**音频驱动程序的蓝牙绕过指南**

* 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/bluetooth-bypass-guidelines-for-audio-drivers.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/bluetooth-bypass-guidelines-for-audio-drivers.md" \o "1个贡献者)

本节中介绍的蓝牙旁路设计指南向音频驱动程序开发人员展示了如何将音频数据重新路由通过蓝牙主机控制器接口（HCI），以便在片上系统（SoC）解决方案中进行处理。

Windows 8.1引入了对蓝牙旁路音频数据流的支持。

本节包含以下主题。

[建筑概述](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/btth-architectural-overview)

[操作理论](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/theory-of-operation)

[蓝牙旁路DDI参考](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/bluetooth-bypass-ddi-reference)

[相关设计准则](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/related-design-guidelines)

# Windows蓝牙主机控制器接口（HCI）体系结构概述

10/22/2018

2分钟阅读

本主题介绍Windows 8.1支持的体系结构概述，该支持用于重新路由音频数据以绕过Bluetooth主机控制器接口（HCI）。

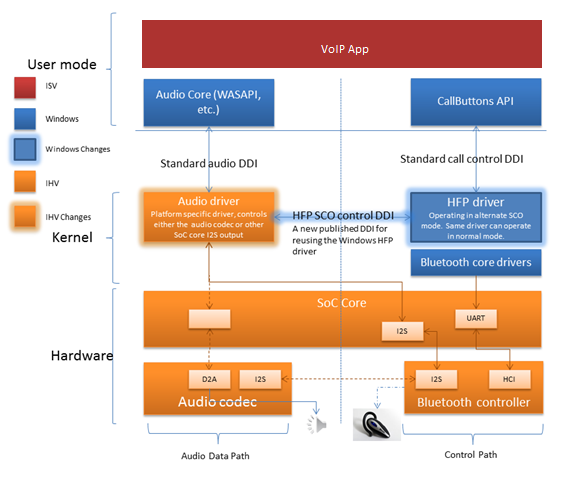
从Windows 8.1开始，Microsoft操作系统已更新，以与低功耗片上系统（SoC）设计解决方案兼容。新的Windows支持与基于Intel或基于ARM的SoC设计兼容。这些新的低功耗设备将针对“始终在线”场景进行优化，因此低电池消耗将是成功的关键因素。

SoC体系结构使用通用异步接收器/发送器（UART）传输模式与蓝牙主机控制器之间来回传输数据。

由于UART无法提供对时间敏感的数据传输，因此除了UART外，还必须实现面向同步连接（SCO）的旁路通道，以通过I2S或音频编解码器与蓝牙无线电之间的某些其他连接来传输音频数据。这意味着必须重新路由音频数据以绕过Bluetooth HCI。蓝牙HCI，通常用于PC上以传输音频数据。

重要的是要注意，此功能只是卸载了Windows 8.1之前版本的Windows中存在的相同功能，因此从用户角度看，SoC上的蓝牙免提配置文件（HFP）之间没有用例有所不同Windows和PC或笔记本电脑上的Windows中的Bluetooth HFP。

下图显示了可以协同工作以在Windows 8.1中提供此新支持的软件和硬件组件。



请注意，此Windows功能不支持使用高级音频分发配置文件（A2DP）的旁路音频流。Windows 8提供了一个单独的A2DP配置文件驱动程序，该驱动程序通过标准的蓝牙HCI完全支持音频功能，而无需任何其他音频驱动程序。

# 操作理论

* 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/theory-of-operation.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/theory-of-operation.md" \o "1个贡献者)

此操作理论主题解释了新Windows 8.1支持蓝牙绕过音频流的内部工作背后的理论。

当蓝牙音频处于旁路模式时，音频控制路径会通过其他一些硬件连接流向蓝牙控制器，而不是通过主机控制器接口（HCI）。该其他硬件连接通常是I2S，但可以是蓝牙主机控制器确定的任何接口。这套主题将这种连接称为“旁路”或“边带”连接。

尽管音频I / O通过旁路连接发生，但仍通过HCI管理面向空中同步连接（SCO）的音频流。Windows 8提供了蓝牙免提配置文件（HFP）驱动程序，以隐藏管理SCO连接和免提配置文件其他方面的大多数复杂性。但是，自定义音频驱动程序控制Windows和旁路连接之间的音频数据I / O。

HFP驱动程序和用于音频I / O数据的自定义控制驱动程序之间的角色分离，意味着必须在Windows HFP驱动程序和自定义音频驱动程序之间建立有效的通信。此通信由从自定义音频驱动程序传递到Windows HFP驱动程序的一组IOCTL处理。

通常，旁路连接本身总是存在的。PnP服务始终枚举包括此连接的硬件，然后加载所需的音频驱动程序。但是，音频系统可能没有配对的HFP耳机，并且旁路连接仅在配对了至少一个HFP耳机时才有用。

对于每个配对的HFP设备，Windows HFP驱动程序在GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS接口类中注册并启用设备接口。因此，对于HFP设备，以下条件成立：

* Windows激活HFP驱动程序时（通常是在启动过程中），HFP驱动程序将注册并启用每个配对的HFP设备的接口。
* 当HFP设备首次配对且Windows已启动并正在运行时，HFP驱动程序将注册并启用该设备的接口。
* 如果有n对配对的HFP设备，则Windows HFP驱动程序将注册n个设备接口实例。
* 删除配对的HFP设备后，Windows HFP驱动程序将禁用设备接口。
* 当Windows停止HFP驱动程序时（通常是在关机或重新引导过程中），HFP驱动程序将为每个配对的HFP设备禁用接口。
* 音频驱动程序必须随时处理多个接口的到达和删除，而不仅仅是在启动或关闭期间。

以下主题提供有关连接生命周期以及HFP设备及其音频驱动程序的某些设计功能的更多信息。

[HFP设备启动](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/startup)

[HFP设备连接](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/hfp-device-connection)

[HFP设备拆卸](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/removal)

[内核流注意事项](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-streaming-considerations)

[音频端点容器ID](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-endpoint-container-id)

[I2S和SCO资源的管理](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/management-of-i2s-and-sco-resources)

# HFP设备启动

* 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/startup.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/startup.md" \o "2位贡献者)

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

HFP设备启动主题讨论了蓝牙免提配置文件（HFP）设备到达音频系统时发生的情况。

对于到达音频系统中的每个配对的HFP设备，Windows HFP驱动程序都会在GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS类中注册一个设备接口。音频驱动程序使用设备接口通知来始终了解GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS接口的所有实例。音频驱动程序从其AVStrMiniDevicePostStart驱动程序例程中（或从等效的Portcls例程）调用IoRegisterPlugPlayNotification，以注册回调以发现当前安装的HFP设备，并通知新的HFP设备。

当音频驱动程序调用IoRegisterPlugPlayNotification时，将使用以下参数进行调用。

* EventCategory设置为EventCategoryDeviceInterfaceChange。
* 通常将EventCategoryFlags设置为PNPNOTIFY\_DEVICE\_INTERFACE\_INCLUDE\_EXISTING\_INTERFACES，以便立即接收现有接口的通知。但是，某些替代音频驱动程序设计可能会通过其他方式找到现有接口。
* EventCategoryData设置为GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS。
* DriverObject设置为音频驱动程序的DriverObject。
* CallbackRoutine在音频驱动程序中设置为将接收通知的例程。

以下各节概述了音频驱动程序可以为配对的HFP设备的每个注册实例执行的任务。

**处理接口实例**

对于在GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS类中注册的每个接口实例，音频驱动程序必须使用以下协议进行通信：

当Windows调用在音频驱动程序称为IoRegisterPlugPlayNotification时注册的音频驱动程序的回调例程时，Windows使用DEVICE\_INTERFACE\_CHANGE\_NOTIFICATION传递HFP接口的符号链接。*SymbolicLinkName*。

当音频驱动程序调用IoGetDeviceObjectPointer时，驱动程序将使用符号链接获取HFP设备的HFP FileObject和DeviceObject。

当音频驱动程序将IOCTL发送到HFP驱动程序时，该驱动程序将HFP FileObject和DeviceObject用于HFP设备。

**检索静态信息**

音频驱动程序可以从HFP驱动程序检索静态信息。例如，HFP驱动程序可以提供配对的HFP设备的ksnodetype，容器ID和友好名称。音频驱动程序可以使用此信息来创建和初始化KS过滤器或代表配对HFP设备的过滤器。音频驱动程序使用**[IOCTL\_BTHHFP\_DEVICE\_GET\_DESCRIPTOR](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_device_get_descriptor)**来获取此信息。

音频驱动程序还可以检索配对的HFP设备的蓝牙地址。每个配对的HFP设备都有一个唯一的蓝牙地址，这可用作唯一的标识符字符串。有关更多信息，请参阅[获取HF设备的蓝牙地址](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/obtaining-bluetooth-address-of-hf-device)。

**创建，初始化音频特定的过滤器工厂上下文**

若要创建和初始化特定于音频的过滤器工厂上下文，音频驱动程序必须将HFP DeviceObject和HFP FileObject存储在过滤器工厂上下文中，然后将IsConnected字段初始化为false。

**创建KS过滤器工厂**

对于GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS接口类中的每个设备实例，音频驱动程序将创建并启用一个或多个过滤器工厂。

如果音频驱动程序是AVStream驱动程序，则音频驱动程序将调用KsCreateFilterFactory添加新的筛选器工厂，并调用KsFilterFactorySetDeviceClassesState启用工厂。如果音频驱动程序是PortCls驱动程序，则它将通过调用PcRegisterSubdevice间接创建并启用KS过滤器工厂。对于许多PortCls音频驱动程序设计，都有两个为给定的配对HFP设备注册的子设备。

每个过滤器工厂（或对于PortCls音频驱动程序，每对过滤器工厂）都代表单个配对的HFP设备的音频功能。音频驱动程序为每个配对的HFP设备创建单独的过滤器工厂，这些工厂由GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS接口的唯一实例表示。对于每个配对的HFP设备，音频驱动程序必须对*KsCreateFilterFactory的RefString*参数或PcRegisterSubdevice的*Name*参数使用唯一的字符串。驱动程序开发人员可能会发现将配对的HFP设备的Bluetooth地址字符串用作唯一字符串很有用。有关如何检索唯一字符串的信息[，](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/obtaining-bluetooth-address-of-hf-device)请参见[获取HF设备的蓝牙地址](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/obtaining-bluetooth-address-of-hf-device)。

请注意，没有特定的最大数量的可能配对的HFP设备，因此音频驱动程序应避免硬编码的特定限制。而是，音频驱动程序必须正确处理多个GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS接口的动态到达和移除。

但是，实际上，PortCls驱动程序在调用PcAddAdapterDevice时必须指定最大子设备数。PcAddAdapterDevice为每个潜在的子设备预分配额外的内存。音频驱动程序开发人员应选择足以容纳许多配对设备的数字，但同时选择一个不会浪费资源的数字。例如，仅支持2个HFP设备可能不够用，而支持2000个肯定会导致资源过度使用。但是，支持16可能是合理的。

如果在运行时通知音频驱动程序另一个GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS接口，但是已经注册了其最大子设备数量，则音频驱动程序可以调用某种算法来选择配对的HFP设备，其子设备可以取消注册以为其腾出空间。新的HFP设备。例如，音频驱动程序可以跟踪具有最旧连接的HFP设备。而一个更简单但可能不那么用户友好的音频驱动程序在达到最大值后，可以简单地忽略其他GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS接口。

**发送获取连接状态IOCTL**

音频驱动程序发送获取连接状态IOCTL，以获取有关连接中发生的任何更改的信息。

**发送获取音量状态IOCTL**

音频驱动程序发送获取音量状态IOCTL，以获取有关头戴式耳机的音量状态中发生的任何音量级别变化的信息。

**相关话题**

[**IOCTL\_BTHHFP\_DEVICE\_GET\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_device_get_descriptor)  
[操作](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/theory-of-operation)[**原理**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_device_get_descriptor)  
[获取HF设备的蓝牙地址](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/obtaining-bluetooth-address-of-hf-device)

**HFP设备连接**

* 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/hfp-device-connection.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/hfp-device-connection.md" \o "1个贡献者)

HFP设备连接主题讨论了音频系统如何确定和处理蓝牙免提配置文件（HFP）设备的连接状态信息。

根据所有音频驱动程序的要求，音频驱动程序必须支持**[KSPROPERTY\_JACK\_DESCRIPTION](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-jack-description)**。音频驱动程序在过滤器工厂上下文中维护*IsConnected*字段。音频驱动程序在处理**KSPROPERTY\_JACK\_DESCRIPTION**属性时使用此值。

当**[IOCTL\_BTHHFP\_DEVICE\_GET\_CONNECTION\_STATUS\_UPDATE](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_device_get_connection_status_update)**成功完成时，音频驱动程序将使用新的连接状态更新*IsConnected*。如果状态已更改，则音频驱动程序将引发**[KSEVENT\_PINCAPS\_JACKINFOCHANGE](https://docs.microsoft.com/windows-hardware/drivers/audio/ksevent-pincaps-jackinfochange)**事件，这将导致音频系统重新评估连接状态。音频驱动程序然后调用**IOCTL\_BTHHFP\_DEVICE\_GET\_CONNECTION\_STATUS\_UPDATE的**另一个实例以接收下一个状态更改。如果有一个较早的状态更改请求仍在等待处理，则第二个呼叫将失败，音频驱动程序不会更新其连接状态，也不会发出其他状态更改信息请求。

如[内核流注意事项](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-streaming-considerations)中所述，音频驱动程序必须支持[**KSPROPERTY\_ONESHOT\_RECONNECT**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-oneshot-reconnect)和**[KSPROPERTY\_ONESHOT\_DISCONNECT](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-oneshot-disconnect)**，并且这些属性的处理程序必须分别向HFP驱动程序发送REQUESTCONNECT和REQUESTDISCONNECT IOCTL。这些IOCTL很快完成，并且音频驱动程序需要准备好响应返回的结果。

这是音频驱动程序开发人员必须注意的其他一些与蓝牙音频设备连接有关的因素。

**流通道**

流通道表示音频驱动程序的无线带宽分配。在大多数情况下，这是SCO渠道。但是，管理SCO通道状态的某些细节完全在HFP驱动程序中处理。例如，这包括远程断开连接，这可能是由于呼叫情形而导致，其中HF发起了到AG的音频传输（在这种情况下，PC扮演AG的角色）。

**音频滤波器引脚状态**

音频驱动程序为两个KS引脚实现KS引脚状态处理程序（类似于AVStrMiniPinSetDeviceState）。这些引脚中的任何一个都需要SCO流通道，以通过空中传输数据。当这些引脚中的任何一个转换为KSSTATE\_ACQUIRE时，音频驱动程序都会通过将[**IOCTL\_BTHHFP\_STREAM\_OPEN**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_stream_open)发送给HFP驱动程序来打开通道。这是一个异步调用，可能需要几秒钟才能完成。音频驱动程序不需要实现自己的超时机制，因此应等待IOCTL完成，然后再完成向KSSTATE\_ACQUIRE的转换。

当两个KS引脚都转换为KSSTATE\_STOP时，音频驱动程序将**[IOCTL\_BTHHFP\_STREAM\_CLOSE](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_stream_close)**发送到HFP驱动程序。这很快完成。

要确定何时发送**IOCTL\_BTHHFP\_STREAM\_OPEN**和**IOCTL\_BTHHFP\_STREAM\_CLOSE**，音频驱动程序可以使用简单的参考计数机制来跟踪需要SCO流通道的引脚数。当参考计数从0变为1时，音频驱动程序将打开和关闭SCO流通道。

在**IOCTL\_BTHHFP\_STREAM\_OPEN上**，HFP驱动程序请求一个SCO通道（如果尚未打开），并使用SCO请求的结果完成该请求。如果已打开，则在**IOCTL\_BTHHFP\_STREAM\_CLOSE上**，HFP驱动程序请求断开SCO通道。

**远程SCO连接和断开连接**

在远程SCO断开连接上，如果流通道已关闭，则HFP驱动程序不执行任何操作。如果打开了流通道，则HFP驱动程序将启动重新连接计时器。当计时器到期时，如果SCO仍断开连接并且Stream Channel仍然打开，则驱动程序将请求SCO通道。请注意，当SCO断开连接时，没有音频数据通过空中传输，因此在此期间音频中会出现间隙。如果SCO请求失败，则HFP驱动程序将通过完成对**[IOCTL\_BTHHFP\_STREAM\_GET\_STATUS\_UPDATE的](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_stream_get_status_update)**任何调用，向音频驱动程序[**发送流通**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_stream_get_status_update)道状态更改信号。这应该很少见，因为远程SCO断开通常与请求将呼叫音频传输到音频网关的HF设备关联。音频驱动程序应将此视为中间错误情况。

此过程使VoIP应用程序有时间从CallButtons API接收音频传输回调，并在HFP端点上干净地释放其音频资源，而不会引起流错误。

在远程SCO连接上，如果流通道已打开，则驱动程序仅接受连接。如果关闭了流通道，则HFP驱动程序将接受连接并启动断开连接计时器。当断开连接计时器到期时，如果SCO仍然连接并且Stream通道仍然关闭，则驱动程序将中断SCO连接。

此过程使VoIP应用程序有时间从CallButtons API接收音频传输回调并在HFP端点上建立音频资源，而不会过早拒绝或关闭SCO连接。

**相关话题**

[操作理论](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/theory-of-operation)

**HFP设备拆卸**

* 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/removal.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/removal.md" \o "1个贡献者)

“ HFP设备删除”主题讨论了从（离开）音频系统中删除Bluetooth免提配置文件（HFP）设备时发生的情况。

要删除已配对的HFP设备的注册设备接口，音频驱动程序：

1. 取消所有未决的IOCTL\_BTHHFP\_SPEAKER\_GET\_VOLUME\_STATUS\_UPDATE IOCTL。
2. 取消所有未决的IOCTL\_BTHHFP\_STREAM\_GET\_STATUS\_UPDATE IOCTL。
3. 取消所有未决的IOCTL\_BTHHFP\_DEVICE\_GET\_CONNECTION\_STATUS\_UPDATE IOCTL。
4. 取消引用HFP FileObject（也取消引用DeviceObject）。
5. 调用KsDeleteFilterFactory删除代表与已删除接口关联的HFP设备的过滤器工厂。

**相关话题**

[操作理论](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/theory-of-operation)

# 内核流注意事项

* 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/kernel-streaming-considerations.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/kernel-streaming-considerations.md" \o "1个贡献者)

内核流注意事项主题阐明了所有音频驱动程序的要求和其他特殊注意事项。这些是内核流媒体注意事项，与蓝牙旁路音频流媒体更相关。

音频驱动程序应完全支持WaveRT端口驱动程序，包括“拉模式”。有关更多信息，请参见[WaveRT端口驱动程序简介](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/introducing-the-wavert-port-driver)。而且，尽管不需要为面向同步连接（SCO）的旁路输出实现硬件音频引擎，但这样做没有害处。

Windows徽标对格式支持的要求包括蓝牙例外。

音频驱动程序应支持通过边带硬件可能的格式。这通常是8KHz单声道音频流。

## 拓扑结构

所有蓝牙免提设备均支持捕获和渲染。因此，音频驱动程序应公开用于免提设备的内核流（KS）拓扑，如下图所示，以支持渲染和捕获。

**注意**   音频驱动程序开发人员可以选择是为捕获路径还是渲染路径实现单个过滤器，还是选择单独的过滤器。但是，HFP设备仅在GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS设备接口上允许单个文件对象。因此，使用两个过滤器的设计需要允许两个过滤器共享单个文件对象。

DAC和ADC节点代表模拟/数字转换，但不支持任何KS属性。

卷节点通过将SETVOLUME和GETVOLUMESTATUSUPDATE IOCTL发送到HFP驱动程序来支持[**KSPROPERTY\_AUDIO\_VOLUMELEVEL**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-volumelevel)和**[KSEVENT\_CONTROL\_CHANGE](https://docs.microsoft.com/windows-hardware/drivers/audio/ksevent-control-change)**。

卷节点应实现如下：

* 如果蓝牙耳机支持音量控制，则音频驱动程序应在其KS拓扑中包括一个音量节点。音频驱动程序的音量属性处理程序将上述IOCLT发送到Bluetooth HFP驱动程序以处理音量。
* 如果蓝牙耳机未实现硬件音量，并且编解码器（或DSP）具有硬件音量，则音频驱动程序应处理编解码器（或DSP）上的音量控制。
* 如果蓝牙耳机和音频设备没有硬件音量控制，则不应显示任何音量节点，Windows将插入一个软件音量控制节点。

静音节点是可选的。且仅当DSP或音频编解码器具有将旁路PCM信号传递到Bluetooth控制器之前使之静音的功能时，音频驱动程序才应实现静音节点。静音节点支持[**KSPROPERTY\_AUDIO\_MUTE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-mute)。

## 财产要求

音频驱动程序使用以下KS属性来获取有关音频路径中任何一个或多个音频插孔的信息。音频驱动程序还可以使用适当的属性请求来建立或断开与音频路径中任何Bluetooth音频设备的连接。

**KSPROPERTY\_JACK\_DESCRIPTION**

此属性返回[**KSJACK\_DESCRIPTION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksjack-description)结构。音频驱动程序应如下设置**[KSPROPERTY\_JACK\_DESCRIPTION](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-jack-description)**字段。ChannelMapping = KSAUDIO\_SPEAKER\_MONO颜色= 0 ConnectionType = eConnTypeOtherDigital GeoLocation = eGeoLocNotApplicable GenLocation = eGenLocOther PortConnection = ePortConnUnknown IsConnected = < BOOL用于当前连接状态 > **KSPROPERTY\_JACK\_DESCRIPTION2**

此属性返回[**KSJACK\_DESCRIPTION2**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksjack-description2)结构。音频驱动程序应如下设置**[KSPROPERTY\_JACK\_DESCRIPTION2](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-jack-description2)**字段。DeviceStateInfo = 0 JackCapabilities = JACKDESC2\_PRESENCE\_DETECT\_CAPABILITY **KSPROPERTY\_ONESHOT\_RECONNECT**

音频驱动程序的过滤器应支持[**KSPROPERTY\_ONESHOT\_RECONNECT**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-oneshot-reconnect)。为了创建和初始化此结构，音频驱动程序将[**IOCTL\_BTHHFP\_DEVICE\_REQUEST\_CONNECT**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_device_request_connect)发送到HFP驱动程序。HFP驱动程序完成此请求，然后尝试异步连接到Bluetooth音频设备。 **KSPROPERTY\_ONESHOT\_DISCONNECT**

音频驱动程序的过滤器应支持[**KSPROPERTY\_ONESHOT\_DISCONNECT**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-oneshot-disconnect)。为了创建和初始化此结构，音频驱动程序将[**IOCTL\_BTHHFP\_DEVICE\_REQUEST\_DISCONNECT**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_device_request_disconnect)发送到HFP驱动程序。HFP驱动程序完成此请求，然后尝试异步从Bluetooth音频设备断开连接。当音频驱动程序支持这些属性时，“控制面板”中的“声音”对话框将显示HFP端点的“连接”和“断开连接”命令。

## 相关话题

[操作理论](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/theory-of-operation)

**音频端点容器ID**

* 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-endpoint-container-id.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/audio-endpoint-container-id.md" \o "1个贡献者)

音频端点容器ID主题讨论了可用于获取与Bluetooth音频设备关联的音频端点的容器ID的可靠方法。

音频端点构建器使用枚举算法来确定音频端点的容器ID，然后将这些ID作为属性存储在MMDEVAPI端点属性存储中。在某些情况下，端点构建器使用的逻辑不足以处理蓝牙I2S设计，其中音频驱动程序公开的音频端点的容器ID由另一个枚举器-蓝牙枚举器确定。

这种涉及使用自己的蓝牙枚举器的蓝牙I2S设计的方案很少见。但是无论如何，您都可以开发音频驱动程序来为这种情况提供支持。在这种情况下，您的音频驱动程序可以为端点支持新的容器ID属性。新属性是**[KSPROPERTY\_JACK\_CONTAINERID，](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-jack-containerid)**并且[**已将**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-jack-containerid)其添加到现有的[KSPROPSETID\_Jack](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-jack)属性集中。该值为GUID，它是容器ID的数据类型。

音频驱动程序支持**KSPROPERTY\_JACK\_CONTAINERID**，当且仅当它可以通过其他方式可靠地获取正确的容器ID时，才可以。例如，从蓝牙枚举器。

如果您的音频驱动程序支持**KSPROPERTY\_JACK\_CONTAINERID**属性，则音频系统会从驱动程序中读取该属性的值，然后将该值存储为音频端点的容器ID。

有关容器ID和上一节中提到的算法的更多信息，请参见[容器ID](https://docs.microsoft.com/windows-hardware/drivers/install/container-ids)和[音频端点构建器算法](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-endpoint-builder-algorithm)。

**相关话题**

[操作理论](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/theory-of-operation)

**I2S和SCO资源的管理**

* 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/management-of-i2s-and-sco-resources.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/management-of-i2s-and-sco-resources.md" \o "1个贡献者)

“ I2S和SCO资源的管理”主题讨论了在Windows 8.1中对蓝牙旁路音频流的这种新支持的设计中所做的假设。

此时Windows假定只有一个蓝牙主机控制器。而且，面向HFP同步连接（SCO）的旁路支持假定只有一个旁路连接，并且通过HFP设备驱动程序接口打开的任何通道都与该单个连接相关联。

音频驱动程序应以先到先得的原则为单个使用者仲裁此通道和单个旁路连接。实现此目的的最简单方法是，驱动器仅允许单个滤波器将其引脚转换为ACQUIRE状态。

**相关话题**

[操作理论](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/theory-of-operation)

**蓝牙旁路DDI参考**

* 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/bluetooth-bypass-ddi-reference.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/bluetooth-bypass-ddi-reference.md" \o "1个贡献者)

蓝牙旁路设备驱动程序接口（DDI）参考是一组主题，这些主题显示Windows 8.1引入的结构和IOCTL，以提供对蓝牙免提配置文件（HFP）驱动程序的支持。

有关DDI成员的详细信息，请参阅[Bluetooth HFP DDI参考](https://docs.microsoft.com/windows-hardware/drivers/audio/bluetooth-hfp-ddi-reference)。

**相关话题**

[操作理论](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/theory-of-operation)

**相关设计准则**

* 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/related-design-guidelines.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/related-design-guidelines.md" \o "1个贡献者)

“相关设计指南”部分中的主题显示了端口类音频驱动程序和蓝牙免提设备的音频驱动程序如何设置设备相关的值并注册音频处理对象。

本节包含以下主题。

[设置属性和注册表值](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/setting-properties-and-registry-values)

[设置友好名称，注册APO](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/setting-friendly-name--registering-apos)

[获取HF设备的蓝牙地址](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/obtaining-bluetooth-address-of-hf-device)

**设置属性和注册表值**

* 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/setting-properties-and-registry-values.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/setting-properties-and-registry-values.md" \o "2位贡献者)

* + [[https://github.com/EliotSeattle.png?size=32](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/setting-properties-and-registry-values.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/setting-properties-and-registry-values.md" \o "2位贡献者)

设置属性和注册表值主题描述了端口类音频驱动程序如何为PnP设备接口设置属性和注册表值。

端口类音频驱动程序Portcls必须执行以下步骤才能正确注册设备接口并设置所需的值。

**1.注册设备接口**

在为子设备调用PcRegisterSubdevice之前，驱动程序可以直接调用IoRegisterDeviceInterface来注册KSCATEGORY\_AUDIO接口。这使驱动程序有机会在PcRegisterSubdevice注册并启用接口之前在设备接口上设置接口属性和注册表值。

音频驱动程序如下设置IoRegisterDeviceInterface的参数。

* PhysicalDeviceObject参数是音频驱动程序可以从PcGetPhysicalDeviceObject函数检索的PDEVICE\_OBJECT。
* InterfaceClassGuid设置为接口的类GUID。
* ReferenceString与音频驱动程序传递给PcRegisterSubdevice的Name参数相同。

成功完成上述任务后，IoRegisterDeviceInterface返回已注册接口的SymbolicLinkName。

**2.设置注册表值**

音频驱动程序调用IoOpenDeviceInterfaceRegistryKey以获得设备接口注册表项的句柄。音频驱动程序将参数设置为IoOpenDeviceInterfaceRegistryKey，如下所示。

SymbolicLinkName是在上一步中从IoRegisterDeviceInterface返回的字符串。

DesiredAccess设置为KEY\_WRITE（或驱动程序需要的其他值）。

成功完成上述步骤后，DeviceInterfaceKey返回打开的注册表项句柄。音频驱动程序：

* 调用ZwSetValueKey设置注册表值
* 通过调用ZwClose关闭注册表项句柄

**注意**   如果驱动程序需要在注册表子项中设置值，则驱动程序调用ZwCreateKey来创建子项。准备调用ZwCreateKey时，驱动程序：

* 调用InitializeObjectAttributes，并将ObjectName设置为子项路径
* 将属性设置为OBJ\_CASE\_INSENSITIVE | OBJ\_KERNEL\_HANDLE
* 将RootDirectory设置为IoOpenDeviceInterfaceRegistryKey返回的句柄
* 调用ZwClose以关闭通过调用ZwCreateKey创建的任何句柄

**3.设置属性**

音频驱动程序调用IoSetDeviceInterfacePropertyData设置属性。音频驱动程序将参数设置为IoSetDeviceInterfacePropertyData，如下所示：

* SymbolicLinkName是从IoRegisterDeviceInterface返回的字符串。
* 其余参数取决于所设置的特定属性。

**相关话题**

[相关设计准则](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/related-design-guidelines)

# 设置友好名称，注册APO

* 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/setting-friendly-name--registering-apos.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/setting-friendly-name--registering-apos.md" \o "1个贡献者)

设置友好名称，注册APO主题描述端口类Bluetooth边带音频驱动程序如何设置设备接口的友好名称，以及如何注册蓝牙设备使用的任何音频处理对象（APO）。

对于每个启用的GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS接口，端口类音频驱动程序（PortCls）通常调用PcRegisterSubdevice函数，该函数注册代表音频适配器上子设备的PnP设备接口。在典型的音频驱动程序设计中，音频驱动程序为“ wave”和“ topology”子设备调用PcRegisterSubdevice，然后通过调用其他端口类功能进行连接。

在为“拓扑”子设备调用PcRegisterSubdevice之前，驱动程序将按照“ [设置属性和注册表值”中](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/setting-properties-and-registry-values)描述的过程在KSCATEGORY\_AUDIO接口类的接口上设置属性和注册表值。以下各节介绍了特定的属性和注册表值。

## DEVPKEY\_DeviceInterface\_FriendlyName

音频驱动程序将[**IOCTL\_BTHHFP\_DEVICE\_GET\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ni-bthhfpddi-ioctl_bthhfp_device_get_descriptor)请求发送到免提配置文件（HFP）音频驱动程序。所请求的信息以**[BTHHFP\_DESCRIPTOR](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/bthhfpddi/ns-bthhfpddi-_bthhfp_descriptor)**结构以及该结构引用的其他数据的形式返回。音频驱动程序然后调用IoSetDeviceInterfacePropertyData，以将DEVPKEY\_DeviceInterface\_FriendlyName设置为**BTHHFP\_DESCRIPTOR**结构的FriendlyName字段中的**值**。

音频驱动程序将参数设置为IoSetDeviceInterfacePropertyData，如下所示：

* SymbolicLinkName =从IoRegisterDeviceInterface返回的字符串
* PropertyKey = DEVPKEY\_DeviceInterface\_FriendlyName
* Lcid = LOCALE\_NEUTRAL
* 标志= PLUGPLAY\_PROPERTY\_PERSISTENT
* 类型= DEVPROP\_TYPE\_STRING\_INDIRECT
* 大小= BTHHFP\_DESCRIPTOR.FriendlyName.Length + sizeof（UNICODE\_NULL）
* 数据= BTHHFP\_DESCRIPTOR.FriendlyName.Buffer

## APO注册

如[设置属性和注册表值](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/setting-properties-and-registry-values)中所述，驱动程序可以为设备接口设置注册表值。要注册APO，音频驱动程序会在设备接口上设置几个值。这些值与通常为INF中的APO注册设置的值相同，并且特定值将从一个音频驱动程序更改为另一个音频驱动程序。

这是用于注册APO的INF文件语法的示例：

句法复制

HKR,"EPFX\\0",%PKEY\_FX\_Association%,,%KSNODETYPE\_ANY%

HKR,"EPFX\\0",%PKEY\_FX\_EndpointEffectClsid%,,%FX\_DISCOVER\_EFFECTS\_APO\_CLSID%

**注意**   上段中显示的语法不包括注册APO的COM服务器的说明。

## 相关话题

[相关设计准则](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/related-design-guidelines)

# 获取HF设备的蓝牙地址

* 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/obtaining-bluetooth-address-of-hf-device.md)](https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/audio/obtaining-bluetooth-address-of-hf-device.md" \o "1个贡献者)

获取HF设备的蓝牙地址主题演示了音频驱动程序如何获取配对的免提（HF）设备的蓝牙地址。

地址字符串对于唯一标识特定配对的HF设备很有用。例如，该地址串可以被用作ReferenceString传递给IoRegisterDeviceInterface，参数RefString传递给KsCreateFilterFactory参数或名称传递给PcRegisterSubdevice参数。

请注意，GUID\_DEVINTERFACE\_BLUETOOTH\_HFP\_SCO\_HCIBYPASS设备接口符号链接对于每个配对的HF设备也是唯一的，但是出于某些目的，此字符串可能太长。

以下代码示例中显示的IoHelperGetDevicePdo例程是IoHelperGetDeviceBluetoothAddress使用的实用程序函数。当音频驱动程序正在处理PnP接口到达通知时，可以调用此类功能。音频驱动程序从IoGetDeviceObjectPointer获取设备对象，并将其传递给IoHelperGetDeviceBluetoothAddress。

ManagedCPlusPlus复制

\_IRQL\_requires\_(PASSIVE\_LEVEL)

NTSTATUS

IoHelperGetDevicePdo(

\_In\_ PDEVICE\_OBJECT DeviceObject,

\_Out\_ PDEVICE\_OBJECT \*PhysicalDeviceObject

)

/\*++

Routine description:

Returns a pointer to the physical device object in a driver stack and

increments the reference count on that object.

Parameters:

DeviceObject

Pointer to the device object for which the physical device object is

retrieved.

PhysicalDeviceObject

Returns a pointer to the physical device object in a stack of device

objects after incrementing the reference count on the object.

Return value:

A status code.

Remarks:

This routine uses IRP\_MN\_QUERY\_DEVICE\_RELATIONS TargetDeviceRelation to

get the physical device object from the device stack.

Requirements:

IRQL = PASSIVE\_LEVEL

--\*/

{

NTSTATUS status;

PIRP irp = NULL;

PDEVICE\_RELATIONS deviceRelations = NULL;

PIO\_STACK\_LOCATION ioStack;

KEVENT completionEvent;

PAGED\_CODE();

irp = IoAllocateIrp(DeviceObject->StackSize, FALSE);

if (irp == NULL)

{

status = STATUS\_INSUFFICIENT\_RESOURCES;

goto Exit;

}

irp->IoStatus.Status = STATUS\_NOT\_SUPPORTED;

ioStack = IoGetNextIrpStackLocation(irp);

ioStack->MajorFunction = IRP\_MJ\_PNP;

ioStack->MinorFunction = IRP\_MN\_QUERY\_DEVICE\_RELATIONS;

ioStack->Parameters.QueryDeviceRelations.Type = TargetDeviceRelation;

KeInitializeEvent(&completionEvent, SynchronizationEvent, FALSE);

IoSetCompletionRoutine(irp, OnRequestComplete, &completionEvent, TRUE, TRUE, TRUE);

status = IoCallDriver(DeviceObject, irp);

if (status == STATUS\_PENDING) {

// wait for irp to complete

KeWaitForSingleObject(

&completionEvent,

Suspended,

KernelMode,

FALSE,

NULL);

status = irp->IoStatus.Status;

}

if (NT\_SUCCESS(status))

{

deviceRelations = (PDEVICE\_RELATIONS)irp->IoStatus.Information;

\*PhysicalDeviceObject = deviceRelations->Objects[0];

}

Exit:

if (deviceRelations != NULL)

{

ExFreePool(deviceRelations);

}

if (irp != NULL)

{

IoFreeIrp(irp);

}

return status;

}

\_IRQL\_requires\_(PASSIVE\_LEVEL)

NTSTATUS IoHelperGetDeviceBluetoothAddress(

\_In\_ PDEVICE\_OBJECT DeviceObject,

\_Outptr\_ PWSTR \*BluetoothAddress,

\_In\_ ULONG Tag

)

/\*++

Routine description:

Returns the Bluetooth address string for the physical device object in the

device stack.

Parameters:

DeviceObject

Pointer to a device object in a device stack whose physical object has

a Bluetooth address property.

BluetoothAddress

Returns a string allocated from NonPagedPoolNx pool.

Tag

Return value:

A status code.

Remarks:

This routine retrieves the DEVPKEY\_Bluetooth\_DeviceAddress property from

the physical device object in the device stack containing the specified

device object.

Requirements:

IRQL = PASSIVE\_LEVEL

--\*/

{

NTSTATUS status;

PDEVICE\_OBJECT physicalDeviceObject = NULL;

PWSTR bluetoothAddress = NULL;

ULONG requiredSize;

DEVPROPTYPE devPropType;

PAGED\_CODE();

status = IoHelperGetDevicePdo(DeviceObject, &physicalDeviceObject);

if (!NT\_SUCCESS(status))

{

goto Exit;

}

status = IoGetDevicePropertyData(physicalDeviceObject, &DEVPKEY\_Bluetooth\_DeviceAddress, LOCALE\_NEUTRAL, 0, 0, NULL, &requiredSize, &devPropType);

if (NT\_SUCCESS(status) || devPropType != DEVPROP\_TYPE\_STRING)

{

status = STATUS\_UNSUCCESSFUL;

goto Exit;

}

if (status != STATUS\_BUFFER\_TOO\_SMALL)

{

goto Exit;

}

bluetoothAddress = (PWCH)ExAllocatePoolWithTag(NonPagedPoolNx, requiredSize, Tag);

if (bluetoothAddress == NULL)

{

status = STATUS\_INSUFFICIENT\_RESOURCES;

goto Exit;

}

status = IoGetDevicePropertyData(physicalDeviceObject, &DEVPKEY\_Bluetooth\_DeviceAddress, LOCALE\_NEUTRAL, 0, requiredSize, bluetoothAddress, &requiredSize, &devPropType);

if (!NT\_SUCCESS(status))

{

goto Exit;

}

\*BluetoothAddress = bluetoothAddress;

bluetoothAddress = NULL;

status = STATUS\_SUCCESS;

Exit:

if (bluetoothAddress != NULL)

{

ExFreePoolWithTag(bluetoothAddress, Tag);

}

if (physicalDeviceObject != NULL)

{

ObDereferenceObject(physicalDeviceObject);

}

return status;

}

## 相关话题

[相关设计准则](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/related-design-guidelines)