PortCls系统驱动程序支持[KS属性，事件和方法](https://docs.microsoft.com/windows-hardware/drivers/stream/ks-properties--events--and-methods)中描述的内部操作的子集。

Portcls.sys中的端口驱动程序通过为某些属性和事件请求提供处理程序，以及将其他请求转发到微型端口驱动程序的处理程序来支持属性和事件。

WaveCyclic，WavePci，MIDI和DMus端口驱动程序的当前实现提供以下内容：

* 支持过滤器及其引脚和节点的属性
* 支持引脚和节点上的事件，但不支持过滤器上的事件

客户端可以将过滤器或Pin实例的句柄指定为属性或事件请求的目标。对节点属性或事件的请求除了指定过滤器或引脚句柄外，还指定节点ID。有关更多信息，请参见过[滤器，引脚和节点属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/filter--pin--and-node-properties)。

拓扑端口驱动程序提供以下内容：

* 支持过滤器及其节点上的属性
* 支持节点上的事件

拓扑过滤器上的引脚表示永久存在的硬连线连接，因此无法实例化或删除。

没有端口驱动程序支持过滤器或其引脚和节点上的方法。端口驱动程序从不处理方法请求，也从不将这些请求转发给微型端口驱动程序进行处理。

音频适配器驱动程序支持以下某些或所有标准属性集：

[KSPROPSETID\_AC3](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-ac3)

[KSPROPSETID\_Acoustic\_Echo\_Cancel](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-acoustic-echo-cancel)

[KSPROPSETID\_Audio](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-audio)

[KSPROPSETID\_DirectSound3DBuffer](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-directsound3dbuffer)

[KSPROPSETID\_DirectSound3DListener](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-directsound3dlistener)

[KSPROPSETID\_DrmAudioStream](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-drmaudiostream)

[KSPROPSETID\_General](https://docs.microsoft.com/windows-hardware/drivers/stream/kspropsetid-general)

[KSPROPSETID\_Hrtf3d](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-hrtf3d)

[KSPROPSETID\_Jack](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-jack)

[KSPROPSETID\_Pin](https://docs.microsoft.com/windows-hardware/drivers/stream/kspropsetid-pin)

[KSPROPSETID\_Synth](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-synth)

[KSPROPSETID\_Synth\_Dls](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-synth-dls)

[KSPROPSETID\_TopologyNode](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-topologynode)

所有音频驱动程序都支持**KSPROPSETID\_Audio**属性集。

某些音频适配器驱动程序支持以下事件集：

[KSEVENTSETID\_AudioControlChange](https://docs.microsoft.com/windows-hardware/drivers/audio/kseventsetid-audiocontrolchange)

此外，音频适配器驱动程序可以自由提供头文件Ksmedia.h中定义的其他属性集的属性处理程序。驱动程序也可以定义和支持自己的自定义属性和事件集，但是只有知道自定义属性或事件的应用程序才能使用它。

本节讨论音频特定的属性和事件。它包含以下主题：

[音频属性请求](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-property-requests)

[过滤器，引脚和节点属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/filter--pin--and-node-properties)

[音频属性处理程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-property-handlers)

[音频属性的基本支持查询](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/basic-support-queries-for-audio-properties)

[音频端点生成器算法](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-endpoint-builder-algorithm)

[动态子设备注册和注销](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/dynamic-subdeviceregistration-and-unregistration)

[暴露多通道节点](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/exposing-multichannel-nodes)

[引脚类别属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/pin-category-property)

[音频端点设备的友好名称](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/friendly-names-for-audio-endpoint-devices)

[音频位置属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-position-property)

[引脚数据范围和交叉点属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/pin-data-range-and-intersection-properties)

[杰克描述属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/jack-description-property)

[麦克风阵列的几何属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/microphone-array-geometry-property)

[硬件活动](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/hardware-events)

**音频属性请求**

* 2017/04/20
* 2分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/audio-property-requests.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/audio-property-requests.md" \o "1个贡献者)

Microsoft Windows驱动程序模型（WDM）音频驱动程序的客户端可以将对KS [属性的](https://docs.microsoft.com/windows-hardware/drivers/stream/ks-properties)请求发送到该驱动程序已实例化的KS筛选器和引脚。例如，用户模式客户端可以通过使用I / O控制代码为IOCTL\_KS\_PROPERTY 的[**DeviceIoControl**](https://docs.microsoft.com/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol)函数（请参阅Microsoft Windows SDK文档）来发送KS属性请求。此函数将包含属性请求的IRP发送到指定的过滤器或pin对象。

音频驱动程序支持对属性（KSPROPERTY\_TYPE\_GET，KSPROPERTY\_TYPE\_SET和KSPROPERTY\_TYPE\_BASICSUPPORT）的获取，设置和基本支持请求。有关更多信息，请参见[音频驱动程序属性集](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-property-sets)。

客户端可以发送三种属性的请求：过滤器属性，引脚属性和节点属性。有关更多信息，请参见过[滤器，引脚和节点属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/filter--pin--and-node-properties)。

当向过滤器对象发送过滤器属性请求时，客户端通过其实例句柄指定目标过滤器（请参阅“ [过滤器工厂”](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/filter-factories)）。同样，在向引脚对象发送引脚属性请求时，目标引脚由其实例句柄指定（请参见[引脚工厂](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/pin-factories)）。每种类型的请求都包含一个**[KSPROPERTY](https://docs.microsoft.com/previous-versions/ff564262(v=vs.85))**结构，该结构指定以下内容：

* 标识属性集的GUID
* 标识指定属性集中的属性项的索引
* 指示属性请求类型的标志（获取，设置或基本支持）

相关属性被收集在一起以形成一个属性集。特定属性由其属性集和指定其在该属性集中的位置的索引标识。

节点属性请求包含[**KSNODEPROPERTY**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-ksnodeproperty)结构，该结构结合了KSPROPERTY结构和节点ID。根据节点属性，属性请求的目标是过滤器实例或引脚实例。

如果一个过滤器可以创建一个特定节点类型的多个实例，则该请求的目标由一个引脚句柄指定。句柄在节点实例所在的数据路径的开头或结尾标识引脚实例。对于包含SUM或MUX节点的过滤器（请参阅[**KSNODETYPE\_SUM**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-sum)和[**KSNODETYPE\_MUX**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-mux)），适用以下规则：

* 如果属性属于位于接收器（输入）引脚下游且位于SUM或MUX节点上游的节点，则将属性请求发送到接收器引脚。
* 如果属性属于位于SUM或MUX节点下游且位于源（输出）引脚上游的节点，则将属性请求发送到源引脚。（此外，对SUM或MUX节点的属性请求将发送到源引脚。）

利用这些约定，可以唯一地标识特定数据路径上的特定节点。

有关使用混音器API遍历数据路径中的节点的信息，请参阅[内核流拓扑到音频混音器API转换](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-streaming-topology-to-audio-mixer-api-translation)。

**过滤器，引脚和节点属性**

* 2017/04/20
* 9分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/filter--pin--and-node-properties.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/filter--pin--and-node-properties.md" \o "1个贡献者)

Microsoft Windows驱动程序模型（WDM）音频驱动程序将音频设备表示为KS筛选器，并且将设备上的硬件缓冲区表示为筛选器上的引脚。当客户端向这些过滤器或pin对象之一发送属性请求时，端口驱动程序将接收该请求，并将该请求路由到端口驱动程序或微型端口驱动程序中的相应属性处理程序。

音频设备支持三种属性：

* **筛选器属性**

过滤器属性是整个过滤器的属性，而不是过滤器中特定引脚或节点的属性。对过滤器属性的请求指定了过滤器句柄，但未指定节点ID。

* **引脚特性**

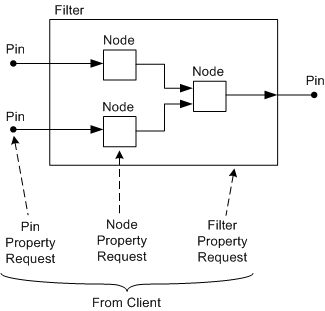
管脚属性是过滤器上特定管脚实例的属性。对这些属性的请求指定了针形手柄，但没有指定节点ID。

* **节点属性**

节点属性是过滤器中的拓扑节点的属性。对节点属性的请求指定了过滤器句柄或引脚句柄以及节点ID。

节点属性请求是指定过滤器还是引脚句柄取决于该节点对于过滤器是唯一的。有关更多信息，请参见下面的“节点属性”部分。

下图显示了这三种属性请求：发送到引脚实例的引脚属性请求，发送到节点（在过滤器或引脚实例上）的节点属性请求，以及发送到过滤器的过滤器属性请求实例。



通常，端口驱动程序处理大多数对过滤器和引脚属性的请求，而微型端口驱动程序处理对节点属性的请求。

端口驱动程序为[SysAudio系统驱动程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-mode-wdm-audio-components#sysaudio_system_driver)（请参阅[KSPROPSETID\_Sysaudio](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-sysaudio)和[KSPROPSETID\_Sysaudio\_Pin](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-sysaudio-pin)）和[WDMAud系统驱动](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/user-mode-wdm-audio-components#wdmaud_system_driver)程序使用的过滤器和引脚属性提供自己的内置处理[程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/user-mode-wdm-audio-components#wdmaud_system_driver)。微型端口驱动程序不需要为端口驱动程序处理的属性实现处理程序。典型的微型端口驱动程序几乎不提供过滤器和引脚属性的处理程序。微型端口驱动程序为节点属性提供处理程序，这些属性代表音频设备的硬件相关功能。端口驱动程序不提供节点属性的内置处理，除了[**KSPROPERTY\_TOPOLOGY\_NAME之外**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-topology-name)。

当端口驱动程序和微型端口驱动程序都提供相同属性的处理程序时，端口驱动程序将使用其自己的处理程序，并忽略微型端口驱动程序的处理程序。

**筛选器描述符**

端口驱动程序通过调用[**IMiniport :: GetDescription**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiport-getdescription)方法来获取指向微型端口驱动程序的属性处理程序的指针。通过此方法，端口驱动程序检索指向微型端口驱动程序的过滤器描述符的指针，该描述符是[**PCFILTER\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcfilter_descriptor)类型的结构。此结构为过滤器，引脚和节点属性指定微型端口驱动程序的属性处理程序：

* PCFILTER\_DESCRIPTOR结构的**AutomationTable**成员指向过滤器的自动化表。该表为过滤器属性指定微型端口驱动程序的属性处理程序。
* PCFILTER\_DESCRIPTOR结构的**Pins**成员包含**引脚**的自动化表。每个表都为特定引脚类型的引脚属性指定属性处理程序。
* PCFILTER\_DESCRIPTOR结构的**Nodes**成员包含过滤器内部拓扑节点的自动化表。每个表都为特定节点类型的节点属性指定属性处理程序。

**筛选器属性**

端口驱动程序通过PCFILTER\_DESCRIPTOR 的**AutomationTable**成员访问微型端口驱动程序的筛选器属性处理程序。通常，此自动化表包含很少的处理程序，因为端口驱动程序为SysAudio和WDMAud用于查询和配置音频设备的所有筛选器属性提供了自己的内置处理程序。

但是，微型端口驱动程序可以为过滤器属性（例如[**KSPROPERTY\_GENERAL\_COMPONENTID）**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-general-componentid)提供处理程序，这些处理程序提供端口驱动程序不可用的硬件相关信息。Microsoft Windows驱动程序工具包（WDK）中的两个示例音频驱动程序处理KSPROPERTY\_GENERAL\_COMPONENTID属性。有关更多信息，请参见Msvad和Sb16示例中的微型端口驱动程序实现。

Portcls.sys中的所有端口驱动程序都提供对[KSPROPSETID\_Pin](https://docs.microsoft.com/windows-hardware/drivers/stream/kspropsetid-pin)和[KSPROPSETID\_Topology](https://docs.microsoft.com/windows-hardware/drivers/stream/kspropsetid-topology)属性集的处理。这些集合中的所有属性都是过滤器属性，但[**KSPROPERTY\_TOPOLOGY\_NAME除外**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-topology-name)，它是一个节点属性（使用过滤器句柄而不是pin句柄来指定请求的目标）。端口驱动程序支持KSPROPSETID\_Pin属性的以下子集：

[**KSPROPERTY\_PIN\_CATEGORY**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-category)

[**KSPROPERTY\_PIN\_CINSTANCES**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-cinstances)

[**KSPROPERTY\_PIN\_COMMUNICATION**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-communication)

[**KSPROPERTY\_PIN\_CONSTRAINEDDATARANGES**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-constraineddataranges)

[**KSPROPERTY\_PIN\_CTYPES**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-ctypes)

[**KSPROPERTY\_PIN\_DATAFLOW**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-dataflow)

[**KSPROPERTY\_PIN\_DATAINTERSECTION**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-dataintersection)

[**KSPROPERTY\_PIN\_DATARANGES**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-dataranges)

[**KSPROPERTY\_PIN\_GLOBALCINSTANCES**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-globalcinstances)

[**KSPROPERTY\_PIN\_INTERFACES**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-interfaces)

[**KSPROPERTY\_PIN\_MEDIUMS**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-mediums)

[**KSPROPERTY\_PIN\_NAME**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-name)

[**KSPROPERTY\_PIN\_NECESSARYINSTANCES**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-necessaryinstances)

[**KSPROPERTY\_PIN\_PHYSICALCONNECTION**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-physicalconnection)

[**KSPROPERTY\_PIN\_PROPOSEDATAFORMAT**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-proposedataformat)

[**KSPROPERTY\_PIN\_PROPOSEDATAFORMAT2**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-proposedataformat2)

这些属性提供有关属于过滤器的引脚工厂的信息。通常，客户端在创建引脚实例之前向过滤器查询这些属性。端口驱动程序支持所有四个KSPROPSETID\_Topology属性，这些属性提供有关过滤器内部拓扑的信息。

此外，DMus端口驱动程序为[**KSPROPERTY\_SYNTH\_MASTERCLOCK**](https://docs.microsoft.com/previous-versions/ff537403(v=vs.85))属性提供了处理程序，该属性是DirectMusic过滤器的仅获取属性。KSPROPERTY\_SYNTH\_MASTERCLOCK是[KSPROPSETID\_SynthClock](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-synthclock)属性集的成员。

**引脚特性**

端口驱动程序通过PCFILTER\_DESCRIPTOR 的**Pins**成员访问微型端口驱动程序的引脚属性处理程序。该成员指向引脚描述符的数组，每个描述符指向引脚类型的自动化表（由引脚ID标识，该ID只是数组索引）。

通常，这些自动化表包含很少的条目，因为端口驱动程序为SysAudio和WDMAud使用的所有引脚属性提供了自己的处理程序。微型端口驱动程序可以为端口驱动程序不处理的一个或多个引脚属性提供处理程序，但只有了解这些属性的客户端才能为其发送属性请求。

除拓扑端口驱动程序外，Portcls.sys中的所有端口驱动程序都为以下引脚属性提供内置处理程序：

[**KSPROPERTY\_CONNECTION\_STATE**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-connection-state)

[**KSPROPERTY\_CONNECTION\_DATAFORMAT**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-connection-dataformat)

[**KSPROPERTY\_CONNECTION\_ALLOCATORFRAMING**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-connection-allocatorframing)

[**KSPROPERTY\_STREAM\_ALLOCATOR**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-stream-allocator)

[**KSPROPERTY\_STREAM\_MASTERCLOCK**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-stream-masterclock)

[**KSPROPERTY\_AUDIO\_POSITION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-position)

[**KSPROPERTY\_DRMAUDIOSTREAM\_CONTENTID**](https://docs.microsoft.com/previous-versions/ff537351(v=vs.85))

此列表中的某些属性要求微型端口驱动程序提供与硬件有关的信息。当端口驱动程序收到包含对这些属性之一的请求的IRP时，它不会将IRP传递给微型端口驱动程序。而是，端口驱动程序自己处理请求，但是其处理程序通过调用微型端口驱动程序中的入口点来获取所需的信息。例如，端口驱动程序为KSPROPERTY\_AUDIO\_POSITION请求提供其自己的属性处理程序。该处理程序仅调用微型端口驱动程序流的**GetPosition**方法（例如[**IMiniportWavePciStream :: GetPosition**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavepcistream-getposition)）来获取当前位置。

**节点属性**

端口驱动程序通过PCFILTER\_DESCRIPTOR 的**Nodes**成员访问微型端口驱动程序的节点属性处理程序。该成员指向节点描述符的数组，每个描述符指向节点类型的自动化表（由节点ID标识，该ID只是数组索引）。通常，属于微型端口驱动程序的所有或大多数属性处理程序都驻留在**Nodes**数组中。音频驱动程序将音频设备中的硬件控件表示为拓扑节点，并且使用属性机制为客户端提供对与硬件相关的控件设置的访问权限。

如前所述，客户端将过滤器属性请求发送到过滤器句柄，并将插针属性请求发送到销子句柄。与过滤器或引脚实例不同，节点不是内核对象，并且没有句柄。客户端将节点属性请求发送到pin句柄或过滤器句柄，但该请求还指定了节点ID，以指示该请求是针对节点属性而不是pin或filter属性。

以下是确定节点属性应使用过滤器句柄还是引脚句柄的一般规则：

* 如果过滤器包含特定引脚类型的多个实例，并且该类型的每个引脚都包含具有特定节点ID的节点，则每个引脚实例都包含该节点的实例。在这种情况下，节点属性请求必须指定一个pin句柄（而不仅仅是过滤器句柄）以区分相同节点类型的多个实例。引脚句柄和节点ID的组合明确地将特定节点实例标识为请求的目标。
* 如果过滤器仅包含特定节点的一个实例，则节点属性请求将指定过滤器句柄。过滤器句柄和节点ID的组合足以明确标识作为请求目标的节点。

但是，在为特定节点属性实现处理程序之前，驱动程序编写者应参考“ [音频驱动程序属性集”，](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-property-sets)以检查是否应将属性目标指定为过滤器句柄或引脚句柄。

除了KSPROPERTY\_TOPOLOGY\_NAME，Portcls.sys中的端口驱动程序当前不提供对节点属性的内置处理。

**规格过多和规格不足的财产要求**

驱动程序应准备好处理来自不遵循上述规则的客户端的属性请求。请求可以指定得过多或不足：

* **要求过多**

如果属性请求仅需要过滤器句柄，但客户端将请求发送到pin句柄，则该请求的目标将被过度指定。但是，驱动程序通常将请求视为有效。也就是说，他们将请求视为已发送到包含该引脚的过滤器。

* **未指定的要求**

如果属性请求需要一个针形手柄，但是客户端将请求发送到过滤器手柄，则该请求的目标未指定。例如，如果一个过滤器包含多个具有相同节点类型的引脚实例，并且客户端将对该节点类型的属性的请求发送到过滤器句柄而非引脚句柄，则驱动程序将无法确定应该使用哪个节点实例收到请求。在这种情况下，行为取决于驱动程序。某些驱动程序不会自动使所有未指定的请求失败，而是将未指定的设置属性请求视为有效。在这种情况下，解释是请求为指定的节点ID设置了默认值。当引脚工厂创建新节点实例时，属于新节点的属性将初始化为默认值。更改默认值的请求对在请求之前创建的节点实例无效。此外，由于处理程序无法确定要查询该属性的节点实例，因此驱动程序总是会失败地处理未指定的get-property请求。

**规则例外**

由于历史原因，一些音频属性具有违反这些一般规则的行为怪癖。以下是示例：

* 如[应用扬声器配置设置中所述](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/applying-speaker-configuration-settings)，客户端可以通过设置3-D节点（[**KSNODETYPE\_3D\_EFFECTS**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-3d-effects)）的[**KSPROPERTY\_AUDIO\_CHANNEL\_CONFIG**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-channel-config)属性来更改音频设备的扬声器配置。扬声器配置设置是全局的，因为它会更改设备通过扬声器播放的混音中所有流的扬声器配置。根据一般规则，影响整个过滤器的节点属性请求应指定过滤器句柄（加上节点ID）。但是，此特定属性需要使用针形手柄而不是过滤器手柄。引脚句柄指定包含3-D节点的引脚实例，该节点是请求的目标。
* [**KSPROPERTY\_SYNTH\_VOLUME**](https://docs.microsoft.com/previous-versions/ff537409(v=vs.85))和[**KSPROPERTY\_SYNTH\_MASTERCLOCK**](https://docs.microsoft.com/previous-versions/ff537403(v=vs.85))是合成节点（[**KSNODETYPE\_SYNTHESIZER**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-synthesizer)）的属性。尽管两者都是节点属性，但对这些属性的请求不包括节点ID。（请注意，请求的属性描述符是[**KSPROPERTY**](https://docs.microsoft.com/previous-versions/ff564262(v=vs.85))类型的结构，而不是[**KSNODEPROPERTY**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-ksnodeproperty)类型。）此行为违反了节点属性需要节点ID的一般规则。尽管存在这种差异，但支持任一属性的微型端口驱动程序应通过PCFILTER\_DESCRIPTOR的**Nodes**成员（而不是**Pins**成员）提供属性处理程序。

# 音频属性处理程序

* 2018/08/07
* 5分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/audio-property-handlers.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/audio-property-handlers.md" \o "1个贡献者)

微型端口驱动程序在[**PCPROPERTY\_ITEM**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcproperty_item)结构中存储有关它支持的每个属性的信息。此结构包含有关该属性的以下信息：

* 属性集GUID和属性ID（或索引）
* 指向属性处理程序例程的函数指针
* 指定处理程序支持的属性操作的标志

微型端口驱动程序为过滤器提供了一个自动化表（由[**PCAUTOMATION\_TABLE**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcautomation_table)结构指定）。驱动程序为过滤器的引脚类型和节点类型提供了其他自动化表-每个引脚或节点类型都有自己的表。每个自动化表都包含一个PCPROPERTY\_ITEM结构的数组（可能为空），并且每个结构都描述了过滤器，引脚或节点的一个属性。当客户端将属性请求发送到过滤器，引脚或节点时，端口驱动程序会将请求通过自动化表路由到适当的属性处理程序。

微型端口驱动程序可以为每个属性指定唯一的属性处理程序例程。但是，如果驱动程序处理多个相似的属性，为方便起见，有时可以将它们合并为一个处理程序例程。是为每个属性提供唯一的处理程序，还是将多个属性合并为一个处理程序，都是由驱动程序编写者决定的实施决定，并且对于提交属性请求的客户端应该是透明的。

用户模式客户端可以通过将dwIoControlCode调用参数设置为IOCTL\_KS\_PROPERTY 的Microsoft Win32函数[**DeviceIoControl**](https://docs.microsoft.com/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol)来发送获取，设置或基本支持属性请求。操作系统将此调用转换为**[IRP](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/wdm/ns-wdm-_irp)**，并将其分派给类驱动程序。有关更多信息，请参见[KS属性](https://docs.microsoft.com/windows-hardware/drivers/stream/ks-properties)。

当客户端向过滤器句柄或引脚句柄发送KS属性请求（即IOCTL\_KS\_PROPERTY I / O控制IRP）时，KS系统驱动程序（Ks.sys）将该请求传递给该过滤器对象的端口驱动程序或固定物件。如果微型端口驱动程序为该属性提供了处理程序，则端口驱动程序会将请求转发给该处理程序。在转发请求之前，端口驱动程序将信息从属性请求转换为[**PCPROPERTY\_REQUEST**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-_pcproperty_request)结构指定的格式。端口驱动程序将此结构传递给微型端口驱动程序的处理程序。

所述**MajorTarget**的PCPROPERTY\_REQUEST点构件到用于音频设备的主要微型端口驱动程序接口。例如，对于WavePci设备，这是指向微型端口驱动程序对象的[IMiniportWavePci](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iminiportwavepci)接口的指针。

如果将KS属性请求发送到过滤器句柄，则**PCPROPERTY\_REQUEST的MinorTarget**成员为**NULL**。在将请求发送到Pin句柄的情况下，**MinorTarget**指向该Pin的流接口。例如，对于WavePci设备，这是指向流对象的[IMiniportWavePciStream](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iminiportwavepcistream)接口的指针。

PCPROPERTY\_REQUEST 的**Instance**和**Value**成员分别指向KS属性请求的输入和输出缓冲区。（这些缓冲区由[**DeviceIoControl**](https://docs.microsoft.com/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol)函数的lpInBuffer和lpOutBuffer参数指定。）这些缓冲区分别包含属性描述符（实例数据）和属性值（操作数据），如[音频驱动程序属性](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-property-sets)集中所述。所述**价值**构件点到输出缓冲器的开始，但**实例**指针从输入缓冲器的起始位置的偏移。

输入缓冲区以[**KSPROPERTY**](https://docs.microsoft.com/previous-versions/ff564262(v=vs.85))或[**KSNODEPROPERTY**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-ksnodeproperty)结构[**开头**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-ksnodeproperty)。端口驱动程序将信息从该结构复制到PCPROPERTY\_REQUEST结构的**Node**，**PropertyItem**和**Verb**成员中。如果缓冲区中的KSPROPERTY或KSNODEPROPERTY结构后面有任何数据，则端口驱动程序将使用指向该数据的指针加载**Instance**成员。否则，它将**Instance**设置为**NULL**。

如果输入缓冲区以不包含任何节点信息的KSPROPERTY结构开头，则端口驱动程序会将PCPROPERTY\_REQUEST结构的**Node**成员设置为ULONG（-1）。在这种情况下，端口驱动程序从微型端口驱动程序的自动化表中为过滤器或引脚调用适当的处理程序，具体取决于属性请求的目标是由过滤器手柄还是引脚手柄指定的。（如果该表未为该属性指定处理程序，则端口驱动程序将处理该请求。）

如果输入缓冲区以KSNODEPROPERTY结构开头，则端口驱动程序将节点ID从该结构复制到PCPROPERTY\_REQUEST结构的**Node**成员中，并从微型端口驱动程序的自动化表中为该节点调用适当的处理程序。（同样，如果表未为该属性指定处理程序，则端口驱动程序将代替处理该请求。）

端口驱动程序检查属性请求的操作标志中的KSPROPERTY\_TYPE\_TOPOLOGY位，以确定KSPROPERTY或KSNODEPROPERTY这两种结构中的哪一种驻留在输入缓冲区的开头：

* 如果设置了此位，则请求节点属性，并且输入缓冲区以KSNODEPROPERTY结构开头。
* 否则，输入缓冲区以KSPROPERTY结构开始。

有关KSPROPERTY\_TYPE\_TOPOLOGY的更多信息，请参见[**KSPROPERTY**](https://docs.microsoft.com/previous-versions/ff564262(v=vs.85))。

PCPROPERTY\_REQUEST结构的**InstanceSize**和**ValueSize**成员指定**Instance**和**Value**成员指向的缓冲区的大小。**ValueSize**等于属性请求的输出缓冲区的大小，但是**InstanceSize**是输入缓冲区中遵循KSPROPERTY或KSNODEPROPERTY结构的数据的大小。也就是说，**InstanceSize**是输入缓冲区的大小减去KSPROPERTY或KSNODEPROPERTY结构的大小。如果没有其他数据遵循此结构，则端口驱动程序会将**InstanceSize**设置为零（而**Instance设置**为**NULL**）。

例如，如果客户端在输入缓冲区[**中将KSNODEPROPERTY\_AUDIO\_CHANNEL**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-ksnodeproperty_audio_channel)结构指定为实例数据，则端口驱动程序将向处理程序传递PCPROPERTY\_REQUEST结构，该结构的**Instance**成员指向KSNODEPROPERTY\_AUDIO\_CHANNEL结构的**Channel**成员，并且**InstanceSize**成员包含该值

**的sizeof**（KSNODEPROPERTY\_AUDIO\_CHANNEL） - **的sizeof**（KSNODEPROPERTY）

在提交获取属性值的获取属性请求之前，客户端应分配一个输出缓冲区，微型端口驱动程序的属性处理程序可以在该输出缓冲区中写入属性值。对于某些属性，输出缓冲区的大小取决于设备，并且客户端必须向属性处理程序查询所需的缓冲区大小。在这些情况下，客户端提交初始属性请求，其输出缓冲区指针为nullptr，输出缓冲区长度为零。处理程序通过返回所需的缓冲区大小以及状态代码STATUS\_BUFFER\_OVERFLOW进行响应。然后，客户端通过分配指定大小的输出缓冲区并在第二个get-property请求中发送此缓冲区来检索属性值。

如果指定的缓冲区大小太小而无法接收任何请求的信息，则该方法返回STATUS\_BUFFER\_TOO\_SMALL。

在某些情况下，PortCls端口驱动程序返回STATUS\_BUFFER\_TOO\_SMALL而不是STATUS\_BUFFER\_OVERFLOW来响应具有非零输出缓冲区地址和大小的属性请求。在这种情况下，不会返回所需的缓冲区大小。

有关更多信息，请参见[使用NTSTATUS值](https://docs.microsoft.com/windows-hardware/drivers/kernel/using-ntstatus-values)和这些博客文章：

* [如何返回后续操作所需的字节数](https://blogs.msdn.microsoft.com/doronh/2006/12/12/how-to-return-the-number-of-bytes-required-for-a-subsequent-operation/)
* [STATUS\_BUFFER\_OVERFLOW实际上应命名为STATUS\_BUFFER\_OVERFLOW\_PREVENTED](https://devblogs.microsoft.com/oldnewthing/?p=22863)

# 音频属性的基本支持查询

* 2017/04/20
* 3分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/basic-support-queries-for-audio-properties.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/basic-support-queries-for-audio-properties.md" \o "1个贡献者)

在向过滤器，引脚或节点指定用于设置属性请求的数据时，客户端经常需要了解其为属性指定的一个或多个值的有效数据范围。范围可能因设备而异，甚至可能同一设备内的节点之间也有所不同。

定义了一些属性，以允许设置属性请求指定超出范围的值，但是微型端口驱动程序会默默地将这些值限制在支持的范围内（例如，请参阅[**KSPROPERTY\_AUDIO\_VOLUMELEVEL**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-volumelevel)）。随后对相同属性的获取请求将检索驱动程序的一个或多个值的实际设置，这些设置可能是客户端在设置请求中指定的值的固定版本。

但是，客户端可能需要了解属性值的范围，而不是仅仅依靠微型端口驱动程序来自动钳位超出范围的值。例如，为音频设备提供音量控制滑块的窗口应用程序可能需要知道设备的音量范围，才能将该范围映射到滑块的全长。

特定属性的驱动程序处理程序例程应能够响应基本支持属性请求（KSPROPERTY\_TYPE\_BASICSUPPORT）提供范围信息。当向驱动程序发送基本支持属性请求时，客户端提供一个值缓冲区，属性处理程序将基本支持信息写入该值缓冲区，该信息由[**KSPROPERTY\_DESCRIPTION**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksproperty_description)结构组成，后跟特定于属性的数据。根据属性，此数据通常包含一个或多个参数范围的规范。

通常，客户端不预先知道此值缓冲区应该有多大，并且必须向属性处理程序发送一个或两个初步请求以确定值大小。这些初步请求的格式已明确定义。客户希望驱动程序在处理基本支持请求时遵循以下约定：

* 如果请求指定值大小**的sizeof**（ULONG），则属性处理程序应该写入的值**AccessFlags**的KSPROPERTY\_DESCRIPTION结构入ULONG大小值缓冲区的构件。如果该处理程序为基本支持属性请求提供了进一步的支持，则将其设置为KSPROPERTY\_TYPE\_BASICSUPPORT标志位。
* 如果请求将值大小指定为**sizeof**（KSPROPERTY\_DESCRIPTION），则处理程序应将KSPROPERTY\_DESCRIPTION结构写入数据缓冲区。处理程序将结构的**DescriptionSize**字段设置为等于该结构的大小加上该处理程序可用于加载到结构后的数据缓冲区中的所有其他特定于属性的信息的大小。这是客户端需要分配以包含属性的基本支持信息的值缓冲区的大小。
* 如果请求指定的值大小足够大以包含KSPROPERTY\_DESCRIPTION结构和特定于属性的信息，则处理程序应将KSPROPERTY\_DESCRIPTION结构写入缓冲区的开头，并且应将特定于属性的信息写入该部分KSPROPERTY\_DESCRIPTION结构末尾的数据缓冲区的大小。编写KSPROPERTY\_DESCRIPTION结构时，处理程序应将**DescriptionSize**字段设置为该结构的大小加上该结构后的特定于属性的信息的大小。

如果请求指定的值大小与这三种情况之一不匹配，则属性处理程序将拒绝该请求并返回状态代码STATUS\_BUFFER\_TOO\_SMALL。

处理程序写入值缓冲区的特定于属性的信息可能包括属性值的数据范围。所述**MembersSize** KSPROPERTY\_MEMBERSHEADER的成员指示数据范围是否包括：

* 如果不需要范围，则**MemberSize**为零。例如，如果属性值的类型为BOOL，就是这种情况。
* 如果KSPROPERTY\_MEMBERSHEADER结构后跟一个或多个属性值的范围描述符，则**MemberSize**不为零。

对于类型为BOOL的属性值，不需要范围描述符，因为范围隐式地限制为**TRUE**和**FALSE**值。但是，需要使用范围描述符来指定整数类型的属性值的范围。

例如，对卷节点上的[**KSPROPERTY\_AUDIO\_VOLUMELEVEL**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-volumelevel)属性的基本支持请求（[**KSNODETYPE\_VOLUME**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-volume)）检索该节点的最小和最大音量设置。在这种情况下，客户端需要分配一个足够大的值缓冲区以包含以下结构：

[**KSPROPERTY\_DESCRIPTION**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksproperty_description)

[**KSPROPERTY\_MEMBERSLIST**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksproperty_memberslist)

[**KSPROPERTY\_STEPPING\_LONG**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksproperty_stepping_long)

这三个结构按照前面列表中所示的顺序打包到缓冲区中的相邻位置。在处理请求时，微型端口驱动程序将最小和最大音量级别写入KSPROPERTY\_STEPPING\_LONG结构的**Bounds**成员中。

有关带有范围描述符数组的基本支持请求的示例，请参阅“ [公开多通道节点”中](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/exposing-multichannel-nodes)的图。有关基本支持属性请求的更多信息，请参见[KS属性](https://docs.microsoft.com/windows-hardware/drivers/stream/ks-properties)。有关代码示例，请参考Microsoft Windows驱动程序工具包（WDK）中的[示例音频驱动程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/sample-audio-drivers)中的属性处理程序实现。

# 音频端点生成器算法

* 2017/04/20
* 6分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/audio-endpoint-builder-algorithm.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/audio-endpoint-builder-algorithm.md" \o "1个贡献者)

在Windows Vista和更高版本的Windows中，AudioEndpointBuilder是一项系统服务，它枚举，初始化和激活系统中的音频端点。本主题概述了AudioEndpointBuilder服务使用的算法。

AudioEndpointBuilder服务使用一种算法来发现和枚举端点。该算法旨在简化对多路复用（MUXed）捕获设备的系统访问，并有助于处理涉及多个主机引脚和多个桥接引脚或两者的拓扑。

在Windows XP中，音频模型使用术语“音频设备”来指代即插即用（PnP）树中的概念性设备。在Windows Vista和更高版本的Windows中，音频设备的概念已经过重新设计，以更好地表示用户与之进行物理交互的设备。

使用Windows Vista中的两个新API [MMDevice API](https://go.microsoft.com/fwlink/p/?linkid=130863)和[WASAPI](https://go.microsoft.com/fwlink/p/?linkid=130864)，您可以访问和操作这些新的音频设备。MMDevice API将新的音频设备称为终结点。

AudioEndpointBuilder服务监视[**KSCATEGORY\_AUDIO**](https://docs.microsoft.com/windows-hardware/drivers/install/kscategory-audio)类以了解设备接口的到达和移除。音频设备驱动程序注册KSCATEGORY\_AUDIO设备接口类的新实例时，AudioEndpointBuilder服务将检测设备接口通知，并使用算法检查系统中音频设备的拓扑并采取适当的措施。

以下列表总结了AudioEndpointBuilder使用的算法如何工作：

1. 查找任何未连接的桥接引脚。
2. 为任何未连接的桥接引脚创建一个端点。例如，当AudioEndpointBuilder找到一个引脚类型GUID为KSNODETYPE\_SPEAKER的未连接的桥脚时，它将为此桥脚创建一个扬声器端点。有关KSNODETYPE\_SPEAKER和其他Pin-类别GUIDS的更多信息，请参见WinDDK \ <内部版本号> \ inc \ api中的Ksmedia.h。
3. 设置端点的默认属性。例如，AudioEndpointBuilder设置名称，图标和形状系数。
4. 确定是否存在从端点到支持脉冲编码调制（PCM），音频编解码器3（AC3）或Windows Media视频（WMV）的主机引脚的路径。主机引脚是KSPIN结构，其通信成员设置为KSPIN\_COMMUNICATION\_SINK或KSPIN\_COMMUNICATION\_BOTH。有关KSPIN结构的更多信息，请参见[**KSPIN**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-_kspin)。
5. 使用来自音频设备界面的注册表项的属性信息填充端点PropertyStore。
6. 设置端点的状态。端点的状态可以是以下三个值之一：
   * 活性。这表明存在路径，如步骤4中所述。
   * 拔掉 如果音频设备支持插孔检测，则此状态表示端点存在路径，并且已从音频适配器上的物理连接器拔出插孔。
   * 不存在。此状态指示在步骤4中找不到路径，并且此端点不支持插孔检测。
7. 如果在关联的INF文件中指定了此端点，则将此端点设置为默认端点。

枚举端点之后，音频系统的客户端可以通过使用新的Windows Vista API（如前所述）直接操作它们，或通过使用更熟悉的API（例如Wave，[DShow](https://go.microsoft.com/fwlink/p/?linkid=130871)或[DirectSound](https://go.microsoft.com/fwlink/p/?linkid=130872)）间接地操作它们[。](https://go.microsoft.com/fwlink/p/?linkid=130872)提供了新的API方法，以便音频客户端可以以端点的MMDevice ID开头并访问同一端点的Wave或DirectSound ID。

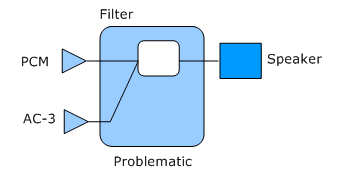
使用端点时，可以利用以下优势：

* 无论重新启动计算机的频率如何，都可以使用相同的全局唯一ID（GUID）。具有此持久性GUID比为端点保存waveOut ID或友好名称更可靠。
* 无论重新启动计算机的频率如何，都可以使用相同的PropertyStore。与音频设备相关的元数据保存在端点PropertyStore中。
* 复用（MUX）和解复用（DEMUX）引脚是自动管理的，并由AudioEndpointBuilder服务枚举。

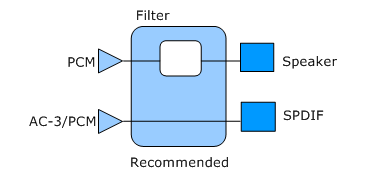
如果您开发自己的音频设备驱动程序和INF文件以与您的音频设备一起使用，或者开发一个音频应用程序，或同时开发这两个应用程序，则最好了解以下问题和最佳实践。在牢记这些建议的情况下开发驱动程序和应用程序时，您会生成与AudioEndpointBuilder一起更有效地工作的驱动程序，INF文件和音频客户端。

* 命名约定。端点使用的命名约定基于网桥引脚的友好名称。但是，对于扬声器端点，该名称已被硬编码为“扬声器”，并且不能由您的驱动程序或第三方应用程序更改。
* 次优拓扑。由于AudioEndpointBuilder用来枚举端点的算法，某些拓扑被认为不是最佳的。例如，当您创建这些次优拓扑之一时，您创建的主机引脚具有隐藏的端点，并且AudioEndpointBuilder或AudioEndpointBuilder无法链接到其关联的主机引脚的拆分器（拆分端点）无法看到它们。
  + **隐藏的端点**

在下图中，显示了KS滤波器具有两个主机引脚，这些主机引脚连接到单个桥接引脚（扬声器）。



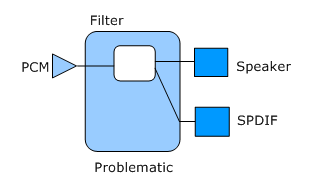
当AudioEndpointBuilder发现此桥接器引脚时，它会追溯到仅一个主机引脚的路径，设置桥接器引脚的默认值，创建并激活Speaker端点，并继续发现其他桥接器引脚。因此，另一个主机引脚对AudioEndpointBuilder保持隐藏状态。



在上图中，已经重新设计了有问题的拓扑，以便AudioEndpointBuilder可以发现两个主机引脚（PCM和AC-3 / PCM），因为它现在可以看到两个桥接引脚（扬声器和SPDIF）。

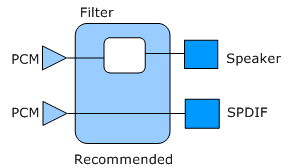
* + **分离器**

当一个主机引脚连接到多个桥接引脚时，会产生另一种次优拓扑。下图显示了一个拓扑，其中PCM主机引脚连接到扬声器桥接引脚和SPDIF桥接引脚。



在这种情况下，AudioEndpointBuilder会发现一个桥接引脚并追溯到PCM主机引脚的路径，设置默认值，然后创建并激活Speaker端点。当AudioEndpointBuilder发现下一个桥接引脚时，它将追溯到同一PCM主机引脚的路径，设置默认值，然后创建并激活SPDIF端点。但是，尽管两个端点均已初始化和激活，但向其中一个端点的流传输使得无法同时向另一个端点流传输。换句话说，它们是互斥的端点。

下图显示了此拓扑的重新设计，其中存在单独的连接。这种设计使AudioEndpointBuilder可以为两个桥接器引脚中的每一个追溯到PCM主机引脚的路径。



* 端点格式。当音频引擎以共享模式运行时，端点的格式采用安装时INF文件指示的特定设置。例如，音频设备的音频驱动程序使用其关联的INF文件将默认终结点设置为44.1 kHz，16位立体声PCM格式。安装后，必须使用控制面板或第三方应用程序更改端点格式。
* 默认设备。在安装时，将使用INF文件中的信息选择设置为默认设备的端点。安装完成后，必须使用“控制面板”或第三方应用程序选择另一个端点作为默认端点。

**注意**    如果您的INF文件未选择安装过程中要设置为默认端点的端点，则客户端应用程序可以使用MMDevice API选择端点。API的选择基于形状因子等级以及端点是渲染端点还是捕获端点。下表显示了选择顺序。

| **渲染等级** | **捕获等级** |
| --- | --- |
| 讲者 | 麦克风 |
| 线路输出 | 输入 |
| SPDIF | SPDIF |

如果您使用MMDevice API选择默认端点，并且可用端点的排名相同，则MMDevice API将按字母顺序排列端点ID，以确定要选择哪个端点作为默认端点。例如，如果音频适配器同时具有输出连接器和输入连接器，并且关联的INF文件在安装时未选择其中一个作为默认设置，则MMDevice API会识别哪个Endpoint ID首先按字母顺序排列并进行设置该连接器为默认连接器。重新启动系统后，该选择将继续存在，因为端点ID是永久的。但是，如果系统中出现更高级别的端点（例如，带有麦克风连接器的第二个适配器），则选择不会持续。

# 动态子设备注册和注销

* 2017/04/20
* 2分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/dynamic-subdeviceregistration-and-unregistration.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/dynamic-subdeviceregistration-and-unregistration.md" \o "1个贡献者)

支持某种形式的插孔存在检测的设备称为动态设备，其插孔必须支持[**KSPROPERTY\_JACK\_DESCRIPTION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-jack-description)属性。以下步骤显示了动态设备的驱动程序用来为这些动态设备创建，注册或注销关联子设备的算法。子设备以过滤器的形式创建。

以下步骤显示了加载音频设备驱动程序时将音频设备插入插孔的情况：

1. 驾驶员使用插孔存在检测来确定是否有设备插入插孔。该驱动程序调用**[PcRegisterSubdevice](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-pcregistersubdevice)**注册了拓扑过滤器[Portcls](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/introduction-to-port-class)。拓扑过滤器注册的结果是创建了一个[**KSCATEGORY\_AUDIO**](https://docs.microsoft.com/windows-hardware/drivers/install/kscategory-audio)接口。
2. 当创建**KSCATEGORY\_AUDIO**接口并且[AudioEndpoint Builder](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-endpoint-builder-algorithm)创建并初始化关联的端点，然后将其状态设置为active 时，将通知音频堆栈。
3. 驱动程序向Portcls注册一个滤波器，并通知音频堆栈。
4. 驱动程序调用[**PcRegisterPhysicalConnection**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-pcregisterphysicalconnection)将波形滤波器与拓扑滤波器连接。然后，此物理连接将在Portcls中注册。
5. 驱动程序将[**KSJACK\_DESCRIPTION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksjack-description)结构的IsConnected成员设置为**TRUE，**以指示已将设备插入插孔。

**注意**    如果音频设备缺少插孔存在检测，则**IsConnected**成员必须始终为**TRUE**。为了确认设备是否支持插孔存在检测，客户端应用程序可以调用[IKsJackDescription2 :: GetJackDescription2](https://go.microsoft.com/fwlink/p/?linkid=143698)来读取[**KSJACK\_DESCRIPTION2**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksjack-description2)结构的JackCapabilities标志。如果此标志的JACKDESC2\_PRESENCE\_DETECT\_CAPABILITY位置1，则表明端点支持插孔存在检测。在那种情况下，**IsConnected**成员的返回值可以解释为插孔的插入状态的准确反映。

以下步骤说明了在加载驱动程序时，如果没有音频设备插入插孔，会发生什么情况：

1. 驾驶员使用插孔存在检测来确定没有设备插入插孔。但是它使用Portcls为该插孔注册了一个拓扑过滤器，并创建了**KSCATEGORY\_AUDIO**接口。
2. 创建**KSCATEGORY\_AUDIO**接口时，将通知音频堆栈。AudioEndpointBuilder查询微型端口驱动程序，以从**KSJACK\_DESCRIPTION**属性确定是否将端点状态设置为未插入。
3. 驱动程序将**KSJACK\_DESCRIPTION**结构的**IsConnected**成员设置为**FALSE，**以指示没有设备插入插孔。

有关音频端点的不同状态的更多信息，请参见“ [音频端点生成器算法”](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-endpoint-builder-algorithm)。

为了符合前面对子设备注册和注销过程的描述，支持插孔存在检测的设备驱动程序必须以下列方式响应插头的插入和拔出：

**设备驱动程序对插入插头的响应**

1. 驱动程序必须调用**PcRegisterSubdevice**来向**Portcls**注册一个滤波器。 **注意**当未在插孔中插入任何设备的情况下加载该驱动程序时，   该驱动程序已在拓扑过滤器上调用了**PcRegisterSubdevice**。
2. 驱动程序必须调用**PcRegisterPhysicalConnection**才能向**Portcls**注册“波形拓扑过滤器”连接。
3. 驱动程序必须将**KSJACK\_DESCRIPTION**结构的**IsConnected**成员设置为**TRUE**。

**设备驱动程序对拔出插头的响应**

1. 驱动程序必须调用[**IUnregisterPhysicalConnection :: UnregisterPhysicalConnection取消**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iunregisterphysicalconnection-unregisterphysicalconnection)注册波形滤波器和拓扑滤波器之间的物理连接。
2. 驱动程序必须调用[**IUnregisterSubdevice :: UnregisterSubdevice**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iunregistersubdevice-unregistersubdevice)来注销波形滤波器。
3. 驱动程序必须设置**KSJACK\_DESCRIPTION**结构**FALSE**的**IsConnected**成员。

**暴露多通道节点**

* 2017/04/20
* 6分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/exposing-multichannel-nodes.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/exposing-multichannel-nodes.md" \o "1个贡献者)

在Windows XP之前的Microsoft Windows版本中，WDM音频驱动程序没有一种简化的方式来公开以下类型的多通道节点：

[**KSNODETYPE\_VOLUME**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-volume)

[**KSNODETYPE\_MUTE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-mute)

[**KSNODETYPE\_TONE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-tone)

特别是，不存在用于显式查询节点所支持的通道数的机制。尽管存在解决此问题的方法，但它们也有缺点。例如，客户端可以使用[**KSPROPERTY\_AUDIO\_VOLUMELEVEL**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-volumelevel)属性来迭代地查询卷节点（[**KSNODETYPE\_VOLUME**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-volume)）以获得每个通道-0，-1的音量级别，依此类推-直到请求返回错误，表明不再存在任何通道。但是，此技术需要多个查询，并且效率太低，无法处理较新的多声道音频设备。在Windows XP和更高版本的操作系统中，可以通过在[**KSPROPERTY\_MEMBERSHEADER**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksproperty_membersheader)的**Flags**成员中定义两个其他标志位来解决此限制。 属性处理程序根据基本支持查询输出的结构：

* KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_MULTICHANNEL

在节点上进行基本支持属性请求期间，处理程序将设置此标志位，以指示**KSPROPERTY\_MEMBERSHEADER的MembersCount**成员包含该节点支持的通道数。对于Windows Vista和更高版本的Windows操作系统，必须为每个通道属性设置此标志。

* KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_UNIFORM

处理程序在此标志位和KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_MULTICHANNEL标志位之间执行按位“或”操作，以指示单个属性值均匀地应用于节点中的所有通道。例如，如果硬件仅为所有通道提供单个音量级别控制，则音量节点的基本支持处理程序将设置KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_UNIFORM标志以指示此限制。如果未设置此标志，则可以独立于其他通道的音量水平来控制每个通道的音量水平。

**注意**    Windows Vista操作系统不使用KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_UNIFORM标志。

在Windows XP和更高版本的微型端口驱动程序中，多通道卷节点的属性处理程序应将KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_MULTICHANNEL位置为响应KSPROPERTY\_AUDIO\_VOLUMELEVEL基本支持查询。该处理程序返回一个**[KSPROPERTY\_STEPPING\_LONG](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksproperty_stepping_long)**结构数组-节点公开的每个通道一个结构-并将**MemberSize**设置为**sizeof**（KSPROPERTY\_STEPPING\_LONG）。每个数组元素描述一个通道的最小和最大音量级别以及该范围内连续值之间的增量。可以为每个单独的通道指定一个不同的范围，以便可以正确显示范围不均匀的通道。例如，低音炮通道的范围可能与其他通道的范围不同。

下面的代码示例演示如何处理具有不均匀属性值[的音频属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/basic-support-queries-for-audio-properties)的[基本支持查询](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/basic-support-queries-for-audio-properties)。下面第一行代码中的变量pDescription指向处理程序向其中写入基本支持信息的数据缓冲区开头的**[KSPROPERTY\_DESCRIPTION](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksproperty_description)**结构：

C ++复制

//

// Fill in the members header.

//

PKSPROPERTY\_MEMBERSHEADER pMembers = PKSPROPERTY\_MEMBERSHEADER(pDescription + 1);

pMembers->MembersFlags = KSPROPERTY\_MEMBER\_STEPPEDRANGES;

pMembers->MembersSize = sizeof(KSPROPERTY\_STEPPING\_LONG);

pMembers->MembersCount = ulNumChannels;

pMembers->Flags = KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_MULTICHANNEL;

//

// Fill in the stepped range with the driver default.

//

PKSPROPERTY\_STEPPING\_LONG pRange = PKSPROPERTY\_STEPPING\_LONG(pMembers + 1);

pRange->Reserved = 0;

for (ULONG i=0; i<ulNumChannels; i++)

{

pRange[i].Bounds.SignedMinimum = ulChannelMin[i];

pRange[i].Bounds.SignedMaximum = ulChannelMax[i];

pRange[i].SteppingDelta = ChannelStepping[i];

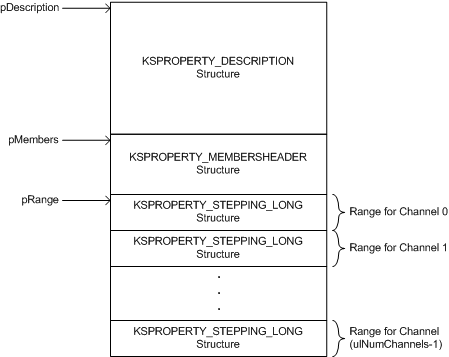
}

pPropertyRequest->ValueSize = sizeof(KSPROPERTY\_DESCRIPTION) +

sizeof(KSPROPERTY\_MEMBERSHEADER) +

ulNumChannels \* sizeof(KSPROPERTY\_STEPPING\_LONG);

下图显示了此示例的数据缓冲区的布局。显示的pDescription，pMembers和pRange指针指向它们在缓冲区中的各自偏移量。



对于此示例，处理程序将**MemberCount**设置为**ulNumChannels**，即通道数。范围数组的字节大小为

**会员人数** \* **会员人数**

请注意，如果在此示例中设置了KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_UNIFORM标志，则处理程序会将数组中的所有KSPROPERTY\_STEPPING\_LONG结构设置为相同范围。

音调节点的[**KSPROPERTY\_AUDIO\_BASS**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-bass)，[**KSPROPERTY\_AUDIO\_TREBLE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-treble)或[**KSPROPERTY\_AUDIO\_MID**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-mid)属性的基本支持处理程序以类似的方式运行。

如果多通道节点的每个通道的属性值的类型均为BOOL，则基本支持处理程序必须填写步进范围数组的值。在这种情况下，处理程序将成员设置为以下代码示例中所示的值。这种类型的属性的两个例子是**[KSPROPERTY\_AUDIO\_MUTE](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-mute)**静音节点的属性和[**KSPROPERTY\_AUDIO\_BASS\_BOOST**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-bass-boost)音调节点的属性。

下面的代码示例显示在每个通道属性值为BOOL类型的属性的情况下，如何处理多通道节点的基本支持请求：

C ++复制

//

// Fill in the members header.

//

PKSPROPERTY\_MEMBERSHEADER pMembers = PKSPROPERTY\_MEMBERSHEADER(pDescription + 1);

pMembers->MembersFlags = KSPROPERTY\_MEMBER\_STEPPEDRANGES;

pMembers->MembersSize = sizeof (KSPROPERTY\_STEPPING\_LONG);

pMembers->MembersCount = ulNumChannels;

pMembers->Flags = KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_MULTICHANNEL;

pPropertyRequest->ValueSize = sizeof(KSPROPERTY\_DESCRIPTION) +

sizeof(KSPROPERTY\_MEMBERSHEADER) +

ulNumChannels \* sizeof(KSPROPERTY\_STEPPING\_LONG);

//

// Fill in the stepped range with values in FOR loop.

//

PKSPROPERTY\_STEPPING\_LONG pRange = PKSPROPERTY\_STEPPING\_LONG(pMembers + 1);

pRange->Reserved = 0;

for (ULONG i=0; i<ulNumChannels; i++)

{

pRange[i].Bounds.SignedMinimum = 0;

pRange[i].Bounds.SignedMaximum = 1;

pRange[i].SteppingDelta = 1;

}

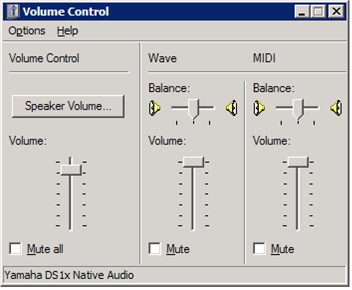
请注意，在前面的代码示例中，FOR循环使用零（0）和一（1）来设置每个通道范围的最小值和最大值。这是因为我们正在配置每个通道属性值为BOOL类型的多通道节点。

如果通道属性是统一的，则可以在KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_UNIFORM标志和KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_MULTICHANNEL标志之间执行按位或运算，并将结果分配给pMembers-> Flags成员。此值用于指示硬件在节点中的所有通道上均匀地应用相同的属性值。

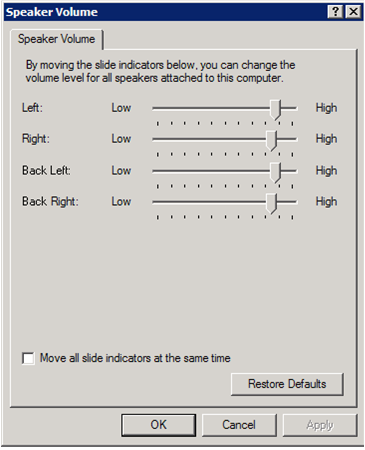
使用KSPROPERTY\_MEMBER\_FLAG\_UNIFORM和KSPROPERTY\_MEMBER\_FLAG\_MULTICHANNEL标志可以消除将通道分成几对并为每对通道显示单独的立体声音量节点的需要，就像在Windows驱动程序工具包（WDK）中的Ac97示例驱动程序中所做的那样。因为Windows XP之前的Windows版本不支持这些标志，所以驱动程序的基本支持处理程序必须使用[IPortClsVersion](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iportclsversion)接口来查询Portcls.sys版本，以确定是否使用这些标志。

拓扑解析器（在内核模式[WDMAud系统驱动程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/user-mode-wdm-audio-components#wdmaud_system_driver) Wdmaud.sys中）从其WDM音频驱动程序获取音频设备的拓扑。解析器通过旧版Windows Multimedia 混合器 API 将该设备公开为传统混合器设备。在Windows XP和更高版本中，WDMAud使用KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_MULTICHANNEL标志来确定要在cChannels中报告的通道数MIXERLINE结构的成员。此外，如果节点的基本支持处理程序指定了KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_UNIFORM标志，则WDMAud在相应的MIXERCONTROL结构中设置MIXERCONTROL\_CONTROLF\_UNIFORM标志。通过此标志，应用程序可以确定是否可以通过主控件单独调整每个通道还是可以统一调整所有通道。有关MIXERCONTROL，MIXERLINE和混合器 API的更多信息，请参见Microsoft Windows SDK文档。

在Windows XP和更高版本中，SndVol32音量控制程序（请参阅[SysTray和SndVol32](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/systray-and-sndvol32)）显示用于多通道设备的控件，如下图所示。



如果SndVol32检测到一条线路具有两个以上的通道，它将用标有“ **扬声器音量”**的按钮替换正常的声像控制，该按钮显示在上图中主音量滑块上方。单击“ **扬声器音量”**按钮将弹出一个对话框，显示特定线路的所有通道的控件，如下图所示。



由于**混音器** API通过数字公开通道，因此它会从Windows多媒体控制面板（Mmsys.cpl）的“ **高级音频属性”**对话框中当前选择的扬声器配置中推断出通道名称。

例如，如果设备在线上公开了四个声道，并且用户选择了“四声道扬声器”，则声道名称将为“左”（声道0），“右”（声道1），“后左”（声道2） ）和“右后方”（通道3），如上图所示。将扬声器配置更改为“环绕声扬声器”将导致“左”（通道0），“右”（通道1），“前中心”（通道2）和“后中心”（通道3）的通道映射）。

在驱动程序级别，KSPROPERTY\_AUDIO\_CHANNEL\_CONFIG属性使用掩码值KSAUDIO\_SPEAKER\_QUAD或KSAUDIO\_SPEAKER\_SURROUND分别表示四声道或环绕扬声器配置。头文件Ksmedia.h定义这些值如下：

C ++复制

#define KSAUDIO\_SPEAKER\_QUAD (SPEAKER\_FRONT\_LEFT | SPEAKER\_FRONT\_RIGHT | \

SPEAKER\_BACK\_LEFT | SPEAKER\_BACK\_RIGHT)

#define KSAUDIO\_SPEAKER\_SURROUND (SPEAKER\_FRONT\_LEFT | SPEAKER\_FRONT\_RIGHT | \

SPEAKER\_FRONT\_CENTER | SPEAKER\_BACK\_CENTER)

任一掩码都包含四个位，用于指定四个通道的扬声器位置。在两种情况下，KSPROPERTY\_AUDIO\_VOLUMELEVEL属性都将这四个相同的通道分别标识为通道0、1、2和3。

如果节点的基本支持处理程序将KSPROPERTY\_MEMBER\_FLAG\_BASICSUPPORT\_UNIFORM标志位置1，则“ **扬声器音量”**对话框中显示的滑块将与对任何单个滑块所做的更改一致地移动。

**引脚类别属性**

* 2017/04/20
* 4分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/pin-category-property.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/pin-category-property.md" \o "1个贡献者)

用于USB音频设备，IEEE 1394音频设备和内部总线上的音频设备的Microsoft Windows驱动程序模型（WDM）音频驱动程序都将它们的设备表示为带引脚的KS过滤器。WDM音频驱动程序为其支持的每种引脚类型维护一个[**KSPIN\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-kspin_descriptor)结构。在此结构中，驱动程序存储引脚类型的[KSPROPSETID\_Pin](https://docs.microsoft.com/windows-hardware/drivers/stream/kspropsetid-pin)属性。这些属性中包括**[KSPROPERTY\_PIN\_CATEGORY](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-category)**属性。对此属性的请求从**KSPIN\_DESCRIPTOR**结构的**类别中**检索KS引脚类别GUID会员。该GUID指示引脚提供的功能的一般类别。例如，特定的插针类别GUID KSNODETYPE\_HEADPHONES将一个插针标识为耳机的输出插孔。

对于内部总线上的波形音频设备（例如PCI），PortCls微型端口驱动程序包含一个引脚描述符数组，每个描述符都描述代表该设备的过滤器中的引脚类型。每个引脚描述符都是**[PCPIN\_DESCRIPTOR](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcpin_descriptor)**结构，其中包含带有引脚类别GUID 的嵌入式[**KSPIN\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-kspin_descriptor)结构。从客户[**端接**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-category)收到[**KSPROPERTY\_PIN\_CATEGORY**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-category)属性请求后，端口驱动程序会从微型端口驱动程序的引脚描述符中为指定的引脚类型检索引脚类别GUID。有关引脚描述符的更多信息，请参见[引脚工厂](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/pin-factories)。

USB音频设备具有一定数量的端子，数字流和模拟信号可以通过这些端子进入和退出设备。当构造表示USB音频设备的KS过滤器时，[USBAudio类系统驱动程序会将](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-mode-wdm-audio-components#usbaudio_class_system_driver)设备上的端子转换为过滤器上的引脚。头文件Ksmedia.h定义了每个USB终端类型标识符到KS引脚类别GUID的映射。以下六个表显示了端子类型标识符及其对应的引脚类别GUID。

**输入端子类型**

| **USB端子ID** | **KS引脚类别GUID** |
| --- | --- |
| 0x0201 | KSNODETYPE\_MICROPHONE |
| 0x0202 | KSNODETYPE\_DESKTOP\_MICROPHONE |
| 0x0203 | KSNODETYPE\_PERSONAL\_MICROPHONE |
| 0x0204 | KSNODETYPE\_OMNI\_DIRECTIONAL\_MICROPHONE |
| 0x0205 | KSNODETYPE\_MICROPHONE\_ARRAY |
| 0x0206 | KSNODETYPE\_PROCESSING\_MICROPHONE\_ARRAY |

**输出端子类型**

| **USB端子ID** | **KS引脚类别GUID** |
| --- | --- |
| 0x0301 | KSNODETYPE\_SPEAKER |
| 0x0302 | KSNODETYPE\_HEADPHONES |
| 0x0303 | KSNODETYPE\_HEAD\_MOUNTED\_DISPLAY\_AUDIO |
| 0x0304 | KSNODETYPE\_DESKTOP\_SPEAKER |
| 0x0305 | KSNODETYPE\_ROOM\_SPEAKER |
| 0x0306 | KSNODETYPE\_COMMUNICATION\_SPEAKER |
| 0x0307 | KSNODETYPE\_LOW\_FREQUENCY\_EFFECTS\_SPEAKER |

**双向端子类型**

| **USB端子ID** | **KS引脚类别GUID** |
| --- | --- |
| 0x0401 | KSNODETYPE\_HANDSET |
| 0x0402 | KSNODETYPE\_HEADSET |
| 0x0403 | KSNODETYPE\_SPEAKERPHONE\_NO\_ECHO\_REDUCTION |
| 0x0404 | KSNODETYPE\_ECHO\_SUPPRESSING\_SPEAKERPHONE |
| 0x0405 | KSNODETYPE\_ECHO\_CANCELING\_SPEAKERPHONE |

**电话终端类型**

| **USB端子ID** | **KS引脚类别GUID** |
| --- | --- |
| 0x0501 | KSNODETYPE\_PHONE\_LINE |
| 0x0502 | KSNODETYPE\_TELEPHONE |
| 0x0503 | KSNODETYPE\_DOWN\_LINE\_PHONE |

**外部端子类型**

| **USB端子ID** | **KS引脚类别GUID** |
| --- | --- |
| 0x0601 | KSNODETYPE\_ANALOG\_CONNECTOR |
| 0x0602 | KSNODETYPE\_DIGITAL\_AUDIO\_INTERFACE |
| 0x0603 | KSNODETYPE\_LINE\_CONNECTOR |
| 0x0604 | KSNODETYPE\_LEGACY\_AUDIO\_CONNECTOR |
| 0x0605 | KSNODETYPE\_SPDIF\_INTERFACE |
| 0x0606 | KSNODETYPE\_1394\_DA\_STREAM |
| 0x0607 | KSNODETYPE\_1394\_DV\_STREAM\_SOUNDTRACK |

**嵌入式功能端子类型**

| **USB端子ID** | **KS引脚类别GUID** |
| --- | --- |
| 0x0701 | KSNODETYPE\_LEVEL\_CALIBRATION\_NOISE\_SOURCE |
| 0x0702 | KSNODETYPE\_EQUALIZATION\_NOISE |
| 0x0703 | KSNODETYPE\_CD\_PLAYER |
| 0x0704 | KSNODETYPE\_DAT\_IO\_DIGITAL\_AUDIO\_TAPE |
| 0x0705 | KSNODETYPE\_DCC\_IO\_DIGITAL\_COMPACT\_CASSETTE |
| 0x0706 | KSNODETYPE\_MINIDISK |
| 0x0707 | KSNODETYPE\_ANALOG\_TAPE |
| 0x0708 | KSNODETYPE\_PHONOGRAPH |
| 0x0709 | KSNODETYPE\_VCR\_AUDIO |
| 0x070A | KSNODETYPE\_VIDEO\_DISC\_AUDIO |
| 0x070B | KSNODETYPE\_DVD\_AUDIO |
| 0x070C | KSNODETYPE\_TV\_TUNER\_AUDIO |
| 0x070D | KSNODETYPE\_SATELLITE\_RECEIVER\_AUDIO |
| 0x070E | KSNODETYPE\_CABLE\_TUNER\_AUDIO |
| 0x070F | KSNODETYPE\_DSS\_AUDIO |
| 0x0710 | KSNODETYPE\_RADIO\_RECEIVER |
| 0x0711 | KSNODETYPE\_RADIO\_TRANSMITTER |
| 0x0712 | KSNODETYPE\_MULTITRACK\_RECORDER |
| 0x0713 | KSNODETYPE\_SYNTHESIZER |

有关USB端子类型标识符的更多信息，请参见[USB实施者论坛](https://go.microsoft.com/fwlink/p/?linkid=8780)网站上提供的*端子类型*的*通用串行总线设备类定义*（版本1.0）。

上表中的所有引脚类别GUID的参数名称的形式均为KSNODETYPE\_ *XXX*。请注意，KS节点类型的GUID也具有KSNODETYPE\_ *XXX*参数名称。这种命名约定在引脚类别GUID和节点类型GUID之间产生了混淆的可能性。幸运的是，几乎每个KSNODETYPE\_ *XXX*参数都可以标识引脚类别或节点类型，但不能同时标识两者。规则的一个例外是**[KSNODETYPE\_SYNTHESIZER](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-synthesizer)**，它可以根据上下文标识引脚类别或节点类型。有关节点类型GUID的列表，请参见“ [音频拓扑节点”](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-topology-nodes)。

实例化USB音频设备时，USBAudio类系统驱动程序向设备查询其内部拓扑，包括其端子。借助此信息，USBAudio驱动程序将构造一个代表设备的过滤器，并将每个端子转换为过滤器上的相应引脚。在此过程中，驱动程序将每个USB端子类型标识符转换为对应的KS引脚类别GUID，该GUID是先前表中的GUID之一。驱动程序构造一个**[KSPIN\_DESCRIPTOR](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-kspin_descriptor)**结构来描述该引脚，并将引脚类别GUID写入该结构。

PortCls微型端口驱动程序不一定只使用前面六个表中显示的类别GUID。例如，驱动程序可能定义并使用自定义引脚类别GUID来描述其功能类别超出表中类别之外的引脚类型。自然，自定义引脚类别GUID仅对识别该GUID的客户端有用。

音频子系统在系统注册表中维护引脚类别GUID及其相关联的友好名称的列表。GUID和友好名称存储在注册表路径HKLM \ SYSTEM \ CurrentControlSet \ Control \ MediaCategories中。媒体类安装程序将GUID名称对从位于主Windows文件夹的Inf子文件夹中的Ks.inf文件复制到注册表中（例如，C：\ Windows \ Inf \ Ks.inf）。

在Windows Vista和更高版本中，操作系统使用引脚类别将友好名称与音频终结点设备关联。有关如何将友好名称与音频端点设备相关联的更多信息，请参见音频端点设备的[友好名称](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/friendly-names-for-audio-endpoint-devices)。

在Windows XP，Windows 2000和Windows Millennium Edition中，操作系统仅有限地使用了引脚类别。该[WDMAud系统驱动](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/user-mode-wdm-audio-components" \l "wdmaud_system_driver)的行动代表混频器API翻译销类的GUID到MIXERLINE\_COMPONENTTYPE\_的*XXX*客户端应用程序使用的值。WDMAud仅识别出现在前六个表中的引脚类别GUID的子集。此外，由于历史原因，WDMAud可以识别两个引脚类别的GUID，即KSCATEGORY\_AUDIO和PINNAME\_CAPTURE，它们没有出现在表中。有关将引脚类别转换为混频器线的更多信息，请参见[拓扑引脚](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/topology-pins)。有关混合器API的信息，请参阅Windows SDK文档。

**音频端点设备的友好名称**

* 2017/04/20
* 5分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/friendly-names-for-audio-endpoint-devices.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/friendly-names-for-audio-endpoint-devices.md" \o "2位贡献者)

* + [[https://github.com/JoelCorley.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/friendly-names-for-audio-endpoint-devices.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/friendly-names-for-audio-endpoint-devices.md" \o "2位贡献者)

在Windows Vista，Windows Server 2008和更高版本的Windows中，音频子系统支持音频终结点设备的概念，例如扬声器，耳机，麦克风和CD播放器。音频端点的这一概念有助于创建用户友好的音频应用程序，该应用程序的用户界面引用用户直接操作的端点设备。这些端点具有友好名称，例如“扬声器”，“耳机”，“麦克风”和“ CD播放器”，应用程序可以在它们的用户界面中显示它们。有关端点设备的更多信息，请参见[音频端点设备](https://go.microsoft.com/fwlink/p/?linkid=130876)。

音频子系统将音频适配器上的即插即用（PnP）设备建模为KS滤波器。数据流通过KS引脚进入和离开过滤器。桥接引脚是KS引脚，音频端点设备通过KS引脚连接到KS滤波器。有关桥脚的更多信息，请参见[音频滤波器图](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-filter-graphs)。

音频子系统通过检查终结点设备连接到的桥接引脚的属性来获取有关音频终结点设备的信息。这样的属性之一就是[引脚类别属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/pin-category-property)（[**KSPROPERTY\_PIN\_CATEGORY**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-category)）。

对于每个KS过滤器，适配器驱动程序均提供[**PCPIN\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcpin_descriptor)结构表，该表描述了过滤器上KS引脚的属性。引脚类别GUID存储在PCPIN\_DESCRIPTOR结构的**KsPinDescriptor.Category**成员中。对于桥接引脚，引脚类别GUID的值指示连接到桥接引脚的端点的类型。例如，引脚类别GUID KSNODETYPE\_MICROPHONE指示桥接引脚连接到麦克风，GUID KSNODETYPE\_SPEAKER指示桥接引脚连接到扬声器，依此类推。KSNODETYPE\_ *XXX* GUID在Ksmedia.h头文件中定义。

另外，**PCPIN\_DESCRIPTOR**包括一个GUID，可用于通过唯一名称标识引脚。此引脚名称GUID存储在PCPIN\_DESCRIPTOR结构的**KsPinDescriptor.Name**成员中。（[**KSPROPERTY\_PIN\_NAME**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-name)）属性使用此名称GUID将在注册表中找到的友好名称与该引脚关联。

音频子系统调用**KSPROPERTY\_PIN\_NAME**属性，以将友好名称与音频端点关联。KS通过首先在注册表中搜索描述**KsPinDescriptor.Name** GUID 的unicode字符串来处理此请求。如果KS未找到条目，它将在注册表中搜索描述**KsPinDescriptor.Category** GUID 的unicode字符串。

从**Windows 10 REDSTONE 5**开始，在搜索注册表时，KS首先在设备的软件密钥中查找条目。这是由INF通过设备驱动程序的INF的[Models]部分引用的AddReg部分创建的。AddReg部分使用HKR \ MediaCategories键构造这些条目。这样，驱动程序开发人员就可以为名称和类别GUID创建设备特定的友好名称，而无论该GUID是否为设备唯一。

如果尚未在设备的软件密钥中安装条目，或者驱动程序在**Windows 10 REDSTONE 5**之前的操作系统上运行，则KS在HKLM \ SYSTEM \ CurrentControlSet \ Control \ MediaCategories注册表项下查找。第二个键被视为全局名称空间。从**Windows 10 REDSTONE 5**开始，此空间保留用于全局定义，不应由新驱动程序修改。在将来的OS版本中，将不支持修改该键下的条目。

暴露带有标准类别GUID的引脚的音频设备应在设备INF中包含/需要收件箱KS.INF或KSCAPTUR.INF名称注册。这些收件箱INF包含驱动程序可能希望填充的预定义类别GUID的默认友好名称定义。这些是在PCPIN\_DESCRIPTOR结构的**KsPinDescriptor.Category**成员中找到的相同GUID 。例如，类别GUID KSNODETYPE\_MICROPHONE条目具有关联的友好名称“麦克风”，类别GUID KSNODETYPE\_SPEAKER条目具有关联的友好名称“扬声器”，依此类推。

类别和名称GUID的GUID和友好名称存储在HKR \ MediaCategories下，而全局定义HKLM \ SYSTEM \ CurrentControlSet \ Control \ MediaCategories路径下。对于注册表中的每个GUID名称对，GUID字符串都用作MediaCategories项下的子项。在GUID键下，友好名称在“名称”变量下的Unicode字符串值。

如果音频子系统定义的友好名称和引脚类别中的任何一个都不能充分描述您的设备，则可以定义自己的引脚类别和名称GUID，并在INF中将友好名称与它们相关联。为确保您的密码类别GUID是唯一的，请使用Uuidgen.exe之类的实用程序来生成GUID。接下来，修改安装音频适配器的INF文件，以将针型GUID和友好名称写入注册表路径HKR \ MediaCategories。下面的代码示例显示了一个INF文件的片段，该片段向注册表添加了两个针类GUID及其关联的友好名称：

信息复制

[Manufacturer]

MyOEMName=Unicorn,NTamd64

[Unicorn.NTamd64]

MyDeviceName=MyDevice,Root\MyDevice

[MyDevice.NT]

Include=ks.inf, kscaptur.inf

Needs=KS.Registration, KSCAPTUR.Registration.NT

CopyFiles=MyDevice.CopyFiles

AddReg=PinNameRegistration

...

[PinNameRegistration]

HKR,%MediaCategories%\%GUID.MyNewEndpointCategory%,Name,,%Name.MyNewEndpointCategory%

HKR,%MediaCategories%\%GUID.MyNewEndpointName%,Name,,%Name.MyNewEndpointName%

...

[Strings]

MyOEMName="Unicorns Inc."

MyDeviceName="Sparkly Unicorn"

MediaCategories="MediaCategories"

GUID.MyNewEndpointCategory="{B72FBD1A-4634-4240-B207-0E6B52F3701C}"

GUID.MyNewEndpoint\_2="{71DD3A5D-E303-49A0-ACEE-908634AA9520}"

Name.MyNewEndpointCategory="Unicorn"

Name.MyNewEndpointName="Fred the Unicorn"

这两个GUID字符串都是由Uuidgen.exe生成的。

应用程序可以使用设备的**IPropertyStore**接口访问音频终结点设备的属性。该界面使用Functiondiscoverykeys\_devpkey.h和Mmdeviceapi.h头文件中定义的属性键来标识属性。应用程序可以使用**PKEY\_Device\_FriendlyName**属性键来检索端点设备的友好名称。对于空间受限的用户界面，可以使用**PKEY\_Device\_DeviceDesc**属性键来检索友好名称的较短版本。有关这些属性键的更多信息，请参见[IMMDevice :: OpenPropertyStore](https://go.microsoft.com/fwlink/p/?linkid=155067)。

一个**IPropertyStore**接口实例维护一个音频端点设备的永久属性存储。该属性存储从与注册表路径HKLM \ SYSTEM \ CurrentControlSet \ Control \ MediaCategories中的KS引脚类别GUID关联的友好名称字符串中复制**PKEY\_Device\_DeviceDesc**属性键的初始值。应用程序可以从属性存储中读取**PKEY\_Device\_DeviceDesc**属性值（名称字符串），但不能更改该值。但是，用户可以使用Windows多媒体控制面板Mmsys.cpl修改名称。例如，在Windows Vista中，可以使用以下步骤来修改呈现终结点设备的名称：

1. 要运行Mmsys.cpl，请打开“命令提示符”窗口，然后输入以下命令：

安慰复制

mmsys.cpl

（或者，可以通过右键单击位于任务栏右侧的通知区域中的扬声器图标，然后单击“ **播放设备”**来运行Mmsys.cpl 。）

1. 单击渲染设备的名称，然后单击**属性**。
2. 在“属性”窗口中，单击“ **常规”**选项卡。友好名称应显示在属性表顶部的文本框中。您可以编辑好记的名称，然后单击“ **确定”**保存更改。

前面的步骤更改了音频端点设备的属性存储中存储的友好名称。这些步骤不会影响与属于相同KS引脚类别的其他音频终结点设备关联的友好名称。它们对可能直接查询KS名称的任何组件也没有影响。

**音频位置属性**

* 2017/04/20
* 7分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/audio-position-property.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/audio-position-property.md" \o "1个贡献者)

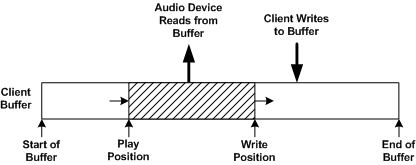
音频驱动程序的客户端使用[**KSPROPERTY\_AUDIO\_POSITION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-position)属性获取并设置音频流中的当前位置。该属性使用**[KSAUDIO\_POSITION](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-ksaudio_position)**结构来描述当前位置。该结构包含两个成员：**PlayOffset**和**WriteOffset**。

的**PlayOffset**和**WriteOffset**构件限定当前保留用于专有使用音频设备的客户端缓冲的区域的边界。客户端必须假设设备当前可能正在访问此区域中包含的任何数据。因此，客户端必须仅访问缓冲区位于该区域之外的部分。该区域的边界随着流的前进而移动。

如果循环了客户端缓冲区（即，流类型为[**KSINTERFACE\_STANDARD\_LOOPED\_STREAMING**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksinterface-standard-looped-streaming)），则**PlayOffset**和**WriteOffset**是缓冲区相对偏移量。也就是说，它们被指定为从循环客户端缓冲区开始的字节偏移量。当任一偏移量增加到缓冲区的末尾时，它会回绕到缓冲区的开头。（缓冲区开始处的偏移量为零。）因此，两个偏移量都不会超过缓冲区大小。

如果客户端缓冲区是非循环的（即，流类型为[**KSINTERFACE\_STANDARD\_STREAMING**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksinterface-standard-streaming)），则**PlayOffset**和**WriteOffset**是相对于流的偏移量。也就是说，它们被指定为从流开始的字节偏移量。这些偏移量可以看作是理想缓冲区的偏移量，该缓冲区包含整个流，并且从头到尾是连续的。

对于渲染流，**PlayOffset**成员指定流的播放位置，而**WriteOffset**成员指定流的写入位置。下图显示了客户端缓冲区中的播放和写入位置。



播放位置是当前正在播放的样本（即锁存在数模转换器或DAC输入处的样本）的字节偏移。写入位置是客户端可以安全地写入缓冲区的位置。在流播放时，播放和写入位置在上图中从左向右移动。客户的写操作必须保持在写位置之前。另外，如果缓冲区是循环的，则客户端的写操作绝不能超过播放位置。

尽管WaveCyclic或WavePci端口驱动程序依靠微型端口驱动程序来跟踪播放位置，但是端口驱动程序仍可以跟踪写入位置。WaveCyclic和WavePci端口驱动程序如下更新写入位置：

* **WaveCyclic**

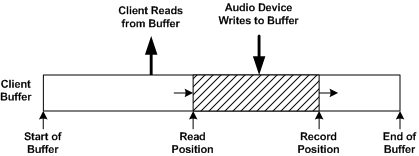
每次WaveCyclic端口驱动程序调用[**IDmaChannel :: CopyTo**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-idmachannel-copyto)将新的数据块复制到循环缓冲区（从客户端缓冲区）时，写入位置都会前进到数据块中最后一个字节的位置（在客户端缓冲区中）。 。

* **WavePci**

默认情况下，每次WavePci微型端口驱动程序调用[**IPortWavePciStream :: GetMapping**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportwavepcistream-getmapping)以获取新的映射（部分客户端缓冲区）并且调用成功时，写入位置将前进到最后一个字节的位置（在客户端缓冲区中）在新的映射中。

如果WavePci微型端口驱动程序通过为端口驱动程序指定预取偏移量来覆盖默认行为，则当前写入位置始终等于当前播放位置和预取偏移量之和。有关更多信息，请参见[预取偏移量](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/prefetch-offsets)。

对于捕获流，**PlayOffset**成员指定流的记录位置，而**WriteOffset**成员指定流的读取位置。下图显示了客户端缓冲区中的记录和读取位置。



记录位置是要锁存在模数转换器或ADC的输出中的最新采样的字节偏移。（此位置指定音频设备的DMA引擎最终将样本写入其中的缓冲区位置。）读取位置是客户端无法安全地从缓冲区读取的位置。随着流记录的进行，读和记录位置在上图中从左向右前进。客户的读取必须落后于读取位置。此外，如果缓冲区循环，则客户端的读取必须保持在记录位置之前。

尽管WaveCyclic或WavePci端口驱动程序依靠微型端口驱动程序来跟踪记录位置，但端口驱动程序仍可以跟踪读取位置。WaveCyclic和WavePci端口驱动程序如下更新读取位置：

* **WaveCyclic**

每次WaveCyclic端口驱动程序调用[**IDmaChannel :: CopyFrom**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-idmachannel-copyfrom)从循环缓冲区复制新的数据块（到客户端缓冲区）时，读取位置都会前进到数据块中最后一个字节的位置（在客户端缓冲区中）。 。

* **WavePci**

每次WavePci微型端口驱动程序调用[**IPortWavePciStream :: ReleaseMapping**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportwavepcistream-releasemapping)释放先前获取的（客户端缓冲区一部分的）映射时，读取位置都会前进到释放的映射中最后一个字节的位置（在客户端缓冲区中）。

微型端口驱动程序不需要为KSPROPERTY\_AUDIO\_POSITION属性请求实现处理程序例程。而是，WaveCyclic和WavePci端口驱动程序代表微型端口驱动程序处理这些请求。当处理获取属性请求时，WaveCyclic或WavePci端口驱动程序已经具有计算**WriteOffset**值所需的所有信息，但仍需要微型端口驱动程序提供的信息来计算**PlayOffset**值。要获取此信息，端口驱动程序调用微型端口驱动程序的[**IMiniportWaveCyclicStream :: GetPosition**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavecyclicstream-getposition)或[**IMiniportWavePciStream :: GetPosition**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavepcistream-getposition)方法。

对于渲染流，**GetPosition**方法检索播放位置-当前正在通过DAC播放的样本的字节偏移量。对于捕获流，**GetPosition**方法检索记录位置-ADC将捕获的最新样本的字节偏移。

请注意，通过**GetPosition**调用获取的偏移值是与当前通过扬声器插孔发送的信号相对应的播放位置，或者是与当前通过麦克风插孔接收的信号相对应的记录位置。它不是DMA位置。（DMA位置是音频设备中的DMA引擎当前正在向DMA缓冲区传输或从DMA缓冲区传输的样本的字节偏移。）

某些音频硬件包含一个位置寄存器，以跟踪每个DAC或ADC中当前样本的字节偏移，在这种情况下，**GetPosition**方法只是为适当的流检索位置寄存器的内容。其他音频硬件只能为驱动器提供DMA位置，在这种情况下，**GetPosition**方法必须通过考虑当前DMA位置和内部的缓冲延迟来提供DAC或ADC中样本字节偏移的最佳估计。设备。

尽管WaveCyclic或WavePci端口驱动程序中的属性处理程序必须区分循环缓冲区和非循环缓冲区来确定是提供相对于流还是相对于缓冲区的字节偏移，但是此详细信息（即缓冲区是循环还是非循环）是透明的到微型端口驱动程序。

该**IMiniportWaveCyclicStream ::为getPosition**方法总是报告相对缓冲播放或记录位置，无论客户端缓存是否正在循环或nonlooped。如果循环了客户端缓冲区，则属性处理程序会将由微型端口驱动程序报告的相对于缓冲区的位置转换为客户端缓冲区的偏移量，该位置表示为循环缓冲区中的偏移量，然后将其写入**PlayOffset**成员。如果客户端缓冲区是非循环的，则在将其相对于播放的位置写入**PlayOffset**成员之前，属性处理程序会将相对于缓冲区的播放位置转换为相对于流的播放位置。

该**IMiniportWavePciStream ::为getPosition**方法总是报告相对流播放或记录位置，无论客户端缓存是否正在循环或nonlooped。如果客户端缓冲区被循环，则属性处理程序将流相对播放位置转换为缓冲区相对播放位置（表示为客户端缓冲区中的偏移量），然后再将其写入属性请求中KSAUDIO\_POSITION结构中的**PlayOffset**成员。如果客户端缓冲区是非循环的，则属性处理程序会将流相对位置写入**PlayOffset**成员。

流初始化后，播放或记录位置立即为零。转换为KSSTATE\_STOP状态（请参阅[**KSSTATE**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ne-ks-ksstate)）会将位置重置为零。当从KSSTATE\_RUN到KSSTATE\_PAUSE或KSSTATE\_ACQUIRE的转换暂停流时，该位置将冻结。当流从KSSTATE\_PAUSE或KSSTATE\_ACQUIRE转换回KSSTATE\_RUN时，它将解冻。

有关WaveCyclic和WavePci微型端口驱动程序的**GetPosition**方法的实现示例，请参阅Windows驱动程序工具包（WDK）中的示例音频驱动程序。

**引脚数据范围和交叉点属性**

* 2017/04/20
* 2分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/pin-data-range-and-intersection-properties.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/pin-data-range-and-intersection-properties.md" \o "1个贡献者)

几个属性请求提供了有关音频设备能够在其输入和输出引脚上处理的音频流的数据格式的信息。

引脚能够支持的音频流数据格式以[**KSDATARANGE**](https://docs.microsoft.com/previous-versions/ff561658(v=vs.85))派生结构的[**KSMULTIPLE\_ITEM**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksmultiple_item)数组表示。通过过滤器上的以下三个[KSPROPSETID\_Pin](https://docs.microsoft.com/windows-hardware/drivers/stream/kspropsetid-pin)属性公开了引脚数据范围支持：

[**KSPROPERTY\_PIN\_DATARANGES**](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-dataranges) 该属性报告的数据范围是静态的，代表所有受支持的格式。通常，数据范围包含在适配器驱动程序的静态数组中。 **[KSPROPERTY\_PIN\_CONSTRAINEDDATARANGES](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-constraineddataranges)** 此属性报告动态的数据范围，并表示在属性请求时支持的格式的子集。属性处理程序应包含逻辑，以决定引脚在运行时能够支持的格式。例如，硬件实现可能具有DMA约束，这些约束不允许以某些格式组合支持全双工。 **[KSPROPERTY\_PIN\_DATAINTERSECTION](https://docs.microsoft.com/windows-hardware/drivers/stream/ksproperty-pin-dataintersection)** 此属性从数据范围列表中选择一种数据格式。选择基于动态功能，并且格式是从属性请求时驱动程序可以支持的格式子集中获取的。要使用此属性，调用方将提供一个数据范围数组。从第一个元素开始，属性处理程序搜索数组，直到找到当前能够支持的数据范围。如果成功，则处理程序输出从该数据范围获取的数据格式，并返回STATUS\_SUCCESS。否则，处理程序将返回STATUS\_NO\_MATCH。音频系统组件使用KSPROPERTY\_PIN\_DATARANGES和KSPROPERTY\_PIN\_DATAINTERSECTION属性。微型端口驱动程序应支持这些属性。对KSPROPERTY\_PIN\_CONSTRAINEDDATARANGES的支持是可选的。

有关更多信息，请参见[音频数据格式和数据范围](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-data-formats-and-data-ranges)。

**注意**    KSPROPERTY\_PIN\_DATARANGES和KSPROPERTY\_PIN\_CONSTRAINEDDATARANGES均以8字节对齐的地址开头。

**杰克描述属性**

* 2018/05/08
* 8分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/jack-description-property.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/jack-description-property.md" \o "1个贡献者)

在Windows Vista和更高版本中，[**KSPROPERTY\_JACK\_DESCRIPTION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-jack-description)属性描述了音频适配器上的音频插孔或其他物理连接器。该属性值描述插孔的颜色，插孔的物理位置，连接器类型和其他插孔功能。该信息的目的是帮助用户找到正确的插孔，以插入音频端点设备，例如麦克风，耳机或扬声器。有关更多信息，请参见[音频端点设备](https://go.microsoft.com/fwlink/p/?linkid=130876)。

如果音频适配器上的KS过滤器支持KSPROPERTY\_JACK\_DESCRIPTION属性，则Windows多媒体控制面板Mmsys.cpl将显示过滤器上桥接针脚的插孔信息。桥针代表与音频端点设备的连接（通常是插孔）。尽管属性值包含有关某个引脚（或与该引脚关联的一个或多个插孔）的信息，但该属性是过滤器的属性，而不是引脚的属性。有关桥脚的更多信息，请参见[音频滤波器图](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-filter-graphs)。有关过滤器属性和引脚属性的更多信息，请参见过[滤器，引脚和节点属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/filter--pin--and-node-properties)。

音频应用程序可以通过调用DeviceTopology API中的**IKsJackDescription :: GetJackDescription**方法来获取音频终结点设备的KSPROPERTY\_JACK\_DESCRIPTION属性值。例如，应用程序可以使用插孔信息来帮助用户将插入绿色XLR插孔的麦克风与插入橙色XLR插孔的麦克风区分开。有关DeviceTopology API的更多信息，请参见[Device Topologies](https://go.microsoft.com/fwlink/p/?linkid=130878)。

Microsoft HD Audio类驱动程序会根据从HD Audio编解码器中的引脚配置寄存器读取的数据自动构造KSPROPERTY\_JACK\_DESCRIPTION属性值。但是，任何基于KS的音频驱动程序都可以在其过滤器自动化表中实现对此属性的支持。有关HD Audio类驱动程序的更多信息，请参阅[HD Audio和UAA](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/hd-audio-and-uaa)。有关引脚配置寄存器的更多信息，请参见[《高清晰度音频设备引脚配置指南》](https://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/PinConfig.doc)白皮书。

音频终结点设备可以通过一个或多个插孔连接到桥引脚。例如，一组（两声道）立体声扬声器需要一个插孔，而一组5.1环绕声扬声器则需要三个插孔（假设每个插孔处理六个声道中的两个）。

每个插孔的描述都包含在[**KSJACK\_DESCRIPTION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksjack-description)结构中。例如，具有一个插孔的音频端点设备的KSPROPERTY\_JACK\_DESCRIPTION属性值包含一个KSJACK\_DESCRIPTION结构，而具有三个插孔的端点设备的属性值包含三个KSJACK\_DESCRIPTION结构。无论哪种情况，在KSJACK\_DESCRIPTION结构或属性值中的结构之前都有[**KSMULTIPLE\_ITEM**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksmultiple_item)结构，该结构指定属性值的大小。有关更多信息，请参见[**KSPROPERTY\_JACK\_DESCRIPTION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-jack-description)。

插孔信息对于帮助用户区分连接到多声道扬声器配置的插孔特别有用。以下代码示例显示了KSJACK\_DESCRIPTION结构的数组，音频驱动程序使用该结构来描述一组5.1环绕声扬声器的三个插孔：

C ++复制

KSJACK\_DESCRIPTION ar\_5dot1\_Jacks[] =

{

// Jack 1

{

(SPEAKER\_FRONT\_LEFT | SPEAKER\_FRONT\_RIGHT), // ChannelMapping (L,R)

RGB(0,255,0), // Color (green)

eConnType3Point5mm, // ConnectionType

eGeoLocRear, // GeoLocation

eGenLocPrimaryBox, // GenLocation

ePortConnJack, // PortConnection

TRUE // IsConnected

},

// Jack 2

{

(SPEAKER\_FRONT\_CENTER | SPEAKER\_LOW\_FREQUENCY), // (C,Sub)

RGB(0,0,255), // (red)

eConnType3Point5mm,

eGeoLocRear,

eGenLocPrimaryBox,

ePortConnJack,

TRUE

},

// Jack 3

{

(SPEAKER\_SIDE\_LEFT | SPEAKER\_SIDE\_RIGHT), // (SL,SR)

RGB(0,255,255), // (yellow)

eConnType3Point5mm,

eGeoLocRear,

eGenLocPrimaryBox,

ePortConnJack,

TRUE

}

};

如果音频硬件可以检测到设备是否已插入，则驱动程序会动态更新此成员的值，以指示设备当前是插入（**TRUE**）还是拔出（**FALSE**）。

在前面的代码示例中，每个数组元素中的**IsConnected**成员均设置为**TRUE，**以指示端点设备已插入插孔。但是，如果硬件缺少插孔检测功能，则必须始终将**IsConnected**设置为**TRUE**，无论是否有设备插入插孔。要删除从这个双重意义产生歧义**TRUE**返回值，客户端应用程序可以调用[IKsJackDescription2 :: GetJackDescription2](https://go.microsoft.com/fwlink/p/?linkid=143698)读取的JackCapabilities标志[**KSJACK\_DESCRIPTION2**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksjack-description2)结构体。如果此标志设置了JACKDESC2\_PRESENCE\_DETECT\_CAPABILITY位，则表明该端点实际上确实支持插孔存在检测。在那种情况下，**IsConnected**成员的值可以解释为插孔的插入状态的准确反映。

Windows SDK中的头文件Wingdi.h中定义了出现在前述结构中的RGB宏。

另外，可以使用一系列插孔说明来显示两个或多个插孔在功能上彼此等效。在下面的代码示例中，音频驱动程序将黄色RCA插孔和黑色数字光纤插孔的插孔说明组合为一个阵列，以向用户指示两个插孔承载相同的信号：

C ++复制

KSJACK\_DESCRIPTION ar\_SPDIF\_Jacks[] =

{

// Jack 1

{

(SPEAKER\_FRONT\_LEFT | SPEAKER\_FRONT\_RIGHT), // ChannelMapping (L,R)

RGB(0,255,255), // Color (yellow)

eConnTypeRCA, // ConnectionType (RCA)

eGeoLocRear, // GeoLocation

eGenLocPrimaryBox, // GenLocation

ePortConnJack, // PortConnection

TRUE // IsConnected

},

// Jack 2

{

(SPEAKER\_FRONT\_LEFT | SPEAKER\_FRONT\_RIGHT), // (L,R)

RGB(0,0,0), // (black)

eConnTypeOptical, // (optical)

eGeoLocRear,

eGenLocPrimaryBox,

ePortConnJack,

TRUE

}

};

在前面的代码示例中，两个KSJACK\_DESCRIPTION结构中**ChannelMapping**成员的值相同。

WDK中的“简单” MSVAD示例驱动程序（在示例目录Src \ Audio \ Msvad \ Simple中）可以调整为支持KSPROPERTY\_JACK\_DESCRIPTION属性。此示例驱动程序很方便演示该属性的使用，因为它不需要实际的硬件。因此，它可以安装在运行Windows的任何计算机上。（但是，只有Windows Vista和更高版本的操作系统才提供对KSPROPERTY\_JACK\_DESCRIPTION属性的完全支持。）有关此示例的更多信息，请参见[Windows Driver Kit Samples](https://docs.microsoft.com/windows-hardware/drivers/samples/index)。

简单MSVAD示例的拓扑过滤器定义了三个桥接引脚。下表列出了这些引脚。

| **密码** | **描述** |
| --- | --- |
| KSPIN\_TOPO\_SYNTHIN\_SOURCE | MIDI输入插孔 |
| KSPIN\_TOPO\_MIC\_SOURCE | 麦克风输入插孔 |
| KSPIN\_TOPO\_LINEOUT\_DEST | 立体声扬声器输出插孔 |

本主题的其余部分说明了如何修改简单MSVAD示例驱动程序以提供三个桥接引脚的插孔信息。

首先，这些引脚的插孔信息可以指定如下：

C ++复制

// Describe MIDI input jack (pin ID = KSPIN\_TOPO\_SYNTHIN\_SOURCE).

static KSJACK\_DESCRIPTION SynthIn\_Jack[] =

{

{

0, // ChannelMapping

RGB(255,255,0), // Color (cyan)

eConnType3Point5mm, // ConnectionType

eGeoLocRear, // GeoLocation

eGenLocPrimaryBox, // GenLocation

ePortConnJack, // PortConnection

TRUE // IsConnected

}

};

// Describe microphone jack (pin ID = KSPIN\_TOPO\_MIC\_SOURCE).

static KSJACK\_DESCRIPTION MicIn\_Jack[] =

{

{

0,

RGB(0,128,255), // (orange)

eConnType3Point5mm,

eGeoLocFront,

eGenLocPrimaryBox,

ePortConnJack,

TRUE

}

};

// Describe stereo speaker jack (pin ID = KSPIN\_TOPO\_LINEOUT\_DEST).

static KSJACK\_DESCRIPTION LineOut\_Jack[] =

{

{

(SPEAKER\_FRONT\_LEFT | SPEAKER\_FRONT\_RIGHT), // ChannelMapping (L,R)

RGB(0,255,0), // (green)

eConnType3Point5mm,

eGeoLocRear,

eGenLocPrimaryBox,

ePortConnJack,

TRUE

}

};

前面的代码示例将两个捕获引脚的**ChannelMapping**成员设置为0。仅模拟渲染引脚应具有非零的**ChannelMapping**值。

对简单MSVAD示例的主要修改是将以下属性处理程序添加到示例文件Mintopo.cpp中的拓扑微型端口的实现中：

C ++复制

#define ARRAY\_LEN(a) sizeof(a)/sizeof(a[0]);

#define MAXIMUM\_VALID\_PIN\_ID KSPIN\_TOPO\_WAVEIN\_DEST

NTSTATUS

CMiniportTopology::PropertyHandlerJackDescription(

IN PPCPROPERTY\_REQUEST PropertyRequest)

{

PAGED\_CODE();

ASSERT(PropertyRequest);

DPF\_ENTER(("[PropertyHandlerJackDescription]"));

NTSTATUS ntStatus = STATUS\_INVALID\_DEVICE\_REQUEST;

ULONG nPinId = (ULONG)-1;

if (PropertyRequest->InstanceSize >= sizeof(ULONG))

{

nPinId = \*((PULONG)(PropertyRequest->Instance));

if (nPinId > MAXIMUM\_VALID\_PIN\_ID)

{

ntStatus = STATUS\_INVALID\_PARAMETER;

}

else if (PropertyRequest->Verb & KSPROPERTY\_TYPE\_BASICSUPPORT)

{

ntStatus = PropertyHandler\_BasicSupport(

PropertyRequest,

KSPROPERTY\_TYPE\_BASICSUPPORT | KSPROPERTY\_TYPE\_GET,

VT\_ILLEGAL);

}

else

{

PKSJACK\_DESCRIPTION pJack = NULL;

ULONG cJacks = 0;

switch (nPinId)

{

case KSPIN\_TOPO\_SYNTHIN\_SOURCE:

pJack = SynthIn\_Jack;

cJacks = ARRAY\_LEN(SynthIn\_Jack);

break;

case KSPIN\_TOPO\_MIC\_SOURCE:

pJack = MicIn\_Jack;

cJacks = ARRAY\_LEN(MicIn\_Jack);

break;

case KSPIN\_TOPO\_LINEOUT\_DEST:

pJack = LineOut\_Jack;

cJacks = ARRAY\_LEN(LineOut\_Jack);

break;

default:

break;

}

ULONG cbNeeded = sizeof(KSMULTIPLE\_ITEM) +

sizeof(KSJACK\_DESCRIPTION) \* cJacks;

if (PropertyRequest->ValueSize == 0)

{

PropertyRequest->ValueSize = cbNeeded;

ntStatus = STATUS\_BUFFER\_OVERFLOW;

}

else if (PropertyRequest->ValueSize < cbNeeded)

{

ntStatus = STATUS\_BUFFER\_TOO\_SMALL;

}

else if (PropertyRequest->Verb & KSPROPERTY\_TYPE\_GET)

{

PKSMULTIPLE\_ITEM pMI = (PKSMULTIPLE\_ITEM)PropertyRequest->Value;

pMI->Size = cbNeeded;

pMI->Count = cJacks;

// Copy jack description structure into Value buffer.

// RtlCopyMemory correctly handles the case Length=0.

PKSJACK\_DESCRIPTION pDesc = (PKSJACK\_DESCRIPTION)(pMI + 1);

RtlCopyMemory(pDesc, pJack, pMI->Size \* pMI->Count);

ntStatus = STATUS\_SUCCESS;

}

}

}

return ntStatus;

}

NTSTATUS

PropertyHandler\_TopoFilter(IN PPCPROPERTY\_REQUEST PropertyRequest)

{

PAGED\_CODE();

ASSERT(PropertyRequest);

DPF\_ENTER(("[PropertyHandler\_TopoFilter]"));

// PropertyRequest structure is filled by PortCls.

// MajorTarget is a pointer to miniport object for miniports.

//

NTSTATUS ntStatus = STATUS\_INVALID\_DEVICE\_REQUEST;

PCMiniportTopology pMiniport = (PCMiniportTopology)PropertyRequest->MajorTarget;

if (IsEqualGUIDAligned(\*PropertyRequest->PropertyItem->Set, KSPROPSETID\_Jack) &&

(PropertyRequest->PropertyItem->Id == KSPROPERTY\_JACK\_DESCRIPTION))

{

ntStatus = pMiniport->PropertyHandlerJackDescription(PropertyRequest);

}

return ntStatus;

}

前面的代码示例引用了先前定义的三个KSJACK\_DESCRIPTION变量-SynthIn\_Jack，MicIn\_Jack和LineOut\_Jack。如果客户端向过滤器查询有效引脚的插孔说明，但不是桥接引脚（因此没有插孔说明），则查询成功（状态码为STATUS\_SUCCESS），但属性处理程序返回一个空插孔描述由KSMULTIPLE\_ITEM结构组成，仅此而已。如果客户端指定了无效的引脚ID（标识不存在的引脚），则处理程序将返回状态代码STATUS\_INVALID\_PARAMETER。

要支持KSPROPERTY\_JACK\_DESCRIPTION属性，还需要对Simple MSVAD示例进行两次其他修改。这些是：

* 将前面代码示例中的**PropertyHandlerJackDescription**方法的声明添加到头文件Mintopo.h中的CMiniportTopology类定义。
* 为拓扑过滤器实现一个自动化表，并将该表的地址加载到头文件Toptable.h 中的[**PCFILTER\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcfilter_descriptor)结构的**AutomationTable**成员中。该结构名为**MiniportFilterDescriptor**。

要为过滤器实现自动化表，请将以下代码插入头文件Toptable.h中（在**MiniportFilterDescriptor**的定义之前）：

C ++复制

static PCPROPERTY\_ITEM PropertiesTopoFilter[] =

{

{

&KSPROPSETID\_Jack,

KSPROPERTY\_JACK\_DESCRIPTION,

KSPROPERTY\_TYPE\_GET | KSPROPERTY\_TYPE\_BASICSUPPORT,

PropertyHandler\_TopoFilter

}

};

DEFINE\_PCAUTOMATION\_TABLE\_PROP(AutomationTopoFilter, PropertiesTopoFilter);

在前面的代码示例中，[**PCPROPERTY\_ITEM**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcproperty_item)结构的**Handler**成员包含指向在上一步中添加到Mintopo.cpp的属性处理程序的函数指针。若要使属性处理程序可从头文件访问，请在头文件的开头插入PropertyHandler\_TopoFilter 的**外部**函数声明。

有关插孔说明属性的更多信息，请参见[《动态音频子设备](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/jack-descriptions-for-dynamic-audio-subdevices)的插孔说明》。

# 麦克风阵列的几何属性

* 2017/04/20
* 4分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/microphone-array-geometry-property.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/microphone-array-geometry-property.md" \o "1个贡献者)

在Windows Vista和更高版本中，提供了对麦克风阵列的支持。在大多数情况下，笔记本电脑或显示器中嵌入的单个麦克风无法很好地捕获声音。麦克风阵列可以更好地隔离声源并抑制环境噪声和混响。所述[**KSPROPERTY\_AUDIO\_MIC\_ARRAY\_GEOMETRY**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-mic-array-geometry)属性指定的麦克风阵列的几何形状。属性值**[KSAUDIO\_MIC\_ARRAY\_GEOMETRY](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-ksaudio_mic_array_geometry)**描述了阵列类型（线性，平面等），阵列中麦克风的数量以及其他功能。

本主题描述外部USB麦克风阵列如何使用Windows Vista随附的麦克风阵列支持。外部USB麦克风阵列必须提供描述**GET\_MEM**请求的参数，以描述其阵列的几何形状和其他功能。

USB麦克风阵列使用标准格式来提供几何形状信息。Windows Vista USB音频类驱动程序在读取几何信息时必须使用相同的格式。有关标准格式的更多信息，请参见[麦克风阵列几何描述符格式](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/microphone-array-geometry-descriptor-format)。

应用程序可以调用[IPart :: GetSubType](https://go.microsoft.com/fwlink/p/?linkid=143726)来检索有关插孔的信息，以确定插入插孔的设备是否是麦克风阵列。**IPart :: GetSubType**返回表示输入插孔类型的针型GUID。如果插入的设备是麦克风阵列，则返回的GUID等于KSNODETYPE\_MICROPHONE\_ARRAY。该应用程序还可以帮助您确定是否将麦克风阵列插入了错误的插孔。在后一种情况下，返回的引脚类别GUID用于其他设备，或者表示没有设备插入麦克风插孔。有关引脚类别GUID的更多信息，请参见[引脚类别属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/pin-category-property)。

应用程序发现插入正确输入插孔的麦克风阵列后，下一步就是确定阵列的几何形状。存在三种基本几何形状：线性，平面和三维（3-D）。几何信息还提供了详细信息，例如每个麦克风的频率范围和xyz坐标。

以下代码示例显示了KSAUDIO\_MIC\_ARRAY\_GEOMETRY结构，音频驱动程序使用该结构来描述外部USB麦克风阵列：

C ++复制

KSAUDIO\_MIC\_ARRAY\_GEOMETRY mic\_Array =

{

0x100,// usVersion (1.0)

KSMICARRAY\_MICARRAYTYPE\_LINEAR,// usMicArrayType

7854, // wVerticalAngleBegin (45 deg; PI/4 radians x 10000)

-7854, // wVerticalAngleEnd

0, // lHorizontalAngleBegin

0, // lHorizontalAngleEnd

25, // usFrequencyBandLo in Hz

19500, // usFrequencyBandHi in Hz

2, // usNumberOfMicrophones

ar\_mic\_Coordinates // KsMicCoord

};

在前面的代码示例中，ar\_mic\_Coordinates变量是KSAUDIO\_MICROPHONE\_COORDINATES结构的数组，它包含麦克风阵列中麦克风的坐标。

下面的代码示例演示如何使用ar\_mic\_Coordinates数组来描述麦克风阵列中麦克风的几何位置，如前面的代码示例中所述：

C ++复制

KsMicCoord ar\_mic\_Coordinates[] =

{

// Array microphone 1

{

KSMICARRAY\_MICTYPE\_CARDIOID,// usType

100, // wXCoord (mic elements are 200 mm apart)

0,// wYCoord

0, // wZCoord

0,// wVerticalAngle

0,// wHorizontalAngle

},

// Array microphone 2

{

KSMICARRAY\_MICTYPE\_CARDIOID,// usType

-100, // wXCoord

0,// wYCoord

0, // wZCoord

0,// wVerticalAngle

0,// wHorizontalAngle

}

};

在前面的代码示例中，为麦克风阵列中的每个麦克风提供了xyz坐标以及描述其有效工作区域的垂直和水平角度。

要修改Micarray MSVAD示例驱动程序以提供虚拟麦克风阵列的阵列几何信息，您必须执行以下任务。

首先，导航到Src \ Audio \ Msvad \ Micarray并找到Mintopo.cpp文件。编辑Mintopo.cpp中的属性处理程序部分，以使KSAUDIO\_MIC\_ARRAY\_GEOMETRY结构包含有关麦克风阵列的信息。以下代码示例显示了必须修改的特定代码部分：

C ++复制

// Modify this portion of PropertyHandlerMicArrayGeometry

PKSAUDIO\_MIC\_ARRAY\_GEOMETRY pMAG = (PKSAUDIO\_MIC\_ARRAY\_GEOMETRY)PropertyRequest->Value;

// fill in mic array geometry fields

pMAG->usVersion = 0x0100; // Version of Mic array specification (0x0100)

pMAG->usMicArrayType = (USHORT)KSMICARRAY\_MICARRAYTYPE\_LINEAR; // Type of Mic Array

pMAG->wVerticalAngleBegin = -7854; // Work Volume Vertical Angle Begin (-45 degrees)

pMAG->wVerticalAngleEnd = 7854; // Work Volume Vertical Angle End (+45 degrees)

pMAG->wHorizontalAngleBegin = 0; // Work Volume Horizontal Angle Begin

pMAG->wHorizontalAngleEnd = 0; // Work Volume Horizontal Angle End

pMAG->usFrequencyBandLo = 100; // Low end of Freq Range

pMAG->usFrequencyBandHi = 8000; // High end of Freq Range

pMAG->usNumberOfMicrophones = 2; // Count of microphone coordinate structures to follow.

pMAG->KsMicCoord[0].usType = (USHORT)KSMICARRAY\_MICTYPE\_CARDIOID;

pMAG->KsMicCoord[0].wXCoord = -100; // mic elements are 200 mm apart

pMAG->KsMicCoord[0].wYCoord = 0;

pMAG->KsMicCoord[0].wZCoord = 0;

pMAG->KsMicCoord[0].wVerticalAngle = 0;

pMAG->KsMicCoord[0].wHorizontalAngle = 0;

pMAG->KsMicCoord[1].usType = (USHORT)KSMICARRAY\_MICTYPE\_CARDIOID;

pMAG->KsMicCoord[1].wXCoord = 100; // mic elements are 200 mm apart

pMAG->KsMicCoord[1].wYCoord = 0;

pMAG->KsMicCoord[1].wZCoord = 0;

pMAG->KsMicCoord[1].wVerticalAngle = 0;

pMAG->KsMicCoord[1].wHorizontalAngle = 0;

前面的代码示例显示了为线性麦克风阵列提供的信息，该阵列具有两个麦克风元素，每个麦克风元素都是心形的，并且位于距阵列中心100 mm处。

对于第二个修改，编辑Msvad.inf文件，如[MSVAD Micarray的INF修改](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/modified-inf-for-msvad-micarray)所示。

完成文件修改后，请完成以下过程以生成并安装麦克风阵列的示例驱动程序。

1. 启动要在其中工作的WDK构建环境。例如，x86自由构建环境。
2. 导航到Src \ Audio \ Msvad文件夹。
3. 键入**构建**命令，然后按Enter键。
4. 将修改后的Msvad.inf文件复制到生成过程创建的以下文件夹中：

Src \ Audio \ Msvad \ Micarray \ objfre\_wlh\_x86 \ i386

1. 验证步骤4中的文件夹是否包含名为Vadarray.sys的文件。
2. 打开控制面板，然后使用**添加硬件**来手动安装示例驱动程序。
3. 在“控制面板”中打开“ **声音”**小程序，然后单击“ **录制”**选项卡以验证您可以看到刚刚安装的虚拟麦克风阵列。

有关如何开发一个应用程序来发现麦克风阵列的信息，请参阅附录C中[如何建立和使用麦克风阵列为Windows Vista](https://go.microsoft.com/fwlink/p/?linkid=306613)。

**麦克风阵列几何描述符格式**

* 2017/04/20
* 3分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/microphone-array-geometry-descriptor-format.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/microphone-array-geometry-descriptor-format.md" \o "1个贡献者)

USB音频麦克风阵列必须向其连接的系统描述其自身。这意味着描述阵列所需的参数必须嵌入在阵列设备本身中。通过使用**GET\_MEM**请求从设备中检索阵列几何信息。

有关USB音频设备几何形状的信息必须以标准格式提供。因此，旨在与Windows Vista USB音频类驱动程序一起使用的USB麦克风阵列必须提供使用下表定义的信息格式的描述符。

| **偏移量** | **领域** | **尺寸** | **值** | **描述** |
| --- | --- | --- | --- | --- |
| 0 | guidMicArrayID | 16 | 全局唯一标识符（GUID） | 唯一的ID，用于标记内存中麦克风阵列信息的开头（{07FE86C1-8948-4db5-B184-C5162D4AD314}）。 |
| 16 | wDescriptorLength | 2 | 数 | 麦克风阵列信息的长度（以字节为单位），包括GUID和长度字段。 |
| 18岁 | 版本 | 2 | 二进制编码的十进制（BCD） | 麦克风阵列规范的版本号，后跟此描述符。 |
| 20 | wMicArrayType | 2 | 数 | 定义了以下值：  00：线性。  01：平面  02：3维（3D）。  03-FFFF：保留 |
| 22 | wWorkVertAngBeg | 2 | 数 | 工作容积的开始垂直角。 |
| 24 | wWorkVertAngEnd | 2 | 数 | 端部工作容积垂直角。 |
| 26 | wWorkHorAngBeg | 2 | 数 | 工作体积开始时的水平角。 |
| 28 | wWorkHorAngEnd | 2 | 数 | 工作容积的末端为水平角。 |
| 30 | wWorkFreqBandLo | 2 | 数 | 工作频率范围的下限。 |
| 32 | wWorkFreqBandHi | 2 | 数 | 工作频率范围的上限。 |
| 34 | wNumberOfMics | 2 | 数 | 随后的单个麦克风定义的数量。 |
| 36 | wMicrophoneType（0） | 2 | 数 | 一个唯一标识麦克风0类型的数字：  00：全向  01：副心形  02：心形  03：超级心形  04：超心形  05：8异型  0F-FF：供应商定义 |
| 38 | wXCoordinate（0） | 2 | 数 | 麦克风0的x坐标。 |
| 40 | wY坐标（0） | 2 | 数 | 麦克风0的y坐标。 |
| 42 | wZCoordinate（0） | 2 | 数 | 麦克风0的z坐标。 |
| 44 | wMicVertAngle（0） | 2 | 数 | 麦克风0的主响应轴（MRA）垂直角。 |
| 46 | wMicHorAngle（0） | 2 | 数 | 麦克风0的MRA水平角。 |
| ... | ... | ... | ... | 麦克风定义1至n-2。 |
| 34 +（（n-1）*12）* | wMicType（n-1） |  | 数 | 一个唯一标识麦克风n-1类型的数字：  00：全向  01：副心形  02：心形  03：超级心形  04：超心形  05：8异型  0F-FF：供应商定义 |
| 36 +（（n-1）12） | wXCoordinate（n-1） |  | 数 | 麦克风n-1的x坐标。 |
| 38 +（（n-1）*12）* | wY坐标（n-1） | 2 | 数 | 麦克风n-1的y坐标。 |
| 40 +（（n-1）12） | wZCoordinate（n-1） | 2 | 数 | 麦克风n-1的Z坐标。 |
| 42 +（（n-1）*12）* | wMicVertAngle（n-1） | 2 | 数 | 麦克风n-1的MRA垂直角。 |
| 44 +（（n-1）12） | wMicHorAngle（n-1） | 2 | 数 | 麦克风n-1的MRA水平角。 |

有关如何在4元素麦克风阵列的描述符中使用此信息格式的详细示例，请参见《[如何为Windows Vista构建和使用麦克风阵列》](https://go.microsoft.com/fwlink/p/?linkid=306613)白皮书的附录A。

注意

* 当您在麦克风阵列信息中包含版本号时，它允许在实现原始规格后更新描述符。版本号是BCD值。例如，当前版本（1.0）表示为0x0100。
* 偏移量和大小值以字节为单位。
* 所有角度均以1/10000弧度为单位表示。例如，3.1416弧度表示为31416。该值的范围可以从-31416到31416（含）。
* Xyz坐标以毫米表示。取值范围是-32767到32767（含）。
* 有关坐标系角度的方向，轴和正方向的信息，请参阅麦克风阵列白皮书中的附录B。
* 频率值以Hz表示。频率值的范围仅受wWorkFreqBandLo到wWorkFreqBandHi的字段大小限制。

# MSVAD Micarray的改良INF

* 2017/04/20
* 2分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/modified-inf-for-msvad-micarray.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/modified-inf-for-msvad-micarray.md" \o "1个贡献者)

本主题显示如何修改Micarray.inf示例文件，以提供有关如何安装“ [麦克风阵列几何属性”中](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/microphone-array-geometry-property)描述的示例麦克风阵列的设置信息。

导航到Src \ Audio \ Msvad以找到Micarray.inf。使用新名称复制原始文件，然后按如下所示编辑Micarray.inf：

信息复制

// Modified micarray.inf file tailored for a microphone array

[Version]

Signature="$CHICAGO$"

Class=MEDIA

Provider=%MSFT%

ClassGUID={4d36e96c-e325-11ce-bfc1-08002be10318}

DriverVer = 02/22/2007, 6.0.6000.1

CatalogFile=msvad.cat

[SourceDisksNames]

222="MSVAD Driver Disk","",222

[SourceDisksFiles]

vadarray.sys=222

;;This syntax is only recognized on Windows XP and later versions of Windows- it is needed to install 64-bit drivers on

;;Windows Server 2003 with Service Pack 1 (SP1) and later versions of Windows Server.

[Manufacturer]

%MfgName%=MicrosoftDS,NTAMD64,NTIA64

;; For Windows Server 2003 Service Pack 1 (SP1) and later versions of Windows, a 64-bit OS will not install a driver

;; unless the Manufacturer and Models Sections explicitly show it is a driver for that platform

;; But the individual model section decorations (or lack thereof) work as they always have.

;; All of the model sections referred to are undecorated or NT-decorated, hence work on all platforms

[MicrosoftDS]

%MSVAD\_MicArray.DeviceDesc%=MSVAD\_MicArray,\*MSVADMicArray

;; This section enables you to install on x64-based systems

[MicrosoftDS.NTAMD64]

%MSVAD\_MicArray.DeviceDesc%=MSVAD\_MicArray,\*MSVADMicArray

;; This section enables you to install on Itanium-based systems

[MicrosoftDS.NTIA64]

%MSVAD\_MicArray.DeviceDesc%=MSVAD\_MicArray,\*MSVADMicArray

[DestinationDirs]

MSVAD\_MicArray.CopyList=10,system32\drivers

;======================================================

; MSVAD\_MICARRAY

;======================================================

[MSVAD\_MicArray]

AlsoInstall=ks.registration(ks.inf),wdmaudio.registration(wdmaudio.inf)

CopyFiles=MSVAD\_MicArray.CopyList

AddReg=MSVAD\_MicArray.AddReg

[MSVAD\_MicArray.CopyList]

vadarray.sys

[MSVAD\_MicArray.Interfaces]

AddInterface=%KSCATEGORY\_AUDIO%,%KSNAME\_Wave%,MSVAD.I.Wave

AddInterface=%KSCATEGORY\_CAPTURE%,%KSNAME\_Wave%,MSVAD.I.Wave

AddInterface=%KSCATEGORY\_AUDIO%,%KSNAME\_Topology%,MSVAD.I.Topo

[MSVAD\_MicArray.AddReg]

HKR,,AssociatedFilters,,"wdmaud,swmidi,redbook"

HKR,,Driver,,vadarray.sys

HKR,Drivers,SubClasses,,"wave,midi,mixer"

HKR,Drivers\wave\wdmaud.drv,Driver,,wdmaud.drv

HKR,Drivers\midi\wdmaud.drv,Driver,,wdmaud.drv

HKR,Drivers\mixer\wdmaud.drv,Driver,,wdmaud.drv

HKR,Drivers\wave\wdmaud.drv,Description,,%MSVAD\_MicArray.DeviceDesc%

HKR,Drivers\midi\wdmaud.drv,Description,,%MSVAD\_MIDI%

HKR,Drivers\mixer\wdmaud.drv,Description,,%MSVAD\_MicArray.DeviceDesc%

;======================================================

; COMMON

;======================================================

[MSVAD.I.Wave]

AddReg=MSVAD.I.Wave.AddReg

[MSVAD.I.Wave.AddReg]

HKR,,CLSID,,%Proxy.CLSID%

HKR,,FriendlyName,,%MSVAD.Wave.szPname%

[MSVAD.I.Topo]

AddReg=MSVAD.I.Topo.AddReg

[MSVAD.I.Topo.AddReg]

HKR,,CLSID,,%Proxy.CLSID%

HKR,,FriendlyName,,%MSVAD.Topo.szPname%

;======================================================

; MSVAD\_MICARRAY

;======================================================

[MSVAD\_MicArray.NT]

Include=ks.inf,wdmaudio.inf

Needs=KS.Registration, WDMAUDIO.Registration

CopyFiles=MSVAD\_MicArray.CopyList

AddReg=MSVAD\_MicArray.AddReg

[MSVAD\_MicArray.NT.Interfaces]

AddInterface=%KSCATEGORY\_AUDIO%,%KSNAME\_Wave%,MSVAD.I.Wave

AddInterface=%KSCATEGORY\_CAPTURE%,%KSNAME\_Wave%,MSVAD.I.Wave

AddInterface=%KSCATEGORY\_AUDIO%,%KSNAME\_Topology%,MSVAD.I.Topo

[MSVAD\_MicArray.NT.Services]

AddService=msvad\_micarray,0x00000002,msvad\_MicArray\_Service\_Inst

[msvad\_MicArray\_Service\_Inst]

DisplayName=%msvad\_micarray.SvcDesc%

ServiceType=1

StartType=3

ErrorControl=1

ServiceBinary=%10%\system32\drivers\vadArray.sys

;======================================================

; COMMON

;======================================================

[Strings]

MSFT="Microsoft"

MfgName="Microsoft Audio DDK"

MSVAD\_MicArray.DeviceDesc="Sample Virtual Audio Device (Mic Array) (WDM)"

MSVAD.Wave.szPname="MSVAD Wave"

MSVAD.Topo.szPname="MSVAD Topology"

MSVAD\_MIDI="MSVAD -> WDM Midi Device"

Proxy.CLSID="{17CCA71B-ECD7-11D0-B908-00A0C9223196}"

KSCATEGORY\_AUDIO="{6994AD04-93EF-11D0-A3CC-00A0C9223196}"

KSCATEGORY\_RENDER="{65E8773E-8F56-11D0-A3B9-00A0C9223196}"

KSCATEGORY\_CAPTURE="{65E8773D-8F56-11D0-A3B9-00A0C9223196}"

KSNAME\_Wave="Wave"

KSNAME\_Topology="Topology"

msvad\_micarray.SvcDesc="Sample Virtual Audio Device (Mic Array) (WDM)"

MediaCategories="SYSTEM\CurrentControlSet\Control\MediaCategories"

Simple.NameGuid="{946A7B1A-EBBC-422a-A81F-F07C8D40D3B4}"

Simple.Name="MSVAD (Simple)"

如上例所示，修改文件后，将其保存在原始位置。有关如何构建麦克风阵列样本驱动程序的更多信息，请参见[麦克风阵列几何属性](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/microphone-array-geometry-property)。

# 硬件活动

* 2017/04/20
* 2分钟阅读
  + [[https://github.com/DOMARS.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/hardware-events.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/hardware-events.md" \o "1个贡献者)

某些音频设备提供硬件音量控制旋钮，静音开关或其他类型的手动控件。应用程序可以通过调整音量或更改音频流的播放方式来响应这些控件中的更改。当用户调整硬件控件时，微型端口驱动程序将使用[IPortEvents](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iportevents)接口通知端口驱动程序发生了硬件事件。端口驱动程序进而将事件通知应用程序，以便它可以从设备读取新的控制设置。

您的微型端口驱动程序可以在从端口驱动程序为**Init**调用提供服务时**为其**查询**IPortEvents**接口的端口驱动程序（例如，请参阅[**IMiniportWavePci :: Init**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavepci-init)）。在Microsoft Windows 98 SE，Windows Me和Windows 2000及更高版本上，该查询成功。有关代码示例，请参见Windows驱动程序工具包（WDK）中的Sb16示例音频适配器。

当端口驱动程序调用驱动程序的[**IMiniport :: GetDescription**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiport-getdescription)方法时，该方法将输出[**PCFILTER\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcfilter_descriptor)结构，该结构除其他外指定设备支持的事件。可以在自动化表中为PCFILTER\_DESCRIPTOR 的**Pins**和**Nodes**成员指定事件，也可以在**AutomationTable**成员中指定事件，后者指向过滤器本身的自动化表。每个事件由[**PCEVENT\_ITEM**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcevent_item)结构指定。您的驱动程序应将PCEVENT\_ITEM结构的**Set**和**Id**成员**设置**为[KSEVENTSETID\_AudioControlChange](https://docs.microsoft.com/windows-hardware/drivers/audio/kseventsetid-audiocontrolchange)和[**KSEVENT\_CONTROL\_CHANGE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksevent-control-change)，并将指向驱动程序的[**EventHandler**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nc-portcls-pcpfnevent_handler)例程的指针加载到**Handler**成员中。您的驱动程序还应将**Flags**成员中的PCEVENT\_ITEM\_FLAG\_BASICSUPPORT位置1，以指示对控制更改事件的基本支持，并且应将PCEVENT\_ITEM\_FLAG\_ONESHOT和/或PCEVENT\_ITEM\_FLAG\_ENABLE位置1，以表明它支持单次和/或重复通知。

当应用程序稍后调用**mixerOpen**函数（在Microsoft Windows SDK文档中描述）以请求特定事件的通知时，端口驱动程序然后使用指向[**PCEVENT\_REQUEST**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-_pcevent_request)结构的指针来调用驱动程序的**EventHandler**例程。该结构的**Verb**成员设置为PCEVENT\_VERB\_ADD，其**EventItem**成员指定要启用的事件。该PCEVENT\_REQUEST结构还包含一个指向[**KSEVENT\_ENTRY**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-_ksevent_entry)结构，你的驱动程序应该当作不透明的系统数据。启用事件后，您的处理程序应调用[**IPortEvents :: AddEventToEventList**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportevents-addeventtoeventlist)使用相同的KSEVENT\_ENTRY指针。通过此调用，处理程序确认事件已启用。

当发生硬件事件并且驱动程序的中断服务例程检测到静音或音量变化时，驱动程序将通过调用[**IPortEvents :: GenerateEventList**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportevents-generateeventlist)及其一组描述事件的参数，将事件通知端口驱动程序。例如，以下调用描述了lineout-volume节点中的控件更改：

C ++复制

pPE->GenerateEventList(NULL, KSEVENT\_CONTROL\_CHANGE,

FALSE, ULONG(-1), TRUE, LINEOUT\_VOL);

在此呼叫期间，端口驱动程序在其事件列表中搜索与呼叫参数匹配的所有事件，并将通知发送到监视这些事件的客户端。在此示例中，pPE是指向**IPortEvents**对象的指针，而LINEOUT\_VOL是微型端口驱动程序分配给lineout-volume节点的节点ID。未指定的参数（例如前面示例中的事件集GUID和引脚ID）被视为通配符，并且始终与列表中的相应参数匹配。