WDM音频驱动程序概览

[Windows驱动程序模型](https://docs.microsoft.com/windows-hardware/drivers/kernel/windows-driver-model)（WDM）音频驱动程序使用[内核流](https://docs.microsoft.com/windows-hardware/drivers/stream/kernel-streaming)（KS）组件，这些组件以内核模式运行，并且是操作系统的一部分。

在开始开发基于Windows的音频硬件设备之前，硬件供应商应做出一些设计决策。

**第一个决策**是**是否设计需要供应商提供的自定义驱动程序的音频设备**。对于符合Microsoft [通用音频体系结构](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/universal-audio-architecture)（UAA）准则的PCI，USB和IEEE 1394设备, Windows提供了操作系统支持。供应商无需为兼容UAA的音频设备提供自定义驱动程序。

但是，如果需要供应商提供自定义音频驱动程序，则供应商必须**选择是将驱动程序设计为与哪一类类系统驱动程序结合使用**, PortCls系统驱动程序（Portcls.sys）还是AVStream（Ks.sys）。PortCl和AVStream都是Windows操作系统的一部分。PortCls是大多数音频适配器的正确选择。有关PortCls的更多信息，请参见[Port Class简介](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/introduction-to-port-class)。有关AVStream的更多信息，请参见[AVStream概述](https://docs.microsoft.com/windows-hardware/drivers/stream/avstream-overview)。

设计使用PortCls的自定义适配器驱动程序时，使用WaveRT的应用程序可以使用音频适配器上的设备。有关更多信息，请参见[WaveRT端口驱动程序介绍](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/introducing-the-wavert-port-driver)。

**另外两个决定**涉及**如何向音频应用程序显示适配器拓扑和引脚数据范围**。拓扑是适配器电路中数据路径和控制节点的逻辑映射。数据范围指定设备在其wave和MIDI流中可以支持的数据格式。这两个决定都会影响音频适配器上的设备对应用程序的显示方式。

在做出所有前面提到的决定时，硬件供应商必须权衡性能增强的价值与实施它们的成本。另一个考虑因素是，是否可以针对Windows家族中的许多产品制定特定的解决方案。本节概述了这些问题，并提供了有关特定主题的更详细文档的参考。

本节包括以下主题：

# 1. 通用Windows音频驱动程序

在Windows 10中，您可以编写可在多种类型的硬件上运行的通用音频驱动程序。本主题讨论此方法的优点以及不同平台之间的差异。除了用于音频的通用Windows驱动程序之外，Windows继续支持以前的音频驱动程序技术，例如WDM。

## 1.1. 通用Windows音频驱动程序入门

IHV可以开发一种适用于所有设备（台式机，笔记本电脑，平板电脑，电话）的通用Windows驱动程序。这样可以减少开发时间和初始开发以及以后代码维护的成本。

这些工具可用于开发通用Windows驱动程序支持：

* Visual Studio 2015支持：有一个驱动程序设置，可将“目标平台”设置为等于“通用”。有关设置驱动程序开发环境的更多信息，请参见《[通用Windows驱动程序入门》](https://docs.microsoft.com/windows-hardware/drivers)。
* APIValidator工具：可以使用ApiValidator.exe工具来验证驱动程序调用的API对Universal Windows驱动程序可用的。该工具是Windows 10的Windows驱动程序工具包（WDK）的一部分，如果您使用的是Visual Studio 2015，它将自动运行。有关更多信息，请参见[验证通用Windows驱动程序](https://docs.microsoft.com/windows-hardware/drivers)。
* 更新的DDI参考文档：DDI参考文档正在更新，以指示Universal Windows驱动程序支持哪些DDI。有关更多信息，请参见《[音频设备参考》](https://docs.microsoft.com/previous-versions/ff536192(v=vs.85))。

## 1.2. 创建通用音频驱动程序

有关逐步指导，请参阅《[通用Windows驱动程序入门》](https://docs.microsoft.com/windows-hardware/drivers)。以下是步骤的摘要：

1. 加载通用音频sysvad示例，以用作通用音频驱动程序的起点。或者，从空的WDM驱动程序模板开始，并根据音频驱动程序的需要从通用sysvad示例中添加代码。
2. 在项目属性中，将“目标平台”设置为“通用”。
3. 创建安装软件包：如果目标是运行Windows 10桌面版（家庭版，专业版，企业版和教育版）的设备，请使用可配置的INF文件。如果目标是运行Windows 10移动版的设备，请使用[PkgGen](https://docs.microsoft.com/en-us/windows-hardware/manufacture/iot/create-packages" \l "run-the-pkggenexe-tool)生成.spkg文件。
4. （为台式机版本或Windows 10移动版的Windows 10）编译、安装、部署和调试驱动程序。

## 1.3. 样例代码

Sysvad和SwapAPO已转换为通用Windows驱动程序示例。有关更多信息，请参见[示例音频驱动程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/sample-audio-drivers)。

## 1.4. 通用Windows音频驱动程序的可用编程接口

从Windows 10开始，驱动程序编程接口是Windows基于[OneCoreUAP](https://docs.microsoft.com/en-us/windows/win32/apiindex/umbrella-lib-onecoreuap)的版本的一部分。通过使用该通用集，您可以编写通用Windows驱动程序。这些驱动程序将同时在台式机版Windows 10和Windows 10移动版以及其他Windows 10版本上运行。

使用通用音频驱动程序时，可以使用以下DDI。

* [音频驱动程序事件集](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-event-sets)([Audio Drivers Event Sets](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-event-sets))
* [音频驱动程序接口](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-interfaces)([Audio Drivers Interfaces](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-interfaces))
* [音频驱动程序属性集](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-property-sets)([Audio Drivers Property Sets](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-property-sets))
* [音频驱动器结构](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-structures)([Audio Drivers Structures](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-drivers-structures))
* [音频拓扑节点](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-topology-nodes)([Audio Topology Nodes](https://docs.microsoft.com/windows-hardware/drivers/audio/audio-topology-nodes))
* [高清晰度音频DDI参考](https://docs.microsoft.com/windows-hardware/drivers/audio/high-definition-audio-ddi-reference)([High Definition Audio DDI Reference](https://docs.microsoft.com/windows-hardware/drivers/audio/high-definition-audio-ddi-reference))
* [端口类音频驱动程序参考](https://docs.microsoft.com/windows-hardware/drivers/audio/port-class-audio-driver-reference)([Port Class Audio Driver Reference](https://docs.microsoft.com/windows-hardware/drivers/audio/port-class-audio-driver-reference))

## 1.5. 将现有的音频驱动程序转换为通用Windows驱动程序

请按照此过程将现有的音频驱动程序转换为通用Windows驱动程序。

1. 确定您现有的驱动程序调用是否将在OneCoreUAP Windows上运行。检查参考页的要求部分。有关更多信息，请参见《[音频设备参考》](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/portal-audio-ref)。
2. 将驱动程序重新编译为通用Windows驱动程序。在项目属性中，将“目标平台”设置为“通用”。
3. 使用ApiValidator.exe工具来验证驱动程序调用的DDI对于Universal Windows驱动程序有效。该工具是Windows 10的Windows驱动程序工具包（WDK）的一部分，如果您使用的是Visual Studio 2015，它将自动运行。有关更多信息，请参见[验证通用Windows驱动程序](https://docs.microsoft.com/windows-hardware/drivers)。
4. 如果驱动程序调用不属于OneCoreUAP的接口，则编译器将显示错误。
5. 将这些调用替换为其他调用，或者创建代码替代方法，或者编写新的驱动程序。

## 1.6. 创建组件化的音频驱动程序安装

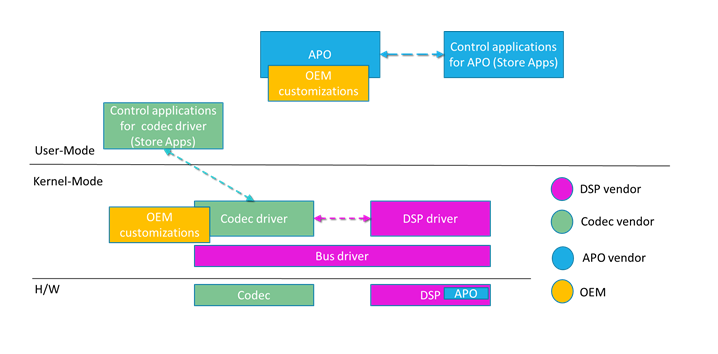
### 1.6.3. 概观

为了创建更流畅，更可靠的安装体验并更好地支持组件服务，请将驱动程序安装过程分为以下组件。

* DSP（如果有）和编解码器(Codec)
* APO
* OEM定制

或者，可以将单独的INF文件用于DSP和编解码器。

此图总结了组件化的音频安装。



一个独立的扩展INF文件用于自定义特定系统的每个基本驱动程序组件。定制包括调整参数和其他特定于系统的设置。有关更多信息，请参见 [使用扩展INF文件](https://docs.microsoft.com/windows-hardware/drivers/install/using-an-extension-inf-file)。

扩展INF文件必须是通用INF文件。有关更多信息，请参见[使用Universal INF文件](https://docs.microsoft.com/windows-hardware/drivers/install/using-a-universal-inf-file)。

有关使用INF文件添加软件的信息，请参阅《[使用组件INF文件》](https://docs.microsoft.com/windows-hardware/drivers/install/using-a-component-inf-file)。

### 1.6.2. 提交组件化的INF文件

APO INF软件包必须与基本驱动程序软件包分开提交给合作伙伴中心。有关创建软件包的更多信息，请参见《[Windows HLK入门》](https://docs.microsoft.com/windows-hardware/test/hlk/getstarted/windows-hlk-getting-started)。

### 16.3. SYSVAD组成的INF文件

要查看组成INF文件的示例，请检查Github上的[sysvad / TabletAudioSample](https://github.com/Microsoft/Windows-driver-samples/tree/master/audio/sysvad/TabletAudioSample)。

| **文档名称** | **描述** |
| --- | --- |
| ComponentizedAudioSample.inf | 基础组件的音频INF例子文件。 |
| ComponentizedAudioSampleExtension.inf | sysvad基础组件的扩展驱动程序，带有其他OEM定制。 |
| ComponentizedApoSample.inf | APO扩展INF例子文件。 |

传统INF文件在SYSVAD示例中仍然可用。

| **文档名称** | **描述** |
| --- | --- |
| tabletaudiosample.inf | 桌面系统上的INF文件，其中包含安装驱动程序所需的所有信息。 |
| phoneaudiosample.inf | 一个电话专用INF文件，其中包含安装驱动程序所需的所有信息。 |

### 1.6.4. APO供应商相关的Tuning参数和功能配置

必须通过扩展INF软件包安装所有APO供应商系统相关的设置，参数和Tuning值。在许多情况下，可以使用[INF AddReg指令](https://docs.microsoft.com/windows-hardware/drivers/install/inf-addreg-directive)以简单的方式执行此操作。在更复杂的情况下，可以使用Tuning文件。

基本驱动程序包必须不依赖于这些自定义, 应可以独立工作（尽管功能可能会减少）。

### 1.6.6. UWP音频设置应用

要实现最终用户UI，请对Windows通用音频驱动程序使用硬件支持应用程序（HSA）。有关更多信息，请参见[硬件支持应用（HSA）：驱动程序开发人员的步骤](https://docs.microsoft.com/windows-hardware/drivers/devapps/hardware-support-app--hsa--steps-for-driver-developers)。

### 1.6.6. 以编程方式启动UWP硬件支持应用

要基于驱动程序事件（例如，连接新的音频设备时）以编程方式启动UWP硬件支持应用程序，请使用Windows Shell API。Windows 10 Shell API支持一种启动UWP UI的方法, 可以基于资源激活或直接通过[IApplicationActivationManager](https://docs.microsoft.com/windows/desktop/api/shobjidl_core/nf-shobjidl_core-iapplicationactivationmanager-activateapplication)。有关UWP应用程序自动启动的更多详细信息, 请参考[自动启动Windows 10 UWP应用程序](https://docs.microsoft.com/windows/uwp/xbox-apps/automate-launching-uwp-apps" \l "launch-activation)。

### 1.6.7. APO和设备驱动程序供应商对AudioModules API的使用

音频模块API / DDI旨在标准化, “UWP应用程序或用户模式服务”和”内核驱动程序模块或DSP处理模块”之间的, 命令的通信传输（而非协议）。音频模块要求驱动程序实现正确的DDI以支持模块枚举和通信。命令以二进制形式传递，解释/定义留给创建者。

音频模块当前不支持(is not currently designed to facilitate)UWP应用和(音频引擎中运行的)软件APO之间的直接通信。

有关音频模块的更多信息，请参阅[实现音频模块通信](https://docs.microsoft.com/windows-hardware/drivers/audio/implementing-audio-module-communication)以及[配置和查询音频设备模块](https://docs.microsoft.com/windows-hardware/drivers/audio/configure-and-query-audiodevicemodules)。

### 1.6.8. APO HWID字符串构造

APO硬件ID包含标准信息和供应商定义的字符串。

它们的构造如下：

SWC\VEN\_v(4)&AID\_a(4)&SUBSYS\_ n(4)s(4) &REV\_r(4)

SWC\VEN\_v(4)&AID\_a(4)&SUBSYS\_ n(4)s(4)

SWC\VEN\_v(4)&AID\_a(4)

其中：

* v（4）是APO设备供应商的4个字符的标识符。这将由Microsoft管理。
* a（4）是APO供应商定义的APO的4个字符的标识符。
* n（4）是父设备的子系统供应商的4个字符的PCI SIG分配的标识符。这通常是OEM标识符。
* s（4）是父设备的4个字符的供应商定义的子系统标识符。这通常是OEM产品标识符。

### 1.6.9. 即插即用INF评估版本信息和日期，以更新驱动程序

在存在多个驱动程序时, Windows即插即用系统评估驱动程序的日期和版本信息，以确定要安装哪个驱动器。有关更多信息，请参见[Windows如何对驱动程序进行排名(Rank)](https://docs.microsoft.com/windows-hardware/drivers/install/how-setup-ranks-drivers--windows-vista-and-later-) 。

要允许使用最新的驱动程序，请确保并为每个新版本的驱动程序更新日期和版本信息。

### 1.6.10. APO驱动程序注册表项

对于第三方定义的音频驱动程序/ APO注册表项，请使用HKR（HKLM \ System \ CurrentControlSet除外）。

### 1.6.11. 使用Windows服务促进UWP <-> APO通信

Windows Service并非严格要求用于管理用户模式组件（如APO），但是，如果您的设计包括RPC服务器以促进UWP <-> APO通信，我们建议在Windows Service中实现该功能，然后控制APO的运行在音频引擎中。

## 1.7. 编译生成(Build)Windows 10桌面Sysvad通用音频示例

完成以下步骤，即可为Windows 10桌面生成sysvad示例。

1. 找到桌面inf文件（tabletaudiosample.inf），并将制造商名称设置为诸如“ Contoso”的值
2. 在解决方案资源管理器(Visual Studio 的 Solution Explorer)中，右键单击“解决方案(Solution)'sysvad”，然后选择“配置管理器(Configuration Manager)”。如果要部署到64位版本的Windows，请将目标平台设置为x64。确保所有项目的配置和平台设置都相同。
3. 生成sysvad解决方案中的所有项目(project)。
4. 找到编译(Build)的输出目录。例如，它可以位于以下目录中：

C:\Program Files(x86)\Windows Kits\10\src\audio\sysvad\x64\Debug\package

1. 导航到WDK安装中的“工具”文件夹，然后找到PnpUtil工具。例如，查看以下文件夹：

C:\Program Files(x86)\Windows Kits\10\Tools\x64\PnpUtil.exe

1. 将以下文件复制到要安装sysvad驱动程序的系统：

|  |  |
| --- | --- |
| TabletAudioSample.sys | 驱动程序文件。 |
| tabletaudiosample.inf | 信息（INF）文件，其中包含安装驱动程序所需的信息。 |
| sysvad.cat | 目录文件。 |
| SwapAPO.dll | 用于管理APO的UI的示例驱动程序扩展。 |
| PropPageExt.dll | 属性页的示例驱动程序扩展。 |
| KeywordDetectorAdapter.dll | 示例关键字检测器。 |

## 1.8. 安装并测试驱动程序

请按照以下步骤在目标系统上使用PnpUtil安装驱动程序。

1. 打开“管理员”命令提示符(cmd.exe)，然后在驱动程序文件所在目录中键入以下内容。

**pnputil -i -a tabletaudiosample.inf**

1. sysvad驱动程序安装应该完成。如果有任何错误，您可以检查此文件以获取其他信息：%windir%\inf\setupapi.dev.log
2. 在设备管理器的“视图”菜单上，选择“设备类型”。在设备树中，找到“ Microsoft虚拟音频设备（WDM）-Sysvad示例”。通常在“声音，视频和游戏控制器”节点下。
3. 在目标计算机上，打开“控制面板”，然后导航至“ **硬件和声音”** >“ **管理音频设备”**。在“声音”对话框中，选择标有“ Microsoft虚拟音频设备（WDM）-Sysvad Sample”的扬声器图标，然后单击“设置默认值”，但不要单击“确定”。这将使“声音”对话框保持打开状态。
4. 在目标计算机上找到MP3或其他音频文件，然后双击播放。然后在“声音”对话框中，验证与Microsoft虚拟音频设备（WDM）-Sysvad Sample驱动程序关联的音量指示器中是否有活动。

## 1.9. 为Windows 10 Mobile编译Sysvad通用音频示例

完成以下步骤，为Windows 10移动版生成sysvad示例。

1. 找到移动版inf文件（phoneaudiosample.inf），并将制造商名称设置为诸如“ Contoso”的值
2. 在sysvad解决方案(Solution)中编译(Build)以下项目：

* EndPointsCommon
* PhoneAudioSample

1. 从编译(Build)的输出目录。例如，Visual Studio的默认位置可能位于这样的目录中：

C:\Program Files (x86)\Windows Kits\10\src\audio\sysvad\x64\Debug\package`

1. 按照[创建软件包中](https://docs.microsoft.com/previous-versions/windows/hardware/packaging/dn756642(v=vs.85))的指导创建一个包含移动版驱动程序镜像(image)文件的软件包。
2. 要安装移动驱动程序软件包（.spkg文件），您需要将软件包合并到移动OS映像中。使用ImgGen将.spkg驱动程序包添加到完整的闪存更新（FFU）映像，然后可以将该闪存更新到移动设备。可能有必要删除移动版镜像(image)中存在的其他音频驱动程序，以测试sysvad虚拟音频驱动程序。
3. 在OS映像中包含正在运行的驱动程序包之后，播放声音剪辑并验证sysvad电话音频示例是否正常运行。您可以建立内核调试器连接来监视移动设备上的sysvad虚拟驱动程序。

# 2. 通用音频架构

Microsoft通用音频体系结构（UAA）使符合该体系结构的音频设备完全依赖操作系统来提供驱动程序支持。

为了与UAA兼容，PCI音频适配器必须符合Intel High Definition Audio规范，该规范是Intel Audio Codec '97（AC'97）规范的后继产品。与AC'97不同，除了编码解码器设备和串行接口链接（将控制器连接到编码解码器的链接）之外，英特尔高清晰度音频（英特尔高清音频）还使总线控制器标准化。(英特尔高清音频设备的)UAA设计准则包含不属于英特尔高清晰度音频规范的其他要求。Windows Vista为兼容UAA的PCI音频适配器提供了完整的驱动程序支持。

为了与UAA兼容，USB音频设备必须同时符合USB音频规范和USB音频设备的UAA设计准则。USB音频设备可以使用Windows附带的USBAudio [类系统驱动程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-mode-wdm-audio-components#usbaudio_class_system_driver)（Usbaudio.sys）。根据定义，与Windows中的USBAudio类系统驱动程序兼容的USB音频设备符合UAA。

为了与UAA兼容，IEEE 1394 AV / C音频设备必须同时符合相关的IEEE 1394规范和(针对IEEE 1394 AV / C音频设备的)UAA设计准则。IEEE 1394音频设备可以使用Windows内置的的[AVCAudio类系统驱动程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-mode-wdm-audio-components#avcaudio_class_system_driver)（Avcaudio.sys）。

有关Microsoft UAA计划的更多信息，请参见[通用音频体系结构](https://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/UAA_Guidelines.doc)白皮书。有关Intel HD Audio的更多信息，请访问[Intel HD Audio](https://go.microsoft.com/fwlink/p/?linkid=42508)网站。有关USB和IEEE 1394音频设备的相关规范的列表，请参见[USBAudio类系统驱动](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-mode-wdm-audio-components#avcaudio_class_system_driver)[程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-mode-wdm-audio-components#usbaudio_class_system_driver)和[AVCAudio类系统驱动程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-mode-wdm-audio-components#avcaudio_class_system_driver)。

# 3. 音频信号处理模式

驱动程序应声明每个设备支持的音频信号处理模式。

## 3.1. 可用的信号处理模式

音频类别（由应用程序选择）映射到音频模式（由驱动程序定义）。Windows定义了七个音频信号处理模式。OEM和IHV可以确定要实施的模式。建议IHV / OEM使用新模式来添加音频效果，以优化音频信号以提供最佳的用户体验。下表中汇总了这些模式。

|  |  |  |
| --- | --- | --- |
| 模式 | 渲染/捕获 | 描述 |
| 原始(Raw) | 都 | 原始模式下不应对流应用任何信号处理。  应用程序可以请求完全未经处理的原始流，并执行自己的信号处理。 |
| 默认(Default) | 都 | 此模式定义默认的音频处理 |
| 电影(Movies)\* | 渲染 | 电影音频播放 |
| 媒体(Media) \* | 都 | 音乐音频播放（大多数媒体流的默认设置） |
| 语音 (Speech)\* | 捕获 | 人声抓取(例如: 输入到Cortana) |
| 通信(Communication)\* | 都 | VOIP渲染和捕获（例如: Skype，Lync） |
| 通知(Notification)\* | 渲染 | 铃声，警报，警报等 |

\* Windows 10的新功能。

## 3.2. 信号处理模式驱动程序要求

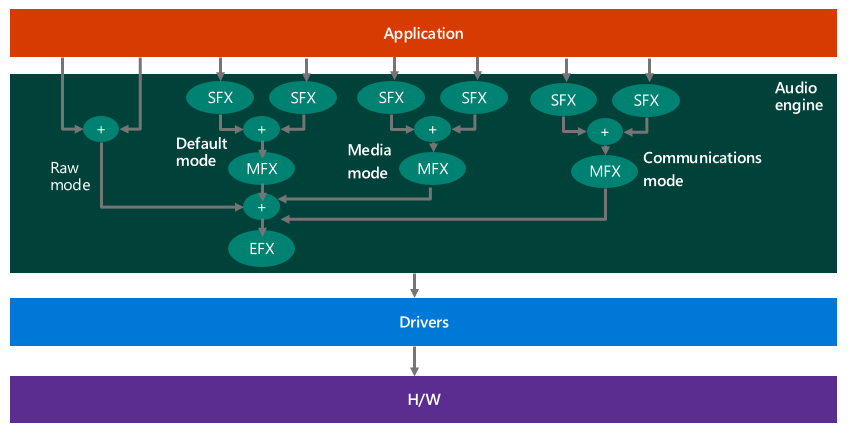
音频设备驱动程序需要至少支持Raw或Default模式。支持其他模式是可选的。

特定的系统可以不必支持所有的模式。驱动程序定义它们支持的信号处理模式（即，作为驱动程序的一部分安装了哪种类型的APO）并相应地通知OS。如果驱动程序不支持特定模式，则Windows将使用下一个最佳匹配的模式。

下图显示了一个支持多种模式的系统：

## 3.3. Windows音频流类别

为了通知系统音频流的使用，应用程序可以选择用特定的音频流类别标记该流。在创建音频流之后，应用程序可以使用任何音频API来设置音频类别。在Windows 10中，有九种音频流类别。



|  |  |
| --- | --- |
| **类别** | 描述 |
| 电影(Movie)\* | 带有对话框的电影，视频(替换ForegroundOnlyMedia) |
| 媒体(Media)\* | 媒体播放的默认类别（替换BackgroundCapableMedia） |
| 游戏聊天(Game Chat)\* | 游戏内用户之间的通信（Windows 10中的新类别） |
| 语音(Speech)\* | 语音输入（例如个人助理）和输出（例如导航应用程序）（Windows 10中的新类别） |
| 通讯(Communications) | VOIP，实时聊天 |
| 警报(Alerts) | 闹钟，铃声，通知 |
| 声音特效(Sound Effects) | 哔声，叮当声等 |
| 游戏媒体(Game Media) | 游戏音乐 |
| 游戏特效(Game Effects) | 球弹跳的声音，汽车发动机声音，子弹声等。 |
| 其他 | 未分类的流 |

如前所述，音频类别（由应用程序选择）被映射到音频模式（由驱动程序定义）。应用程序可以使用10种音频类别之一标记其每个流。

应用程序没有选项来更改音频类别和信号处理模式之间的映射。应用程序不了解“音频处理模式”的概念。他们无法找出每个流使用哪种模式。

### 3.3.1. WASAPI代码示例

WASAPIAudio示例中的以下WASAPI代码显示了如何设置不同的音频类别。

// The ActivateAudioInterfaceAsync is a replacment for IMMDevice::Activate

IActivateAudioInterfaceAsyncOperation \*asyncOp = nullptr;

HRESULT hr = S\_OK;

String ^defaultRender = Windows::Media::Devices::MediaDevice::GetDefaultAudioRenderId( Windows::Media::Devices::AudioDeviceRole::Default );

hr = ActivateAudioInterfaceAsync( defaultRender->Data(), \_\_uuidof( IAudioClient3 ), nullptr, this, &asyncOp );

if ( FAILED( hr ) ) { … }

…

// the app’s implementation of IActivateAudioInterfaceCompetionHandler is invoked asynchronously

HRESULT ActivateAudioInterfaceCompletionHandler::ActivateCompleted( IActivateAudioInterfaceAsyncOperation \*activateOperation ) {

HRESULT hr = S\_OK;

HRESULT hrActivateResult = S\_OK;

IUnknown \*pUnknown = nullptr;

IAudioClient3 \*pAudioClient3 = nullptr;

hr = activateOperation->GetActivateResult( &hrActivateResult, &pUnknown );

if ( FAILED( hr ) ) { … }

if ( FAILED( hrActivateResult ) ) { … }

hr = pUnknown->QueryInterface( IID\_PPV\_ARGS( &pAudioClient3 ) );

if ( FAILED( hr ) ) { … }

// The IAudioClient3::SetClientProperties call needs to happen after activation completes,

// but before the call to IAudioClient3::Initialize or IAudioClient3::InitializeSharedAudioStream.

AudioClientProperties props = {};

props.cbSize = sizeof(props);

props.eCategory = AudioCategory\_GameEffects;

pAudioClient3->SetClientProperties( &props );

if ( FAILED( hr ) ) { … }

hr = pAudioClient3->InitializeSharedAudioStream( … );

if ( FAILED( hr ) ) { … }

…

## 3.4. 信号处理模式和效果

OEM定义每种模式将使用哪种效果。Windows定义了十七种音频效果的列表。

有关如何将APO与模式关联的信息，请参见[实现音频处理对象](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/implementing-audio-processing-objects)。

应用程序可能会询问在RAW或非RAW模式下将对特定流应用什么效果。当效果或原始处理状态发生变化时，应用程序也可以要求得到通知。应用程序可以使用此信息来确定特定的流类别（例如通信）是否可用，或者是否仅使用RAW模式。如果只有RAW模式可用，则应用程序可以确定要添加多少自己的音频处理。

如果System.Devices.AudioDevice.RawProcessingSupported为true，则应用程序还可以选择在某些流上设置“使用RAW模式”标志。如果System.Devices.AudioDevice.RawProcessingSupported为false，则应用程序无法设置“使用RAW模式”标志。

除了RAW /非RAW，应用程序无法查看存在的模式数。

无论音频硬件配置如何，应用程序都应请求最佳音频效果处理。例如，将流标记为“通讯”将使Windows知道暂停背景音乐。

有关静态音频流类别的更多信息，请参见[AudioCategory](https://docs.microsoft.com/uwp/api/Windows.UI.Xaml.Media.AudioCategory)枚举和[MediaElement.AudioCategory](https://docs.microsoft.com/uwp/api/Windows.UI.Xaml.Media.AudioCategory)属性。

## 3.5. 系统效果的CLSID

### FX\_DISCOVER\_EFFECTS\_APO\_CLSID

这是MsApoFxProxy.dll“代理效果”的CLSID，它查询驱动程序以获取活动效果的列表；

FX\_DISCOVER\_EFFECTS\_APO\_CLSID = "{889C03C8-ABAD-4004-BF0A-BC7BB825E166}"

### KSATTRIBUTEID\_AUDIOSIGNALPROCESSING\_MODE

KSATTRIBUTEID\_AUDIOSIGNALPROCESSING\_MODE是内核流的标识符，该标识符标识所引用的特定属性是信号处理模式属性。

此处显示的#define语句可在KSMedia.h头文件中找到。

#define STATIC\_KSATTRIBUTEID\_AUDIOSIGNALPROCESSING\_MODE 0xe1f89eb5, 0x5f46, 0x419b, 0x96, 0x7b, 0xff, 0x67, 0x70, 0xb9, 0x84, 0x1

DEFINE\_GUIDSTRUCT("E1F89EB5-5F46-419B-967B-FF6770B98401", KSATTRIBUTEID\_AUDIOSIGNALPROCESSING\_MODE);

#define KSATTRIBUTEID\_AUDIOSIGNALPROCESSING\_MODE DEFINE\_GUIDNAMED(KSATTRIBUTEID\_AUDIOSIGNALPROCESSING\_MODE)

KSATTRIBUTEID\_AUDIOSIGNALPROCESSING\_MODE用于可以识别模式的驱动程序, 后者支持包含[KSATTRIBUTE\_LIST](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ks/ns-ks-ksattribute_list)的[KSDATARANGE](https://docs.microsoft.com/previous-versions/ff561658(v=vs.85))结构。此列表中有一个元素，即KSATTRIBUTE。KSATTRIBUTE结构的Attribute成员设置为KSATTRIBUTEID\_AUDIOSIGNALPROCESSING\_MODE。

## 3.6. 音效

以下音频效果可在Windows 10中使用。

|  |  |
| --- | --- |
| 回声消除（Acoustic Echo Cancellation, AEC） | 在声音流中存在回声时，通过回声消除（AEC）可以消除回声改善音频质量。 |
| 噪音抑制（Noise Suppression, NS） | 当音频流中存在噪声时，噪声抑制（NS）会抑制诸如嗡嗡声和嗡嗡声之类的噪声。 |
| 自动增益控制（AGC） | 自动增益控制（AGC）-旨在在输入信号的幅度变化的情况下在其输出端提供受控信号幅度。平均或峰值输出信号电平用于将输入到输出增益动态调整为合适的值，即使输入信号电平范围很广，也能实现稳定的输出电平。 |
| 波束成形（BF） | 波束成形（BF）是一种用于定向信号传输或接收的信号处理技术。这是通过将相控阵(Phased array) 中的元素组合在一起的方式实现的，以使特定角度的信号发生相长干扰(constructive interference)，而其他角度的信号发生相消干扰(destructive interference)。与全向接收/发送(omnidirectional reception/transmission)相比的改进被称为接收/发送增益（或损耗）。 |
| 恒定音调去除(Constant Tone Removal) | 恒定音调去除用于衰减恒定的背景噪音，例如磁带嘶嘶声，电风扇或嗡嗡声。 |
| 均衡器(Equalizer) | 均衡器效果用于使用线性滤波器来更改音频系统的频率响应。这样可以增强信号的不同部分，类似于高音或低音设置。 |
| 响度均衡器(Loudness Equalizer) | 响度均衡器效果通过调整音频输出的音量来减小感知到的音量差异，从而使更大声和更安静的声音更接近平均响度水平。 |
| 低音增强(Bass Boost) | 在诸如笔记本电脑之类的扬声器的低音能力有限的系统中，有时可以通过增强扬声器所支持的频率范围内的低频响应来提高音频的感知质量。低音增强通过增加中低音范围的增益来改善扬声器非常小的移动设备上的声音。 |
| 虚拟环绕(Virtual Surround) | 虚拟环绕声使用简单的数字方法将多声道信号合并为两个声道。通过使用大多数现代音频接收器中可用的Pro Logic解码器，可以将转换后的信号恢复为原始的多通道信号。对于具有两声道声音硬件和具有环绕声增强机制的接收器的系统，虚拟环绕声是理想的选择。 |
| 虚拟耳机(Virtual Headphones) | 虚拟环绕声使戴着耳机的用户能够从前到后以及从一侧到另一侧区分声音。这是通过传输空间提示(Spatial cues)来完成的，这些提示可以帮助大脑定位声音并将它们整合到声场中。这样可以使声音听起来像超越耳机一样，从而产生“头顶外”的聆听体验。通过使用称为头相关传递函数（HRTF）的先进技术，可以达到此效果。HRTF生成基于人头形状的声音提示。这些提示不仅可以帮助听众找到声音的方向和来源，还可以增强听众周围的声学环境。 |
| 声道扩展(Speaker Fill) | 大多数音乐仅有两个通道，因此并未针对音频或视频发烧友的多通道音频设备进行优化。仅从左前和右前扬声器发出音乐的音频体验是不够理想的。 通道扩展模拟多声道扬声器设置。 它允许在房间中的所有扬声器上播放原本只能在两个扬声器上听到的音乐，从而增强了空间感。 |
| 房间校准(Room Correction) | 通过自动计算延迟，频率响应和增益调整的最佳组合，这种处理方式可以优化房间中特定位置（例如沙发的中央坐垫）的聆听体验。  房间校正功能可以更好地使声音与视频屏幕上的图像匹配，并且在桌面扬声器放置在非标准位置的情况下也很有用。房间校正处理是对高端接收器中类似功能的改进，因为它可以更好地说明人耳处理声音的方式。  校准在麦克风的帮助下进行，并且该程序可用于立体声和多声道系统。用户将麦克风放在用户打算坐在的地方，然后激活一个测量房间响应的向导程序。该向导程序依次从每个扬声器播放一组特殊设计的音调，并从麦克风的位置测量每个扬声器的距离，频率响应和整体增益。 |
| 低音管理(Bass Management) | 有两种低音管理模式：前低音管理(forward bass management)和后低音管理(reverse bass management)。  前低音管理可滤除音频数据流的低频部分。根据可以处理深低音 (Deep bass frequencies, 16~80Hz) 的通道的情况, 前低音管理算法将滤波后的输出重定向到低音炮或左前和右前扬声器通道。该处理基于LRBig标志的设置。  若要设置LRBig标志，用户可以使用“控制面板”中的“声音”小程序访问“低音管理设置”对话框。用户选中一个复选框以指示例如右前和左前扬声器处于全音域，并且此操作将设置LRBig标志。要清除此标志，请单击复选框以清除它。  反向低音管理将信号从低音炮通道分配到其他输出通道。根据LRBig标志的设置，信号被定向到所有通道，或定向到左前和右前通道。将低音炮信号混入其他通道时，此过程会大幅降低增益。  使用的低音管理模式取决于低音炮的可用性以及主扬声器的低音处理能力。在Windows中，用户通过“控制面板”中的“声音”小程序提供此信息。 |
| 环境音效(Environmental Effects) | 环境以逆袭啊通过更准确地模拟真实的音频环境来提高音频播放的真实性。您可以选择多种不同的环境，例如，“体育场”模拟运动场的声音。 |
| 扬声器保护(Speaker Protection) | 扬声器保护的目的是抑制共振频率，该共振频率会导致扬声器对PC的任何系统组件造成物理伤害。例如，某些物理硬盘驱动器可能会因以正确的频率播放响亮的声音而损坏。其次，当信号超过一定值时，扬声器保护会通过衰减信号来最大程度地减少对扬声器的损坏。 |
| 扬声器补偿(Speaker Compensation) | 一些扬声器比其他扬声器更擅长再现声音。例如，特定的扬声器可能会衰减100 Hz以下的声音。有时，音频驱动程序和固件DSP解决方案对正在播放的扬声器的特定性能特性有所了解，并且可以添加旨在补偿扬声器限制的处理。例如，可以创建一个端点效应（EFX），将增益应用于低于100 Hz的频率。当与物理扬声器中的衰减结合使用时，这种效果会改善的音频保真度 |
| 动态范围压缩(Dynamic Range Compression) | 动态范围压缩通过缩小或“压缩”音频信号的动态范围来放大安静的声音。音频压缩会放大低于特定阈值的安静声音，而大声声音不受影响。 |

此处显示的#define语句可在KSMedia.h头文件中找到。

// DEFAULT, 默认

#define STATIC\_AUDIO\_SIGNALPROCESSINGMODE\_DEFAULT 0xc18e2f7e, 0x933d, 0x4965, 0xb7, 0xd1, 0x1e, 0xef, 0x22, 0x8d, 0x2a, 0xf3

DEFINE\_GUIDSTRUCT("C18E2F7E-933D-4965-B7D1-1EEF228D2AF3", AUDIO\_SIGNALPROCESSINGMODE\_DEFAULT);

#define AUDIO\_SIGNALPROCESSINGMODE\_DEFAULT DEFINE\_GUIDNAMED(AUDIO\_SIGNALPROCESSINGMODE\_DEFAULT)

// RAW

#define STATIC\_AUDIO\_SIGNALPROCESSINGMODE\_RAW 0x9e90ea20, 0xb493, 0x4fd1, 0xa1, 0xa8, 0x7e, 0x13, 0x61, 0xa9, 0x56, 0xcf

DEFINE\_GUIDSTRUCT("9E90EA20-B493-4FD1-A1A8-7E1361A956CF", AUDIO\_SIGNALPROCESSINGMODE\_RAW);

#define AUDIO\_SIGNALPROCESSINGMODE\_RAW DEFINE\_GUIDNAMED(AUDIO\_SIGNALPROCESSINGMODE\_RAW)

// ACOUSTIC ECHO CANCELLATION, 回声消除

#define STATIC\_AUDIO\_EFFECT\_TYPE\_ACOUSTIC\_ECHO\_CANCELLATION 0x6f64adbe, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adbe-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_ACOUSTIC\_ECHO\_CANCELLATION);

#define AUDIO\_EFFECT\_TYPE\_ACOUSTIC\_ECHO\_CANCELLATION DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_ACOUSTIC\_ECHO\_CANCELLATION)

// NOISE SUPPRESSION, 噪音抑制

#define STATIC\_AUDIO\_EFFECT\_TYPE\_NOISE\_SUPPRESSION 0x6f64adbf, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adbf-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_NOISE\_SUPPRESSION);

#define AUDIO\_EFFECT\_TYPE\_NOISE\_SUPPRESSION DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_NOISE\_SUPPRESSION)

// AUTOMATIC GAIN CONTROL, 自动增益控制

#define STATIC\_AUDIO\_EFFECT\_TYPE\_AUTOMATIC\_GAIN\_CONTROL 0x6f64adc0, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adc0-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_AUTOMATIC\_GAIN\_CONTROL);

#define AUDIO\_EFFECT\_TYPE\_AUTOMATIC\_GAIN\_CONTROL DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_AUTOMATIC\_GAIN\_CONTROL)

// BEAMFORMING, 波束成形

#define STATIC\_AUDIO\_EFFECT\_TYPE\_BEAMFORMING 0x6f64adc1, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adc1-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_BEAMFORMING);

#define AUDIO\_EFFECT\_TYPE\_BEAMFORMING DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_BEAMFORMING)

// CONSTANT TONE REMOVAL, 恒定音调去除

#define STATIC\_AUDIO\_EFFECT\_TYPE\_CONSTANT\_TONE\_REMOVAL 0x6f64adc2, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adc2-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_CONSTANT\_TONE\_REMOVAL);

#define AUDIO\_EFFECT\_TYPE\_CONSTANT\_TONE\_REMOVAL DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_CONSTANT\_TONE\_REMOVAL)

// EQUALIZER, 均衡器

#define STATIC\_AUDIO\_EFFECT\_TYPE\_EQUALIZER 0x6f64adc3, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adc3-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_EQUALIZER);

#define AUDIO\_EFFECT\_TYPE\_EQUALIZER DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_EQUALIZER)

// LOUDNESS EQUALIZER, 响度均衡器

#define STATIC\_AUDIO\_EFFECT\_TYPE\_LOUDNESS\_EQUALIZER 0x6f64adc4, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adc4-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_LOUDNESS\_EQUALIZER);

#define AUDIO\_EFFECT\_TYPE\_LOUDNESS\_EQUALIZER DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_LOUDNESS\_EQUALIZER)

// BASS BOOST, 低音增强

#define STATIC\_AUDIO\_EFFECT\_TYPE\_BASS\_BOOST 0x6f64adc5, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adc5-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_BASS\_BOOST);

#define AUDIO\_EFFECT\_TYPE\_BASS\_BOOST DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_BASS\_BOOST)

// VIRTUAL SURROUND, 虚拟环绕

#define STATIC\_AUDIO\_EFFECT\_TYPE\_VIRTUAL\_SURROUND 0x6f64adc6, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adc6-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_VIRTUAL\_SURROUND);

#define AUDIO\_EFFECT\_TYPE\_VIRTUAL\_SURROUND DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_VIRTUAL\_SURROUND)

// VIRTUAL HEADPHONES, 虚拟耳机

#define STATIC\_AUDIO\_EFFECT\_TYPE\_VIRTUAL\_HEADPHONES 0x6f64adc7, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adc7-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_VIRTUAL\_HEADPHONES);

#define AUDIO\_EFFECT\_TYPE\_VIRTUAL\_HEADPHONES DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_VIRTUAL\_HEADPHONES)

// ROOM CORRECTION, 房间校准

#define STATIC\_AUDIO\_EFFECT\_TYPE\_ROOM\_CORRECTION 0x6f64adc9, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adc9-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_ROOM\_CORRECTION);

#define AUDIO\_EFFECT\_TYPE\_ROOM\_CORRECTION DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_ROOM\_CORRECTION)

// BASS MANAGEMENT, 低音管理

#define STATIC\_AUDIO\_EFFECT\_TYPE\_BASS\_MANAGEMENT 0x6f64adca, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adca-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_BASS\_MANAGEMENT);

#define AUDIO\_EFFECT\_TYPE\_BASS\_MANAGEMENT DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_BASS\_MANAGEMENT)

// ENVIRONMENTAL EFFECTS, 环境音效

#define STATIC\_AUDIO\_EFFECT\_TYPE\_ENVIRONMENTAL\_EFFECTS 0x6f64adcb, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adcb-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_ENVIRONMENTAL\_EFFECTS);

#define AUDIO\_EFFECT\_TYPE\_ENVIRONMENTAL\_EFFECTS DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_ENVIRONMENTAL\_EFFECTS)

// SPEAKER PROTECTION, 扬声器保护

#define STATIC\_AUDIO\_EFFECT\_TYPE\_SPEAKER\_PROTECTION 0x6f64adcc, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adcc-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_SPEAKER\_PROTECTION);

#define AUDIO\_EFFECT\_TYPE\_SPEAKER\_PROTECTION DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_SPEAKER\_PROTECTION)

// SPEAKER COMPENSATION, 扬声器补偿

#define STATIC\_AUDIO\_EFFECT\_TYPE\_SPEAKER\_COMPENSATION 0x6f64adcd, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adcd-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_SPEAKER\_COMPENSATION);

#define AUDIO\_EFFECT\_TYPE\_SPEAKER\_COMPENSATION DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_SPEAKER\_COMPENSATION)

// DYNAMIC RANGE COMPRESSION, 动态范围压缩

#define STATIC\_AUDIO\_EFFECT\_TYPE\_DYNAMIC\_RANGE\_COMPRESSION 0x6f64adce, 0x8211, 0x11e2, 0x8c, 0x70, 0x2c, 0x27, 0xd7, 0xf0, 0x01, 0xfa

DEFINE\_GUIDSTRUCT("6f64adce-8211-11e2-8c70-2c27d7f001fa", AUDIO\_EFFECT\_TYPE\_DYNAMIC\_RANGE\_COMPRESSION);

#define AUDIO\_EFFECT\_TYPE\_DYNAMIC\_RANGE\_COMPRESSION DEFINE\_GUIDNAMED(AUDIO\_EFFECT\_TYPE\_DYNAMIC\_RANGE\_COMPRESSION)

# 4. 定制音频驱动程序

供应商需要为不兼容UAA的音频设备提供自定义驱动程序。另外，与UAA兼容的音频适配器可以包含UAA类驱动程序不支持的专有功能；例如，仅在供应商提供自定义音频驱动程序的情况下，应用程序才能使用这些功能。通过系统提供的UAA驱动程序只能访问标准UAA功能。有关UAA支持的功能的信息，请参见《[通用音频体系结构》](https://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/UAA_Guidelines.doc)白皮书。

硬件供应商可以有两种方案来编写自定义音频驱动程序：开发与PortCls系统驱动程序（Portcls.sys）一起使用的*自定义音频适配器驱动程序*，或开发与AVStream类系统驱动程序一起使用（Ks.sys ）的*自定义微型驱动程序(minidriver)*。

大多数*自定义音频适配器驱动程序*都使用PortCls，它是操作系统的一部分。PortCls系统驱动程序（Portcls.sys）包含*内置的音频驱动程序基础结构*(built-in audio-driver infrastructure)，使编写*自定义音频驱动程序*的任务更加容易。PortCls实现了几个端口驱动程序，每个端口驱动程序专门用于管理特定类型的wave，MIDI或混音器设备的通用功能。在选择了一组适当的*端口驱动程序*(Port driver) 以管理音频适配器上的音频功能之后，供应商需要开发一组补充的*微型端口驱动程序 (miniport drivers)* ，这些驱动程序可以与所选的*端口驱动程序*一起工作，并控制音频设备的硬件相关功能。

供应商还可以通过开发自定义的AVStream类*微型驱动程序(minidriver)*来支持音频设备。*微型驱动程序*与作为操作系统一部分提供的AVStream类系统驱动程序一起使用。与使用PortCls相比，实现AVStream驱动程序更加困难，但是对于集成音频和视频的设备而言，也许这样做仍然合适。对于不兼容系统提供的USBAudio或AVCAudio类系统驱动程序的USB或IEEE 1394音频设备，可能也需要AVStream驱动程序。

对于几乎所有需要供应商提供自定义驱动程序的PCI音频适配器，供应商应选择PortCls。

AVStream类系统驱动程序（Ks.sys）缺少大多数(PortCls中支持的)音频特有的功能。

有关PortCls的更多信息，请参见[Port Class简介](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/introduction-to-port-class)。有关AVStream的更多信息，请参见[AVStream概述](https://docs.microsoft.com/windows-hardware/drivers/stream/avstream-overview)。

# 5.指定拓扑

硬件供应商确定要为wave和MIDI设备编写哪些微型端口驱动程序(miniport driver)后，下一步就是表示这些设备的内核流（kernel streaming, KS）拓扑。KS拓扑由一组数据结构组成，这些数据结构描述了音频或MIDI流流经每个设备时遵循的数据路径。通过此拓扑，驱动程序将公开位于每个路径上的控制节点（例如，音量控制）。通常，应用程序使用Windows多媒体mixerXxx函数通过枚举每条路径上节点的顺序来探索拓扑。例如，在发现音量级别控制节点之后，应用程序可以在该节点上设置音量级别。有关Windows多媒体的更多信息，请参见Microsoft Windows SDK文档。有关mixerXxx函数如何表示KS拓扑的更多信息，请参见[内核流拓扑到音频混合器API的转换](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-streaming-topology-to-audio-mixer-api-translation)。

## 译注: KS filter = port driver + miniport driver

PortCls提供六个端口驱动程序：WavePci，WaveCyclic，WaveRT，MIDI，DMus和Topology(拓扑)。（WaveRT自Windows Vista开始提供，是推荐的方法。）*拓扑端口驱动程序(Topology port driver)*控制音频适配器电路的一部分，该部分将wave和MIDI设备的渲染流混合在一起。它还可以控制从输入插孔选择捕获流。尽管其名称有些误导，但*拓扑端口驱动程序*并未体现音频适配器的全部拓扑，尽管通常它确实包含了很大一部分。其他端口驱动程序负责适配器拓扑的其余部分。

每个*端口驱动程序(port driver)*都与相应的*微型端口驱动程序(miniport driver)*配对以形成[KS filter](https://docs.microsoft.com/windows-hardware/drivers/stream/ks-filters)，该filter代表音频适配器上的特定设备（波形，MIDI或混音器），如下表所示。

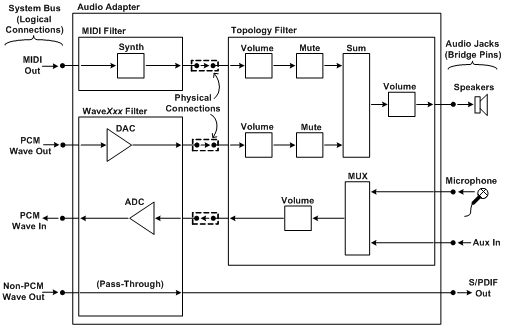
| **filter类型** | **描述** |
| --- | --- |
| Wave *Xxx* filter | 表示将波形输出流(wave output stream)转换为模拟音频信号(analog audio signal)或将模拟音频信号转换为波形输入流的波形设备(wave device)。 |
| MIDI或DMus filter | 表示播放或捕获MIDI流的MIDI设备。 |
| 拓扑filter | 代表适配器的混音器电路。 |

微型端口驱动程序实现filter**特定于设备的功能，**包括定义设备所包含的适配器拓扑结构的一部分。端口驱动程序负责每种filter的**常规filter操作**，包括与操作系统的通信**。**

## 译注:filter 连成图

每个filter都有一个或多个[KS引脚](https://docs.microsoft.com/windows-hardware/drivers/stream/ks-pins)，用作音频数据流进入和离开filter的路径。通常，拓扑filter上的引脚通过适配器电路中的硬线连接与波形filter，MIDI和DMusfilter上的引脚相连。这些filter及其互连一起形成了一个KSfilter图(graph)，该图体现了适配器的拓扑。

下图显示了示例音频适配器的拓扑。



在上图中，顶层的拓扑由MIDI，Wave *Xxx*和拓扑filter之间的连接组成。此外，每个filter都有自己的内部拓扑，该拓扑由通过filter的数据路径和沿每个路径的控制节点组成。节点的标签如下表所示。

| **标签** | **描述** | **KS节点类型GUID** |
| --- | --- | --- |
| 合成器 | 合成器节点 | [**KSNODETYPE\_SYNTHESIZER**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-synthesizer) |
| DAC | 数音频转换器节点 | [**KSNODETYPE\_DAC**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-dac) |
| ADC | 模数转换器节点 | [**KSNODETYPE\_ADC**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-adc) |
| 音量 | 音量级别控制节点 | [**KSNODETYPE\_VOLUME**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-volume) |
| 静音 | 静音控制节点 | [**KSNODETYPE\_MUTE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-mute) |
| 加法器 | 累加节点 | [**KSNODETYPE\_SUM**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-sum) |
| MUX | 多路复用器节点 | [**KSNODETYPE\_MUX**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-mux) |

在上图中，音频适配器左侧的针脚代表逻辑连接（不是物理连接），数据流通过这些逻辑连接从系统总线进入适配器或从适配器进入系统总线。这些引脚在逻辑上连接到适配器外部其他filter（未显示）上的源极和漏极引脚。通常，这些filter是软件模块，它们与适配器拓扑一起形成一个较大的filter图，可以使用混合器*Xxx*功能的应用程序探索其拓扑。例如，上图中标记为“ PCM Wave Out”的引脚在逻辑上连接到Windows中的用户模式音频引擎。这些逻辑连接通过系统总线上的DMA传输来维护。

但是，拓扑filter左边缘上的引脚实际上连接到MIDI和Wave *Xxx* filter上的引脚。这些连接是硬连线的，不能通过软件更改。

音频适配器右侧的桥接针代表系统机箱上的音频插孔。这些引脚称为*桥接引脚，*因为它们桥接了KSfilter图和外部世界之间的边界。

## 译注: filter有属性

filter，引脚和节点通常具有音频驱动程序的客户端（内核模式组件或用户模式应用程序）可访问的属性。客户端可以向filter，引脚或节点发送[KS属性请求](https://docs.microsoft.com/windows-hardware/drivers/stream/ks-properties)，以查询属性的当前值或更改属性值。例如，一个音量控制节点具有[**KSPROPERTY\_AUDIO\_VOLUMELEVEL**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-volumelevel)属性，客户端可以通过KS属性请求来更改该属性。求加法器节点是通常没有属性的节点类型的示例。

## 译注: 各种可能的拓扑

为简单起见，上图中的Wave *Xxx* filter仅提供了一个引脚，用于接受来自系统总线的PCM波输出流。但是，某些波形设备为PCM波形输出提供多个引脚，并包含用于内部混合进入引脚的流的硬件。这些设备通过接受从应用程序声音缓冲区播放的PCM流，为使用DirectSound的应用程序提供硬件加速。为了使DirectSound使用这些引脚，它们必须提供额外的节点来进行二维（2-D）和三维（3-D）处理，如[WDM Audio](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/directsound-hardware-acceleration-in-wdm-audio)中的[DirectSound硬件加速中](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/directsound-hardware-acceleration-in-wdm-audio)所述。

Windows Server 2003，Windows XP，Windows 2000和Windows Me / 98支持这种类型的硬件加速，但Windows Vista不支持。Windows Vista不使用旧的wave设备上的硬件加速引脚。

在上图中，MIDI，Wave *Xxx*和拓扑之间的物理连接都传输模拟音频信号。但是，设备可能采用不同的拓扑, 比如接受MIDI和wave设备的数字输出流，对其进行数字混合，然后将数字混音转换为模拟输出信号，从而达到类似的效果。

上图左下角的“非PCM波形输出”引脚接受S / PDIF直通格式的非PCM输出流，例如AC-3-over-S / PDIF或WMA Pro -over-S / PDIF。使用这些格式中的一种，设备仅通过S / PDIF链路发送压缩数据，而无需对数据进行解码。因此，到上图右下角的“ S / PDIF输出”引脚的数据路径不包含任何音量或静音节点。有关非PCM音频格式和S / PDIF直通传输的更多信息，请参阅[支持非PCM波形格式](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/supporting-non-pcm-wave-formats)。更多信息，标题为白皮书提供音频驱动程序支持的WMA Pro的环比S / PDIF格式的[音频技术](https://go.microsoft.com/fwlink/p/?linkid=8751)的网站。

微型端口驱动程序以**[PCFILTER\_DESCRIPTOR](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-pcfilter_descriptor)**结构的形式向端口驱动程序显示其拓扑。该结构描述了filter的所有引脚和节点，并指定了引脚和节点如何相互连接。

代替上图所示设计单片(monolithic)拓扑filter，可以将音频适配器中的混频器电路划分为几个拓扑filter。例如，在上图中，驱动扬声器的数据路径可以实现为一个拓扑filter，而捕获来自输入设备的音频数据的数据路径可以实现为单独的拓扑filter。当不使用特定拓扑filter中的数据路径时，可以关闭适配器的该部分电源，而无需禁用整个适配器。有关更多信息，请参见[动态音频子设备](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/dynamic-audio-subdevices)。

# 6. 指定引脚数据范围

定义拓扑以表示设备中的数据路径和控制节点后，下一步就是定义每个可配置引脚的[数据范围](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-data-ranges)。可配置引脚可以在软件控制下创建，配置并连接，并连接到wave或MIDI流。相反，物理连接或桥接引脚隐式存在，无法在软件控制下创建或配置。

在连接可配置的引脚以用作Wave或MIDI流的接收器或源之前，必须将引脚配置为处理该流的数据格式。通常，该引脚可以配置为接受多种流格式之一。例如，PCM波输出引脚可能接受以下范围的PCM流参数：

* 采样率11.025 kHz，22.05 kHz，44.1 kHz和48 kHz
* 8、16、24和32位的样本大小
* 从1到8的任意数量的通道

对于每种类型的可配置引脚，微型端口驱动程序描述了该引脚可以处理的各种流数据格式。可以将这些参数范围指定为数据范围描述符的数组，如下面的代码示例所示。

static KSDATARANGE\_AUDIO PinDataRangesPcm[] =

{

{

{

sizeof(KSDATARANGE\_AUDIO),

0,

0,

0,

STATICGUIDOF(KSDATAFORMAT\_TYPE\_AUDIO),

STATICGUIDOF(KSDATAFORMAT\_SUBTYPE\_PCM),

STATICGUIDOF(KSDATAFORMAT\_SPECIFIER\_WAVEFORMATEX)

},

8, // Maximum number of channels

8, // Minimum number of bits-per-sample

32, // Maximum number of bits-per-channel

11025, // Minimum rate

48000 // Maximum rate

}

};

请注意，前面示例中的PinDataRangesPcm数组包含**[KSDATARANGE\_AUDIO](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-ksdatarange_audio)**类型的单个数据范围描述符。更一般而言，数据范围数组可以包含任意数量的描述符。例如，非PCM波输出引脚可能同时支持AC-3-over-S / PDIF和WMA Pro-over-S / PDIF格式。这两种格式中的每一种都由单独的数据范围描述符指定。因此，引脚的数据范围数组将至少包含两个KSDATARANGE\_AUDIO结构。

有的应用程序使用DirectMusic或Windows多媒体midiInxxx xxx和midiOutxxx函数, 为了支持来自这些应用程序的音乐流格式,可配置的引脚使用一个[**KSDATARANGE\_MUSIC**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-ksdatarange_music)类型的数据描述符范围。

端口驱动程序从微型端口驱动程序获取数据范围信息，并在可能的情况下,使用此信息来处理有关每个引脚可以支持的数据格式的信息请求。对于具有简单PCM数据范围的引脚，端口驱动器能够处理该引脚的连接请求 (intersection request) 。在连接请求中，客户端提供一组数据范围，这些数据范围表示流的可能数据格式。如果可能，端口驱动程序的连接处理程序会从请求中的数据范围中选择一种特定的数据格式，该格式也应属于其引脚的数据范围。此格式表示两组数据范围的交集。因此，客户端和引脚都可以处理这种格式的流。对于更复杂的数据范围，微型端口驱动程序可以提供自己的连接处理程序，然后端口驱动程序将使用它而不是其自己的默认处理程序。微型端口驱动程序的连接处理程序可以允许可能难以将其表达为数据范围数组的任何格式要求。有关更多信息，请参见[数据交叉处理程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/data-intersection-handlers)。更多信息，标题为白皮书提供多声道音频数据和WAVE文件的[音频技术](https://go.microsoft.com/fwlink/p/?linkid=8751)的网站。