software\_Fixed, Software\_IO, or Network. The connections between the input multiplexer device and endpoint devices A and B are of type Physical\_External, which means that the connection represents a physical connection to an external device (in other words, a user-accessible audio jack). The connection to the analog signal from the internal CD player is of type Physical\_Internal, which indicates a physical connection to an auxiliary device that is installed inside the system chassis. The connection between the wave capture device and input multiplexer device is of type Software\_Fixed, which indicates a permanent connection that is fixed and cannot be configured under software control. Finally, the connection to the system bus on the right side of the diagram is of type Software\_IO, which indicates that the data I/O for the connection is implemented by a DMA engine under software control. (The diagram does not include an example of a Network connection type.)

The client begins traversing a data path at the endpoint device. First, the client obtains an **[IMMDevice](https://docs.microsoft.com/en-us/windows/desktop/api/Mmdeviceapi/nn-mmdeviceapi-immdevice)** interface that represents the endpoint device, as explained in [Enumerating Audio Devices](https://docs.microsoft.com/zh-cn/windows/win32/coreaudio/enumerating-audio-devices). To obtain the **IDeviceTopology** interface for the endpoint device, the client calls the **[IMMDevice::Activate](https://docs.microsoft.com/en-us/windows/desktop/api/Mmdeviceapi/nf-mmdeviceapi-immdevice-activate)** method with parameter *iid* set to REFIID IID\_IDeviceTopology.

In the example in the preceding diagram, the input multiplexer device contains all the hardware controls (volume, mute, and multiplexer) for the capture streams from the line-input and microphone jacks. The following code example shows how to obtain the **IDeviceTopology** interface for the input multiplexer device from the **IMMDevice** interface for the endpoint device for the line input or microphone:

C++复制

//-----------------------------------------------------------

// The input argument to this function is a pointer to the

// IMMDevice interface of an endpoint device. The function

// outputs a pointer (counted reference) to the

// IDeviceTopology interface of the adapter device that

// connects to the endpoint device.

//-----------------------------------------------------------

#define EXIT\_ON\_ERROR(hres) \

if (FAILED(hres)) { goto Exit; }

#define SAFE\_RELEASE(punk) \

if ((punk) != NULL) \

{ (punk)->Release(); (punk) = NULL; }

const IID IID\_IDeviceTopology = \_\_uuidof(IDeviceTopology);

const IID IID\_IPart = \_\_uuidof(IPart);

HRESULT GetHardwareDeviceTopology(

IMMDevice \*pEndptDev,

IDeviceTopology \*\*ppDevTopo)

{

HRESULT hr = S\_OK;

IDeviceTopology \*pDevTopoEndpt = NULL;

IConnector \*pConnEndpt = NULL;

IConnector \*pConnHWDev = NULL;

IPart \*pPartConn = NULL;

// Get the endpoint device's IDeviceTopology interface.

hr = pEndptDev->Activate(

IID\_IDeviceTopology, CLSCTX\_ALL,

NULL, (void\*\*)&pDevTopoEndpt);

EXIT\_ON\_ERROR(hr)

// The device topology for an endpoint device always

// contains just one connector (connector number 0).

hr = pDevTopoEndpt->GetConnector(0, &pConnEndpt);

EXIT\_ON\_ERROR(hr)

// Use the connector in the endpoint device to get the

// connector in the adapter device.

hr = pConnEndpt->GetConnectedTo(&pConnHWDev);

EXIT\_ON\_ERROR(hr)

// Query the connector in the adapter device for

// its IPart interface.

hr = pConnHWDev->QueryInterface(

IID\_IPart, (void\*\*)&pPartConn);

EXIT\_ON\_ERROR(hr)

// Use the connector's IPart interface to get the

// IDeviceTopology interface for the adapter device.

hr = pPartConn->GetTopologyObject(ppDevTopo);

Exit:

SAFE\_RELEASE(pDevTopoEndpt)

SAFE\_RELEASE(pConnEndpt)

SAFE\_RELEASE(pConnHWDev)

SAFE\_RELEASE(pPartConn)

return hr;

}

The GetHardwareDeviceTopology function in the previous code example performs the following steps to obtain the **IDeviceTopology** interface for the input multiplexer device:

1. Call the **IMMDevice::Activate** method to get the **IDeviceTopology** interface for the endpoint device.
2. With the **IDeviceTopology** interface obtained in the preceding step, call the **[IDeviceTopology::GetConnector](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-idevicetopology-getconnector)** method to get the **IConnector** interface of the single connector (connector number 0) in the endpoint device.
3. With the **IConnector** interface obtained in the preceding step, call the **[IConnector::GetConnectedTo](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-iconnector-getconnectedto)** method to get the **IConnector** interface of the connector in the input multiplexer device.
4. Query the **IConnector** interface obtained in the preceding step for its **IPart** interface.
5. With the **IPart** interface obtained in the preceding step, call the **[IPart::GetTopologyObject](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-gettopologyobject)** method to get the **IDeviceTopology** interface for the input multiplexer device.

Before the user can record from the microphone in the preceding diagram, the client application must make certain that the multiplexer selects the microphone input. The following code example shows how a client can traverse the data path from the microphone until it finds the multiplexer, which it then programs to select the microphone input:

C++复制

//-----------------------------------------------------------

// The input argument to this function is a pointer to the

// IMMDevice interface for a capture endpoint device. The

// function traverses the data path that extends from the

// endpoint device to the system bus (for example, PCI)

// or external bus (USB). If the function discovers a MUX

// (input selector) in the path, it selects the MUX input

// that connects to the stream from the endpoint device.

//-----------------------------------------------------------

#define EXIT\_ON\_ERROR(hres) \

if (FAILED(hres)) { goto Exit; }

#define SAFE\_RELEASE(punk) \

if ((punk) != NULL) \

{ (punk)->Release(); (punk) = NULL; }

const IID IID\_IDeviceTopology = \_\_uuidof(IDeviceTopology);

const IID IID\_IPart = \_\_uuidof(IPart);

const IID IID\_IConnector = \_\_uuidof(IConnector);

const IID IID\_IAudioInputSelector = \_\_uuidof(IAudioInputSelector);

HRESULT SelectCaptureDevice(IMMDevice \*pEndptDev)

{

HRESULT hr = S\_OK;

DataFlow flow;

IDeviceTopology \*pDeviceTopology = NULL;

IConnector \*pConnFrom = NULL;

IConnector \*pConnTo = NULL;

IPart \*pPartPrev = NULL;

IPart \*pPartNext = NULL;

IAudioInputSelector \*pSelector = NULL;

if (pEndptDev == NULL)

{

EXIT\_ON\_ERROR(hr = E\_POINTER)

}

// Get the endpoint device's IDeviceTopology interface.

hr = pEndptDev->Activate(

IID\_IDeviceTopology, CLSCTX\_ALL, NULL,

(void\*\*)&pDeviceTopology);

EXIT\_ON\_ERROR(hr)

// The device topology for an endpoint device always

// contains just one connector (connector number 0).

hr = pDeviceTopology->GetConnector(0, &pConnFrom);

SAFE\_RELEASE(pDeviceTopology)

EXIT\_ON\_ERROR(hr)

// Make sure that this is a capture device.

hr = pConnFrom->GetDataFlow(&flow);

EXIT\_ON\_ERROR(hr)

if (flow != Out)

{

// Error -- this is a rendering device.

EXIT\_ON\_ERROR(hr = AUDCLNT\_E\_WRONG\_ENDPOINT\_TYPE)

}

// Outer loop: Each iteration traverses the data path

// through a device topology starting at the input

// connector and ending at the output connector.

while (TRUE)

{

BOOL bConnected;

hr = pConnFrom->IsConnected(&bConnected);

EXIT\_ON\_ERROR(hr)

// Does this connector connect to another device?

if (bConnected == FALSE)

{

// This is the end of the data path that

// stretches from the endpoint device to the

// system bus or external bus. Verify that

// the connection type is Software\_IO.

ConnectorType connType;

hr = pConnFrom->GetType(&connType);

EXIT\_ON\_ERROR(hr)

if (connType == Software\_IO)

{

break; // finished

}

EXIT\_ON\_ERROR(hr = E\_FAIL)

}

// Get the connector in the next device topology,

// which lies on the other side of the connection.

hr = pConnFrom->GetConnectedTo(&pConnTo);

EXIT\_ON\_ERROR(hr)

SAFE\_RELEASE(pConnFrom)

// Get the connector's IPart interface.

hr = pConnTo->QueryInterface(

IID\_IPart, (void\*\*)&pPartPrev);

EXIT\_ON\_ERROR(hr)

SAFE\_RELEASE(pConnTo)

// Inner loop: Each iteration traverses one link in a

// device topology and looks for input multiplexers.

while (TRUE)

{

PartType parttype;

UINT localId;

IPartsList \*pParts;

// Follow downstream link to next part.

hr = pPartPrev->EnumPartsOutgoing(&pParts);

EXIT\_ON\_ERROR(hr)

hr = pParts->GetPart(0, &pPartNext);

pParts->Release();

EXIT\_ON\_ERROR(hr)

hr = pPartNext->GetPartType(&parttype);

EXIT\_ON\_ERROR(hr)

if (parttype == Connector)

{

// We've reached the output connector that

// lies at the end of this device topology.

hr = pPartNext->QueryInterface(

IID\_IConnector,

(void\*\*)&pConnFrom);

EXIT\_ON\_ERROR(hr)

SAFE\_RELEASE(pPartPrev)

SAFE\_RELEASE(pPartNext)

break;

}

// Failure of the following call means only that

// the part is not a MUX (input selector).

hr = pPartNext->Activate(

CLSCTX\_ALL,

IID\_IAudioInputSelector,

(void\*\*)&pSelector);

if (hr == S\_OK)

{

// We found a MUX (input selector), so select

// the input from our endpoint device.

hr = pPartPrev->GetLocalId(&localId);

EXIT\_ON\_ERROR(hr)

hr = pSelector->SetSelection(localId, NULL);

EXIT\_ON\_ERROR(hr)

SAFE\_RELEASE(pSelector)

}

SAFE\_RELEASE(pPartPrev)

pPartPrev = pPartNext;

pPartNext = NULL;

}

}

Exit:

SAFE\_RELEASE(pConnFrom)

SAFE\_RELEASE(pConnTo)

SAFE\_RELEASE(pPartPrev)

SAFE\_RELEASE(pPartNext)

SAFE\_RELEASE(pSelector)

return hr;

}

The DeviceTopology API implements an **[IAudioInputSelector](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iaudioinputselector)** interface to encapsulate a multiplexer, such as the one in the preceding diagram. (An **[IAudioOutputSelector](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iaudiooutputselector)** interface encapsulates a demultiplexer.) In the preceding code example, the inner loop of the SelectCaptureDevice function queries each subunit that it finds to discover whether the subunit is a multiplexer. If the subunit is a multiplexer, then the function calls the **[IAudioInputSelector::SetSelection](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-iaudioinputselector-setselection)** method to select the input that connects to the stream from the endpoint device.

In the preceding code example, each iteration of the outer loop traverses one device topology. When traversing the device topologies in the preceding diagram, the first iteration traverses the input multiplexer device and the second iteration traverses the wave capture device. The function will terminate when it reaches the connector at the right edge of the diagram. Termination occurs when the function detects a connector with a Software\_IO connection type. This connection type identifies the point at which the adapter device connects to the system bus.

The call to the **[IPart::GetPartType](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-getparttype)** method in the preceding code example obtains an **IPartType** enumeration value that indicates whether the current part is a connector or an audio-processing subunit.

The inner loop in the preceding code example steps across the link from one part to the next by calling the **[IPart::EnumPartsOutgoing](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-enumpartsoutgoing)** method. (There's also an **[IPart::EnumPartsIncoming](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-enumpartsincoming)** method for stepping in the opposite direction.) This method retrieves an **[IPartsList](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-ipartslist)** object that contains a list of all the outgoing parts. However, any part that the SelectCaptureDevice function expects to encounter in a capture device will always have exactly one outgoing part. Thus, the subsequent call to **[IPartsList::GetPart](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipartslist-getpart)** always requests the first part in the list, part number 0, because the function assumes that this is the only part in the list.

If the SelectCaptureDevice function encounters a topology for which that assumption is not valid, the function might fail to configure the device correctly. To avoid such a failure, a more general-purpose version of the function might do the following:

* Call the **[IPartsList::GetCount](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipartslist-getcount)** method to determine the number of outgoing parts.
* For each outgoing part, call **IPartsList::GetPart** to begin to traverse the data path that leads from the part.

Some, but not necessarily all, parts have associated hardware controls that clients can set or get. A particular part might have zero, one, or more hardware controls. A hardware control is represented by the following pair of interfaces:

* A generic control interface, **[IControlInterface](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-icontrolinterface)**, which has methods that are common to all hardware controls.
* A function-specific interface (for example, **[IAudioVolumeLevel](https://msdn.microsoft.com/en-us/library/Dd371019(v=VS.85).aspx)**) that exposes the control parameters for a particular type of hardware control (for example, a volume control).

To enumerate the hardware controls for a part, the client first calls the **[IPart::GetControlInterfaceCount](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-getcontrolinterfacecount)** method to determine the number of hardware controls that are associated with the part. Next, the client makes a series of calls to the **[IPart::GetControlInterface](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-getcontrolinterface)** method to obtain the **IControlInterface** interface for each hardware control. Finally, the client obtains the function-specific interface for each hardware control by calling the **[IControlInterface::GetIID](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-icontrolinterface-getiid)** method to obtain the interface ID. The client calls the **[IPart::Activate](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-activate)** method with this ID to obtain the function-specific interface.

A part that is a connector might support one of the following function-specific control interfaces:

* [**IKsFormatSupport**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iksformatsupport)
* [**IKsJackDescription**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iksjackdescription)

A part that is a subunit might support one or more of the following function-specific control interfaces:

* [**IAudioAutoGainControl**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iaudioautogaincontrol)
* [**IAudioBass**](https://msdn.microsoft.com/en-us/library/Dd370857(v=VS.85).aspx)
* [**IAudioChannelConfig**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iaudiochannelconfig)
* [**IAudioInputSelector**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iaudioinputselector)
* [**IAudioLoudness**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iaudioloudness)
* [**IAudioMidrange**](https://msdn.microsoft.com/en-us/library/Dd368232(v=VS.85).aspx)
* [**IAudioMute**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iaudiomute)
* [**IAudioOutputSelector**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iaudiooutputselector)
* [**IAudioPeakMeter**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-iaudiopeakmeter)
* [**IAudioTreble**](https://msdn.microsoft.com/en-us/library/Dd371001(v=VS.85).aspx)
* [**IAudioVolumeLevel**](https://msdn.microsoft.com/en-us/library/Dd371019(v=VS.85).aspx)
* [**IDeviceSpecificProperty**](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nn-devicetopology-idevicespecificproperty)

A part supports the **IDeviceSpecificProperty** interface only if the underlying hardware control has a device-specific control value and the control cannot be adequately represented by any other function-specific interface in the preceding list. Typically, a device-specific property is useful only to a client that can infer the meaning of the property value from information such as the part type, part subtype, and part name. The client can obtain this information by calling the **[IPart::GetPartType](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-getparttype)**, **[IPart::GetSubType](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-getsubtype)**, and **[IPart::GetName](https://docs.microsoft.com/en-us/windows/desktop/api/Devicetopology/nf-devicetopology-ipart-getname)** methods.