**WDM在不同Windows版本上的音频支持**

本节描述不同版本Windows操作系统对音频设备和驱动的支持

# 1. 实现音频模块通信

音频模块是功能相对独立的独特音频处理逻辑。音频模块可以驻留在音频驱动器或音频DSP中。音频模块的一个例子是基于DSP的音频处理。

从Windows 10版本1703开始，同时提供API和DDI以支持通用Windows平台（UWP）应用程序和内核模式设备驱动程序间的通信。

本主题提供有关在内核设备驱动程序中实现音频模块通信的信息。

以及如何使用UWP app从音频设备模块发送命令和接收更改通知的信息。

## 1.1. 为什么使用音频模块？

OEM通常会在其系统上捆绑一个配置应用程序，以使客户可以控制此音频系统的各个方面，并根据自己的喜好对其进行调整。音频子系统可以包含各种组件，例如主机上的(on-host)音频处理对象(APO)，硬件DSP处理以及专用硬件（例如智能放大器,smart amp）（还有音频编解码器本身）。在大多数情况下，这些组件是由不同的供应商创建和出售的。之前的做法是，IHV创建自己的私有API，以彼此集成并在各个组件之间发送信息。然后，现有的WIN32配置应用程序会利用这些专用API。

在[通用Windows平台（UWP）上](https://docs.microsoft.com/windows/uwp/get-started/universal-application-platform-guide)，提供了一组API，让单个应用程序可以在Windows系列的多个平台上运行。UWP还引入了新的外观，期望在Windows 10上运行的应用程序都能采用。因此，许多OEM都希望在UWP上构建其音频配置应用程序。但是，*UWP（AppContainer沙箱）的核心安全功能*阻止了,从应用程序到音频子系统中其他组件的,通信。这导致私有APO在UWP上是不可用(在之前的配置应用程序上可用)。

从Windows 10版本1703开始，音频模块UWP API允许*配置应用程序和用户模式组件*与*内核和硬件层中的模块(可通过新KS属性集发现)*进行通信。音频IHV和ISV可以编写应用程序和服务，使用Windows提供的定义明确的接口, 与其硬件模块进行通信。有关音频模块API的更多信息，请参见[Windows.Media.Devices命名空间。](https://docs.microsoft.com/uwp/api/Windows.Media.Devices)

## 1.2. 音频模块定义

这些定义是音频模块专有的。

| **术语** | **定义** |
| --- | --- |
| 音频模块 | 执行相对独立功能的独特音频处理逻辑。可以驻留在音频驱动程序或音频DSP中。音频模块的一个例子是音频处理对象（APO）。 |

## 1.3. 通用音频定义

这些定义通常在使用音频驱动程序时使用。

| **术语** | **定义** |
| --- | --- |
| OEM | 原始设备制造商 |
| IHV | 独立硬件供应商 |
| ISV | 独立软件供应商 |
| HSA | 硬件支持应用 |
| UWP | 通用Windows平台 |
| APO | 音频处理对象 |
| DSP | 数字信号处理 |

## 1.4. 架构

音频模块采用Windows支持的机制，以在用户模式和内核模式音频组件之间发送消息。一个重要的区别是音频模块标准化了传输管道。它没有规定在此之上的通信协议，而是让ISV和IHV来定义协议(An important distinction is that Audio Modules standardizes the transport pipeline. It does not establish the communication protocol over that transport and relies on the ISVs and IHVs to define the protocol) 。目的是允许现有的第三方设计只需很少的更改即可轻松迁移到音频模块。

音频模块API通过两种不同的定向方法提供对模块的访问：KS wave filter和已初始化的KS引脚（流）。特定模块的位置和访问都是定制的。

HSA和其他应用程序将只能通过filter句柄访问可用的模块。只有流上加载的APO个体,可以访问流所指向的音频模块(The individual APOs loaded on a stream are the only objects that will have access to the stream targeted audio modules)。

有关APO的更多信息，请参见[Windows音频处理对象](https://docs.microsoft.com/windows-hardware/drivers/audio/windows-audio-processing-objects)。

### 发送命令

对音频模块查询和更改参数的途径是将命令发送到*内核中的音频模块和音频子系统中的硬件组件*。音频模块API的命令结构是松散定义的，它规范化了发现模块和模块自我标识的方式。但是，详细的命令结构必须由相关的ISV和IHV设计和实现，以建立可发送哪些消息和预期响应的协议。

### 音频模块客户端的模块通知

如果客户端已经注册了特定模块上的通知，则音频微型端口就可以通知并传递信息到音频模块客户端。这些通知中传递的信息不是由音频模块API定义的，而是由ISV和/或IHV定义的。

### 启用，禁用和常规拓扑信息

音频模块API定义了如何枚举并将命令发送到模块。但是，API并未明确定义如何启用或禁用音频模块。同样，它也没有给出一种找到拓扑信息或模块相对位置的方式。IHV和ISV可以确定是否需要此功能，并决定如何实现此功能。

推荐的方式是暴露一个全局驱动程序模块。全局驱动程序模块应处理针对拓扑相关请求的自定义命令。

## 1.5. 音频模块DDI

### 内核流音频模块属性

译注:

* [DeviceIoControl](https://docs.microsoft.com/en-us/windows/win32/api/ioapiset/nf-ioapiset-deviceiocontrol)将控制代码直接发送到指定的设备驱动程序，使相应的设备执行相应的操作的API.
* 其前三个参数为:1.设备句柄, 2. Control code (这里应该是IOCTL\_KS\_PROPERTY), 3. KSPROPERTY数据结构, 其字段包括SET和ID还有flags.
* 以下三个属性的使用中, 都是需要调用函数DeviceIoControl, 使用filter或引脚句柄，使用控制代码IOCTL\_KS\_PROPERTY, 并将一个KSPROPERTY对象作为第三个参数, 其中set为KSPROPSETID\_AudioModule, id为对应的ID, 比如KSPROPERTY\_AUDIOMODULE\_DESCRIPTORS, 有时第三个参数是一个包含KSPROPERTY的其他对象, 比如KSAUDIOMODULE\_PROPERTY.
* 查看<https://github.com/microsoft/Windows-driver-samples/blob/master/audio/sysvad/EndpointsCommon/minwavert.cpp> <https://github.com/microsoft/Windows-driver-samples/blob/master/audio/sysvad/EndpointsCommon/AudioModuleHelper.cpp>可以了解一些driver内部的处理逻辑
* 以下相关部分我们有所简化

已为音频模块特有的三个属性定义了新KS属性集[KSPROPSETID\_AudioModule](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-audiomodule)。

PortCls微型端口驱动程序需要直接处理每个属性的响应，因为没有帮助程序接口。

#### ksmedia.h：

#define STATIC\_KSPROPSETID\_AudioModule \

0xc034fdb0, 0xff75, 0x47c8, 0xaa, 0x3c, 0xee, 0x46, 0x71, 0x6b, 0x50, 0xc6

DEFINE\_GUIDSTRUCT("C034FDB0-FF75-47C8-AA3C-EE46716B50C6", KSPROPSETID\_AudioModule);

#define KSPROPSETID\_AudioModule DEFINE\_GUIDNAMED(KSPROPSETID\_AudioModule)

typedef enum {

KSPROPERTY\_AUDIOMODULE\_DESCRIPTORS = 1,

KSPROPERTY\_AUDIOMODULE\_COMMAND = 2,

KSPROPERTY\_AUDIOMODULE\_NOTIFICATION\_DEVICE\_ID = 3,

} KSPROPERTY\_AUDIOMODULE;

### 音频模块描述符

对[KSPROPERTY\_AUDIOMODULE\_DESCRIPTORS](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audiomodule-descriptors)属性的支持, 令驱动程序可以识别音频模块。调用DeviceIoControl时输入缓冲区是一个KSPROPERTY. [KSAUDIOMODULE\_DESCRIPTOR](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudiomodule_descriptor)被用于描述音频硬件中的每个模块。作为这类请求的响应，会返回此描述符的数组.

#### ksmedia**.h：**

#define AUDIOMODULE\_MAX\_NAME\_SIZE 128

typedef struct \_KSAUDIOMODULE\_DESCRIPTOR

{

GUID ClassId;

ULONG InstanceId;

ULONG VersionMajor;

ULONG VersionMinor;

WCHAR Name[AUDIOMODULE\_MAX\_NAME\_SIZE];

} KSAUDIOMODULE\_DESCRIPTOR, \*PKSAUDIOMODULE\_DESCRIPTOR;

有关更多信息，请参见[KSAUDIOMODULE\_DESCRIPTOR](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudiomodule_descriptor)。

### 音频模块命令

对[KSPROPERTY\_AUDIOMODULE\_COMMAND](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audiomodule-command)属性的支持, 令音频模块客户端可以发送自定义命令来查询和设置音频模块上的参数。调用DeviceIoControl时输入缓冲区是一个[KSAUDIOMODULE\_PROPERTY](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudiomodule_property).客户端可以选择在输入缓冲区中紧接着[KSAUDIOMODULE\_PROPERTY](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudiomodule_property)发送其他附加信息, 以发送自定义命令。

#### ksmedia**.h：**

#define AUDIOMODULE\_MAX\_DATA\_SIZE 64000

typedef struct \_KSPAUDIOMODULE\_PROPERTY

{

KSPROPERTY Property;

GUID ClassId;

ULONG InstanceId;

} KSAUDIOMODULE\_PROPERTY, \*PKSPAUDIOMODULE\_PROPERTY;

有关更多信息，请参见[KSAUDIOMODULE\_PROPERTY](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudiomodule_property)。

### 音频模块通知设备ID

需要支持[KSPROPERTY\_AUDIOMODULE\_NOTIFICATION\_DEVICE\_ID](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audiomodule-notification-device-id)，以使微型端口能够(译注:+接受注册+)通知并将信息传递给音频模块客户端。此ID的生存期与该音频设备在Windows Audio堆栈中生成并活跃的周期有关。调用DeviceIoControl时输入缓冲区是一个KSPROPERTY.。

有关更多信息，请参见[KSAUDIOMODULE\_PROPERTY](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudiomodule_property)。

## 1.6. PortCls帮助函数-音频模块通知函数

为了帮助驱动程序开发人员向音频模块客户端发送通知, 添加了新的端口接口(port interface)如下。

### PortCls.h：

typedef struct \_PCNOTIFICATION\_BUFFER

{

UCHAR NotificationBuffer[1];

} PCNOTIFICATION\_BUFFER, \*PPCNOTIFICATION\_BUFFER;

DECLARE\_INTERFACE\_(IPortClsNotifications,IUnknown)

{

DEFINE\_ABSTRACT\_UNKNOWN() // For IUnknown

STDMETHOD\_(NTSTATUS, AllocNotificationBuffer)

( THIS\_

\_In\_ POOL\_TYPE PoolType,

\_In\_ USHORT NumberOfBytes,

\_Out\_ PPCNOTIFICATION\_BUFFER\* NotificationBuffer

) PURE;

STDMETHOD\_(void, FreeNotificationBuffer)

( THIS\_

\_In\_ PPCNOTIFICATION\_BUFFER NotificationBuffer

) PURE;

STDMETHOD\_(void, SendNotificationBuffer)

( THIS\_

\_In\_ const GUID\* NotificationId,

\_In\_ PPCNOTIFICATION\_BUFFER NotificationBuffer

) PURE;

};

//

// Audio module notification definitions.

//

#define STATIC\_KSNOTIFICATIONID\_AudioModule \

0x9C2220F0, 0xD9A6, 0x4D5C, 0xA0, 0x36, 0x57, 0x38, 0x57, 0xFD, 0x50, 0xD2

DEFINE\_GUIDSTRUCT("9C2220F0-D9A6-4D5C-A036-573857FD50D2", KSNOTIFICATIONID\_AudioModule);

#define KSNOTIFICATIONID\_AudioModule DEFINE\_GUIDNAMED(KSNOTIFICATIONID\_AudioModule)

typedef struct \_KSAUDIOMODULE\_NOTIFICATION {

union {

struct {

GUID DeviceId;

GUID ClassId;

ULONG InstanceId;

ULONG Reserved;

} ProviderId;

LONGLONG Alignment;

};

} KSAUDIOMODULE\_NOTIFICATION, \*PKSAUDIOMODULE\_NOTIFICATION;

有关更多信息，请参见：

[IPortClsNotifications](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iportclsnotifications)

[IPortClsNotifications :: AllocNotificationBuffer](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclsnotifications-allocnotificationbuffer)

[IPortClsNotifications :: FreeNotificationBuffer](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclsnotifications-freenotificationbuffer)

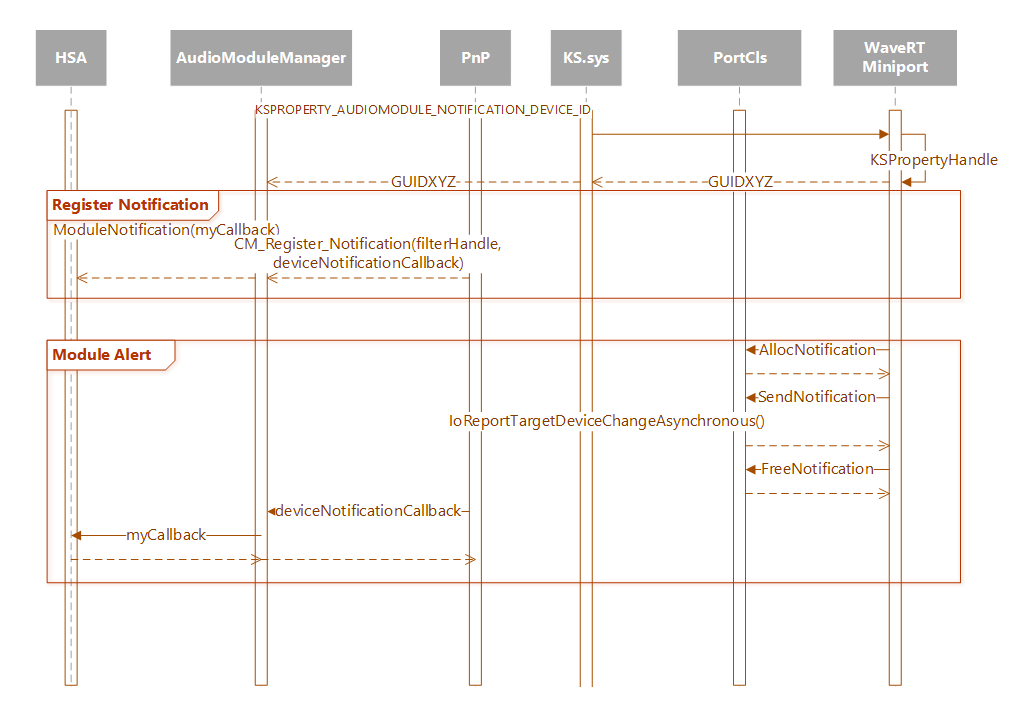
[IPortClsNotifications :: SendNotificationBuffer](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclsnotifications-sendnotification)

## 1.7. 调用顺序

微型端口将调用其端口(port)驱动以创建并发送通知。此图中显示了常规调用顺序。

译注:

* 第一部分的注册通知是音频模块的客户端使用了音频模块DDI的第三个属性实现了注册
* 第二部分是miniport使用了音频模块通知函数通知上层模块



# 2. 配置和查询音频设备模块

本文介绍如何通过UWP应用从音频设备模块发送命令和接收更改通知。音频设备模块可以是硬件效果处理单元，也可以是音频驱动程序定义的任何其他音频配置模块。此功能旨在使模块提供商能够创建UWP应用，这些应用允许用户控制并从运行在DSP中的音频处理模块中获取状态信息。为了使用本文中显示的音频设备模块API，必须在应用程序包清单中指定受限制的audioDeviceConfiguration功能。

## 2.1. 获取AudioDeviceModulesManager类的实例

本文中显示的所有音频设备模块操作都从获取[**AudioDeviceModulesManager**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodulesmanager)的实例开始。为此，请先调用[**MediaDevice**](https://docs.microsoft.com/uwp/api/windows.media.devices.mediadevice)类的静态[**GetDefaultAudioRenderId**](https://docs.microsoft.com/uwp/api/windows.media.devices.mediadevice.getdefaultaudiorenderid)方法。这将返回默认音频渲染设备的ID，然后将其传递到**AudioDeviceModulesManager**的构造函数中，以创建与音频设备关联的类的实例。

var endpointId = MediaDevice.GetDefaultAudioRenderId(AudioDeviceRole.Default);

var audioModuleManager = new AudioDeviceModulesManager(endpointId);

## 2.2. 查询已安装的音频设备模块

通过调用[**AudioDeviceModulesManager**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodulesmanager)类中的[**FindAll**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodulesmanager.findall)函数, 可以查询所有已安装的音频设备模块. 通过调用[**FindAllById**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodulesmanager.findallbyid)并传入请求的模块的ID，可以查询一组特定的音频设备模块。下面的示例为一组模块定义一个ID，调用**FindAllById**检索[**AudioDeviceModule**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodule)对象的列表，然后将每个模块的详细信息打印到调试输出。

public const string Contoso\_AudioDeviceModuleId = "F72E09C3-FEBA-4C50-93BE-2CA56123AF09";

var endpointId = MediaDevice.GetDefaultAudioRenderId(AudioDeviceRole.Default);

var audioModuleManager = new AudioDeviceModulesManager(endpointId);

var modules = audioModuleManager.FindAllById(Contoso\_AudioDeviceModuleId);

foreach (var module in modules)

{

var classId = module.ClassId;

var name = module.DisplayName;

var minorVersion = module.MinorVersion;

var majorVersion = module.MajorVersion;

var instanceId = module.InstanceId;

Debug.WriteLine($"{classId} : {name} : {minorVersion} : {majorVersion} : {instanceId}");

}

## 2.3. 向音频设备模块发送命令并接收结果数据

通过调用[**AudioDeviceModule**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodule)对象上的[**SendCommandAsync**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodule.sendcommandasync)函数将命令发送到音频设备模块。该**SendCommandAsync**方法以一个字节数组作为参数。通常，此字节数组包含一个命令标识符，紧接着与该命令关联的数据，但是命令格式和值完全由供应商定义，并且被系统视为透明的。

该**SendCommandAsync**方法返回一个异步操作，在完成后返回一个[**ModuleCommandResult**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodule.sendcommandasync)对象表示该命令的结果。该[**状态**](https://docs.microsoft.com/uwp/api/windows.media.devices.modulecommandresult.status)属性包含枚举值，指示系统是否能够执行的命令。这不一定表示音频设备模块能够成功执行命令。该[**结果**](https://docs.microsoft.com/uwp/api/windows.media.devices.modulecommandresult.result)属性包含由音频设备模块返回指示命令的状态的字节数组。通常，这是一个指示成功或失败以及随后的命令数据结果的值。与模块命令一样，模块响应格式和值由供应商定义。

下面的示例调用**FindAllAsync**以检索一组音频设备模块。调用了[**DataWriter**](https://docs.microsoft.com/uwp/api/windows.storage.streams.datawriter)**函数来**创建一个包含例子命令和数据的字节数组。**调用SendCommandAsync**来发送命令缓冲区，并在异步操作完成后返回**ModuleCommandResult**。如果命令执行成功，则首先使用[**DataReader**](https://docs.microsoft.com/uwp/api/windows.storage.streams.datareader)读取从模块返回的整数状态值。如果此值为供应商定义的成功值，则其余结果数据将由应用读取并使用，例如更新UI。

public const byte Contoso\_ReverbLevel\_Command = 30;

public const byte Contoso\_SendCommand\_Success = 99;

var endpointId = MediaDevice.GetDefaultAudioRenderId(AudioDeviceRole.Default);

var audioModuleManager = new AudioDeviceModulesManager(endpointId);

var modules = audioModuleManager.FindAllById(Contoso\_AudioDeviceModuleId);

foreach (var module in modules)

{

var writer = new Windows.Storage.Streams.DataWriter();

writer.WriteByte(Contoso\_ReverbLevel\_Command);

writer.WriteByte(100);

var command = writer.DetachBuffer();

var result = await module.SendCommandAsync(command);

if (result.Status == SendCommandStatus.Success)

{

using (DataReader reader = DataReader.FromBuffer(result.Result))

{

int bufferStatus = reader.ReadInt32();

if (bufferStatus == Contoso\_SendCommand\_Success)

{

byte[] data = { 0, 0 };

reader.ReadBytes(data);

// Do something with returned data, such as update UI

}

}

}

}

## 2.4. 修改音频设备模块时接收通知

通过注册[**ModuleNotificationReceived**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodulesmanager.modulenotificationreceived)事件，音频设备模块已更新时，应用程序可以接收通知。

var endpointId = MediaDevice.GetDefaultAudioRenderId(AudioDeviceRole.Default);

var audioModuleManager = new AudioDeviceModulesManager(endpointId);

audioModuleManager.ModuleNotificationReceived += AudioModuleManager\_ModuleNotificationReceived;

修改与当前音频设备关联的任何音频设备模块时，将引发**ModuleNotificationReceived**。若要确定事件是否与特定模块相关联，请通过访问传递到事件处理程序中的[**AudioDeviceModuleNoticiationEventArgs**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodulenotificationeventargs)的[**Module**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodulenotificationeventargs.module)属性，然后检查标识该模块的[**ClassId**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodule.classid)属性，获取**AudioDeviceModule**的实例。与事件关联的数据作为存储在[**NotificationData中**](https://docs.microsoft.com/uwp/api/windows.media.devices.audiodevicemodulenotificationeventargs.notificationdata)的字节数组传递事件args的属性。与命令和结果一样，返回的字节数组的格式由供应商定义。在下面的示例中，如果通知数据的第一个字节包含模块混响级别设置的示例值，则将读取该数据并将其用于更新UI。

public const byte Contoso\_ReverbLevel\_Data = 25;

private void AudioModuleManager\_ModuleNotificationReceived(AudioDeviceModulesManager sender, AudioDeviceModuleNotificationEventArgs args)

{

if (args.Module.ClassId == Contoso\_AudioDeviceModuleId)

{

// Get the coefficient data from the reverb module.

using (DataReader reader = DataReader.FromBuffer(args.NotificationData))

{

// read notification data.

byte item = reader.ReadByte();

// if reverb coefficient data are changed.

if (item == Contoso\_ReverbLevel\_Data)

{

// read the new value

byte[] data = { 0 };

reader.ReadBytes(data);

ReverbLevelSlider.Value = data[0];

}

}

}

}

# 3. 低延迟音频

本主题讨论Windows 10中的音频延迟改进。它涵盖了应用程序开发人员的API选项以及可以进行支持低延迟音频的驱动程序更改。

## 3.1. 总览

音频延迟是指从创建声音到听到声音之间的延迟。对于以下几个关键场景，低音频延迟非常重要。

* 专业音频
* 音乐创作
* 通讯技术
* 虚拟现实
* 游戏类

Windows 10做了一些改进以减少音频等待时间。本文档的目标是：

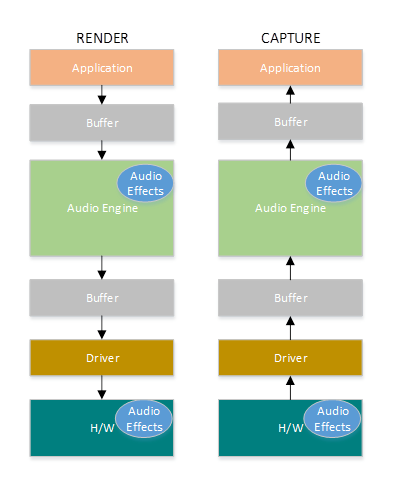
1. 描述Windows中音频延迟的来源。
2. 说明减少Windows 10音频堆栈中音频延迟的改进。
3. 为应用程序开发人员和硬件制造商如何利用新的基础结构提供参考，以开发具有低音频延迟的应用程序和驱动程序。本主题涵盖以下项目：
   1. 用于交互和媒体创建方案的新[**AudioGraph**](https://docs.microsoft.com/uwp/api/Windows.Media.Audio.AudioGraph) API。
   2. WASAPI中的支持低延迟的改进。
   3. 驱动程序DDI中功能的增强。

## 3.2. 定义

|  |  |
| --- | --- |
| 术语 | 描述 |
| 渲染延迟 | 从应用程序向渲染API提交音频数据缓冲区,到从扬声器听到音频数据之间的延迟。 |
| 捕获延迟 | 从麦克风捕获声音,到将声音发送到应用程序正在使用的捕获API的时间之间的延迟。 |
| 往返延迟 | 从麦克风捕获声音，由应用程序处理并由应用程序提交以渲染到扬声器之间的延迟。它大致等于渲染延迟+捕获延迟。 |
| 触摸到应用程序延迟 | 从用户点击屏幕, 到信号发送到应用程序之间的时延。 |
| 触摸到声音的延迟 | 用户敲击屏幕，事件进入应用程序并通过扬声器听到声音之间的时间延迟。它等于渲染延迟+触摸应用延迟。 |

## 3.3. Windows音频堆栈

下图显示了Windows音频堆栈的简化版本。



这是渲染路径中延迟的摘要：

1. 应用程序将数据写入缓冲区
2. 音频引擎从缓冲区读取数据并进行处理。它还以音频处理对象（APO）的形式加载音频效果。有关APO的更多信息，请参见[Windows音频处理对象](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/windows-audio-processing-objects)。
3. APO的等待时间根据APO中的信号处理而有所不同。
4. 在Windows 10之前，使用浮点数据的应用程序的音频引擎的延迟等于〜12ms，对于使用整数数据的应用程序的音频引擎的延迟等于〜6ms
5. 在Windows 10中，所有应用程序的等待时间已减少至1.3ms
6. 音频引擎将处理后的数据写入缓冲区。
7. 在Windows 10之前，此缓冲区始终设置为〜10ms。
8. 从Windows 10开始，缓冲区大小由音频驱动程序定义（有关更多详细信息，将在本主题后面介绍）。
9. 音频驱动程序从缓冲区读取数据并将其写入硬件。
10. 硬件还可以选择再次处理数据（以其他音频效果的形式）。
11. 用户听到扬声器的音频。

以下是捕获路径中的延迟摘要：

1. 从麦克风捕获音频。
2. 硬件可以选择处理数据（即添加音频效果）。
3. 驱动程序从硬件读取数据并将数据写入缓冲区。
4. 在Windows 10之前，此缓冲区始终设置为10ms。
5. 从Windows 10开始，缓冲区大小由音频驱动程序定义（有关此内容的更多详细信息，请参见下文）。
6. 音频引擎从缓冲区读取数据并进行处理。它同样加载音频处理对象（APO）形式的音频效果。
7. APO的等待时间根据APO中的信号处理而有所不同。
8. 在Windows 10之前，对于使用浮点数据的应用程序，音频引擎的延迟等于〜6ms，对于使用整数数据的应用程序，音频引擎的延迟等于〜0ms。
9. 在Windows 10中，所有应用程序的等待时间已减少到〜0ms。
10. 音频引擎完成处理后，会向应用程序发出信号，通知可以读取数据。

音频堆栈还提供了“独占模式”选项。在这种情况下，数据将绕过音频引擎，并直接从应用程序进入缓冲区，驱动程序从中读取数据。但是，如果应用程序以“独占模式”打开端点，则没有其他应用程序可以使用该端点来呈现或捕获音频。

对于需要低延迟的应用程序，另一种流行的替代方法是使用ASIO（音频流输入/输出）模型，该模型利用独占模式。用户安装了第三方ASIO驱动程序后，应用程序可以将数据直接从应用程序发送到ASIO驱动程序。但是，应用程序的编写方式必须使其直接与ASIO驱动程序对话。

两种选择（独占模式和ASIO）都有其自身的局限性。它们提供低延迟，但是它们有其自身的局限性（其中一些如上所述）。结果，对音频引擎进行了修改，以降低延迟，同时又保持了灵活性。

## 3.4. Windows 10中的音频堆栈改进

Windows 10在三个方面进行了增强，以减少延迟：

1. 与Windows 8.1相比，所有使用音频的应用程序的往返延迟都将减少4.5-16ms（如上节所述），而无需任何代码更改或驱动程序更新。a.使用浮点数据的应用程序的延迟将降低16毫秒。b.使用整数数据的应用程序的等待时间将缩短4.5毫秒。
2. 系统的驱动程序升级后将提供更低的往返延迟：a.驱动程序可以使用新的DDI来报告缓冲区的支持大小，该缓冲区用于在OS和H/W之间传输数据。这意味着数据传输不必总是使用10ms缓冲区（就像以前的OS版本一样）。驱动程序可以指定是否可以使用较小的缓冲区，例如5ms，3ms，1ms等。b.要求低延迟的应用程序可以使用新的音频API（AudioGraph或WASAPI），以便查询驱动程序支持的缓冲区大小，并选择将用于与H/W进行数据传输的缓冲区。
3. 当应用程序使用低于特定阈值的缓冲区大小来渲染和捕获音频时，操作系统进入一种特殊模式，在该模式下，操作系统将以避免音频流与其他子系统之间干扰的方式来管理其资源。这将减少音频子系统执行过程中的中断，并最大程度地减小音频毛刺的可能性。当应用程序停止流式传输时，操作系统将返回其正常执行模式。音频子系统包含以下资源：a.正在处理低延迟音频的音频引擎线程。

b.驱动程序已注册的所有线程和中断（使用有关驱动程序资源注册的部分中描述的新DDI）。c.来自要求小缓冲区的应用程序以及任何与此应用程序共享相同音频设备图（例如，相同信号处理模式）的所有应用程序的部分或全部音频线程：

* 1. 流路径上的AudioGraph回调。
  2. 如果应用程序使用WASAPI，则只有工作项, 会被提交到[实时工作队列API](https://docs.microsoft.com/windows/desktop/ProcThread/platform-work-queue-api)或[**MFCreateMFByteStreamOnStreamEx**](https://docs.microsoft.com/windows/desktop/api/mfidl/nf-mfidl-mfcreatemfbytestreamonstreamex)并被标记为“Audio”或“ ProAudio”。

## 3.5. API改进

以下两类Windows 10 API提供了低延迟功能：

* [**音频图**](https://docs.microsoft.com/uwp/api/Windows.Media.Audio.AudioGraph)
* [Windows音频会话API（WASAPI）](https://docs.microsoft.com/windows/desktop/CoreAudio/wasapi)

应用程序开发人员可以通过以下方式确定要使用两个API中的哪个：

* 尽可能支持AudioGraph，以进行新的应用程序开发。
* 仅在以下情况下使用WASAPI：
  + 您需要除AudioGraph提供的控制之外的其他控制。
  + 您需要比AudioGraph提供的延迟更低的延迟。

后续的测量工具部分妙手了使用内置HDAudio驱动程序从Haswell系统(译注:处理器架构)进行的特定测量。

以下各节将说明每个API的低延迟功能。如前一节所述，为了使系统达到最小延迟，它需要具有支持较小缓冲区大小的升级过的驱动程序。

### 音频图

AudioGraph是Windows 10中新的通用Windows平台API，旨在轻松实现交互式和音乐创建方案。AudioGraph具有多种编程语言（C ++，C＃，JavaScript），并且具有简单且功能丰富的编程模型。

为了针对低延迟情况，AudioGraph提供了[AudioGraphSettings :: QuantumSizeSelectionMode属性](https://docs.microsoft.com/uwp/api/Windows.Media.Audio.AudioGraphSettings#Windows_Media_Audio_AudioGraphSettings_QuantumSizeSelectionMode)。该属性可以具有下表中显示的以下任何值：

|  |  |
| --- | --- |
| 值 | 描述 |
| 系统默认 | 将缓冲区设置为默认缓冲区大小（〜10ms） |
| 最低延迟 | 将缓冲区设置为驱动程序支持的最小值 |
| 最近需求 | 将缓冲区大小设置为等于DesiredSamplesPerQuantum属性定义的值，或者等于驱动程序支持的与DesiredSamplesPerQuantum接近的值。 |

AudioCreation示例（可从GitHub上下载：[https](https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/AudioCreation) : [//github.com/Microsoft/Windows-universal-samples/tree/master/Samples/AudioCreation](https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/AudioCreation)）显示了如何使用AudioGraph降低延迟。以下代码片段显示了如何设置最小缓冲区大小：

AudioGraphSettings settings = new AudioGraphSettings(AudioRenderCategory.Media);

settings.QuantumSizeSelectionMode = QuantumSizeSelectionMode.LowestLatency;

CreateAudioGraphResult result = await AudioGraph.CreateAsync(settings);

### Windows音频会话API（WASAPI）

从Windows 10开始，WASAPI已增强为：

* 允许应用程序发现给定音频设备的音频驱动程序支持的缓冲区大小（即周期性值）的范围。在共享模式下打开流时，这使应用程序可以在默认缓冲区大小（10ms）或较小缓冲区（<10ms）之间进行选择。如果应用程序未指定缓冲区大小，则它将使用默认缓冲区大小。
* 允许应用程序发现音频引擎的当前格式和周期性。这使应用程序可以捕捉到音频引擎的当前设置。
* 允许应用程序指定其希望的格式来渲染/捕获内容，而无需音频引擎进行任何重新采样

以上功能将在所有Windows设备上可用。但是，某些具有足够资源和更新驱动程序的设备将提供比其他设备更好的用户体验。

以上功能由名为[**IAudioClient3**](https://docs.microsoft.com/windows/desktop/api/audioclient/nn-audioclient-iaudioclient3)的新接口提供，该接口派生自[**IAudioClient2**](https://docs.microsoft.com/windows/desktop/api/audioclient/nn-audioclient-iaudioclient2)。

[**IAudioClient3**](https://docs.microsoft.com/windows/desktop/api/audioclient/nn-audioclient-iaudioclient3)定义以下3种方法：

|  |  |
| --- | --- |
| 方法 | 描述 |
| GetCurrentSharedModeEnginePeriod | 返回音频引擎的当前格式和周期性 |
| GetSharedModeEnginePeriod | 返回引擎支持的指定流格式的周期性范围 |
| InitializeSharedAudioStream | 以指定的周期初始化共享流 |

WASAPIAudio示例（在GitHub上提供：[https](https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/WindowsAudioSession)：[//github.com/Microsoft/Windows-universal-samples/tree/master/Samples/WindowsAudioSession](https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/WindowsAudioSession)）展示了如何使用IAudioClient3来降低延迟。

以下代码段显示了音乐创建应用如何在系统支持的最低延迟设置下运行。

// 1. Activation

// Get a string representing the Default Audio (Render|Capture) Device

m\_DeviceIdString = MediaDevice::GetDefaultAudio(Render|Capture)Id(

Windows::Media::Devices::AudioDeviceRole::Default );

// This call must be made on the main UI thread. Async operation will call back to

// IActivateAudioInterfaceCompletionHandler::ActivateCompleted, which must be an agile // interface implementation

hr = ActivateAudioInterfaceAsync( m\_DeviceIdString->Data(), \_\_uuidof(IAudioClient3),

nullptr, this, &asyncOp );

// 2. Setting the audio client properties – note that low latency offload is not supported

AudioClientProperties audioProps = {0};

audioProps.cbSize = sizeof( AudioClientProperties );

audioProps.eCategory = AudioCategory\_Media;

// if the device has System.Devices.AudioDevice.RawProcessingSupported set to true and you want to use raw mode

// audioProps.Options |= AUDCLNT\_STREAMOPTIONS\_RAW;

//

// if it is important to avoid resampling in the audio engine, set this flag

// audioProps.Options |= AUDCLNT\_STREAMOPTIONS\_MATCH\_FORMAT;

hr = m\_AudioClient->SetClientProperties( &audioProps ); if (FAILED(hr)) { ... }

// 3. Querying the legal periods

hr = m\_AudioClient->GetMixFormat( &mixFormat ); if (FAILED(hr)) { ... }

hr = m\_AudioClient->GetSharedModeEnginePeriod(wfx, &defaultPeriodInFrames, &fundamentalPeriodInFrames, &minPeriodInFrames, &maxPeriodInFrames); if (FAILED(hr)) { ... }

// legal periods are any multiple of fundamentalPeriodInFrames between

// minPeriodInFrames and maxPeriodInFrames, inclusive

// the Windows shared-mode engine uses defaultPeriodInFrames unless an audio client // has specifically requested otherwise

// 4. Initializing a low-latency client

hr = m\_AudioClient->InitializeSharedAudioStream(

AUDCLNT\_STREAMFLAGS\_EVENTCALLBACK,

desiredPeriodInFrames,

mixFormat,

nullptr); // audio session GUID

if (AUDCLNT\_E\_ENGINE\_PERIODICITY\_LOCKED == hr) {

/\* engine is already running at a different period; call m\_AudioClient->GetSharedModeEnginePeriod to see what it is \*/

} else if (FAILED(hr)) {

...

}

// 5. Initializing a client with a specific format (if the format needs to be different than the default format)

AudioClientProperties audioProps = {0};

audioProps.cbSize = sizeof( AudioClientProperties );

audioProps.eCategory = AudioCategory\_Media;

audioProps.Options |= AUDCLNT\_STREAMOPTIONS\_MATCH\_FORMAT;

hr = m\_AudioClient->SetClientProperties( &audioProps );

if (FAILED(hr)) { ... }

hr = m\_AudioClient->IsFormatSupported(AUDCLNT\_SHAREMODE\_SHARED, appFormat, &closest);

if (S\_OK == hr) {

/\* device supports the app format \*/

} else if (S\_FALSE == hr) {

/\* device DOES NOT support the app format; closest supported format is in the "closest" output variable \*/

} else {

/\* device DOES NOT support the app format, and Windows could not find a close supported format \*/

}

hr = m\_AudioClient->InitializeSharedAudioStream(

AUDCLNT\_STREAMFLAGS\_EVENTCALLBACK,

defaultPeriodInFrames,

appFormat,

nullptr); // audio session GUID

if (AUDCLNT\_E\_ENGINE\_FORMAT\_LOCKED == hr) {

/\* engine is already running at a different format \*/

} else if (FAILED(hr)) {

...

}

另外，建议使用WASAPI的应用程序也使用[实时工作队列API](https://docs.microsoft.com/windows/desktop/ProcThread/platform-work-queue-api)或[**MFCreateMFByteStreamOnStreamEx**](https://docs.microsoft.com/windows/desktop/api/mfidl/nf-mfidl-mfcreatemfbytestreamonstreamex)来创建工作项并将其标记为Audio或Pro Audio，而不是用其自己的线程来处理。这将允许OS以一种避免干扰非音频子系统的方式来管理它们。相反，操作系统会自动正确管理所有AudioGraph线程。WASAPIAudio示例的以下代码片段显示了如何使用MF工作队列API。

// Specify Source Reader Attributes

Attributes->SetUnknown( MF\_SOURCE\_READER\_ASYNC\_CALLBACK, static\_cast<IMFSourceReaderCallback \*>(this) );

if (FAILED( hr ))

{

goto exit;

}

Attributes->SetString( MF\_READWRITE\_MMCSS\_CLASS\_AUDIO, L"Audio" );

if (FAILED( hr ))

{

goto exit;

}

Attributes->SetUINT32( MF\_READWRITE\_MMCSS\_PRIORITY\_AUDIO, 0 );

if (FAILED( hr ))

{

goto exit;

}

// Create a stream from IRandomAccessStream

hr = MFCreateMFByteStreamOnStreamEx (reinterpret\_cast<IUnknown\*>(m\_ContentStream), &ByteStream );

if ( FAILED( hr ) )

{

goto exit;

}

// Create source reader

hr = MFCreateSourceReaderFromByteStream( ByteStream, Attributes, &m\_MFSourceReader );

另外，以下代码片段显示了如何使用RT Work Queue API。

#define INVALID\_WORK\_QUEUE\_ID 0xffffffff

DWORD g\_WorkQueueId = INVALID\_WORK\_QUEUE\_ID;

//#define MMCSS\_AUDIO\_CLASS L"Audio"

//#define MMCSS\_PROAUDIO\_CLASS L"ProAudio"

STDMETHODIMP TestClass::GetParameters(DWORD\* pdwFlags, DWORD\* pdwQueue)

{

HRESULT hr = S\_OK;

\*pdwFlags = 0;

\*pdwQueue = g\_WorkQueueId;

return hr;

}

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

STDMETHODIMP TestClass::Invoke(IRtwqAsyncResult\* pAsyncResult)

{

HRESULT hr = S\_OK;

IUnknown \*pState = NULL;

WCHAR className[20];

DWORD bufferLength = 20;

DWORD taskID = 0;

LONG priority = 0;

printf("Callback is invoked pAsyncResult(0x%0x) Current process id :0x%0x Current thread id :0x%0x\n", (INT64)pAsyncResult, GetCurrentProcessId(), GetCurrentThreadId());

hr = RtwqGetWorkQueueMMCSSClass(g\_WorkQueueId, className, &bufferLength);

IF\_FAIL\_EXIT(hr, Exit);

if (className[0])

{

hr = RtwqGetWorkQueueMMCSSTaskId(g\_WorkQueueId, &taskID);

IF\_FAIL\_EXIT(hr, Exit);

hr = RtwqGetWorkQueueMMCSSPriority(g\_WorkQueueId, &priority);

IF\_FAIL\_EXIT(hr, Exit);

printf("MMCSS: [%ws] taskID (%d) priority(%d)\n", className, taskID, priority);

}

else

{

printf("non-MMCSS\n");

}

hr = pAsyncResult->GetState(&pState);

IF\_FAIL\_EXIT(hr, Exit);

Exit:

return S\_OK;

}

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

int \_tmain(int argc, \_TCHAR\* argv[])

{

HRESULT hr = S\_OK;

HANDLE signalEvent;

LONG Priority = 1;

IRtwqAsyncResult \*pAsyncResult = NULL;

RTWQWORKITEM\_KEY workItemKey = NULL;;

IRtwqAsyncCallback \*callback = NULL;

IUnknown \*appObject = NULL;

IUnknown \*appState = NULL;

DWORD taskId = 0;

TestClass cbClass;

NTSTATUS status;

hr = RtwqStartup();

IF\_FAIL\_EXIT(hr, Exit);

signalEvent = CreateEvent(NULL, true, FALSE, NULL);

IF\_TRUE\_ACTION\_EXIT(signalEvent == NULL, hr = E\_OUTOFMEMORY, Exit);

g\_WorkQueueId = RTWQ\_MULTITHREADED\_WORKQUEUE;

hr = RtwqLockSharedWorkQueue(L"Audio", 0, &taskId, &g\_WorkQueueId);

IF\_FAIL\_EXIT(hr, Exit);

hr = RtwqCreateAsyncResult(NULL, reinterpret\_cast<IRtwqAsyncCallback\*>(&cbClass), NULL, &pAsyncResult);

IF\_FAIL\_EXIT(hr, Exit);

hr = RtwqPutWaitingWorkItem(signalEvent, Priority, pAsyncResult, &workItemKey);

IF\_FAIL\_EXIT(hr, Exit);

for (int i = 0; i < 5; i++)

{

SetEvent(signalEvent);

Sleep(30);

hr = RtwqPutWaitingWorkItem(signalEvent, Priority, pAsyncResult, &workItemKey);

IF\_FAIL\_EXIT(hr, Exit);

}

Exit:

if (pAsyncResult)

{

pAsyncResult->Release();

}

if (INVALID\_WORK\_QUEUE\_ID != g\_WorkQueueId)

{

hr = RtwqUnlockWorkQueue(g\_WorkQueueId);

if (FAILED(hr))

{

printf("Failed with RtwqUnlockWorkQueue 0x%x\n", hr);

}

hr = RtwqShutdown();

if (FAILED(hr))

{

printf("Failed with RtwqShutdown 0x%x\n", hr);

}

}

if (FAILED(hr))

{

printf("Failed with error code 0x%x\n", hr);

}

return 0;

}

最后，使用WASAPI的应用程序开发人员需要根据每个流的功能，用音频类别标记其流，以及是否使用原始信号处理模式。建议所有音频流不要使用原始信号处理模式，除非可以理解其中的含义。原始模式会绕过OEM选择的所有信号处理，因此：

* 特定端点的渲染信号可能不是最佳的。
* 捕获信号可能以应用程序无法理解的格式出现。
* 延迟可能会得到改善。

## 3.6. 驱动程序改进

为了使音频驱动程序支持低延迟，Windows 10提供了以下3个新功能：

1. [必须] 声明每种模式支持的最小缓冲区大小。
2. [可选，但推荐] 改善驱动程序和OS之间的数据流协调。
3. [可选，但建议使用] 注册驱动程序资源（中断，线程），以便在低延迟情况下可由OS保护它们。内置的HDAudio总线驱动程序hdaudbus.sys枚举的HDAudio微型端口功能驱动程序不需要注册HDAudio中断，因为hdaudbus.sys已经完成了此操作。但是，如果微型端口驱动程序创建了自己的线程，则需要注册它们。

以下三个部分将更深入地说明每个新功能。

### 声明最小缓冲区大小

在OS，驱动程序和硬件之间移动音频数据时，驱动程序会在各种约束下运行。这些约束可能是由于在存储器和硬件之间移动数据的物理硬件传输，和/或由于硬件或关联的DSP中的信号处理模块。

在Windows 10中，驱动程序可以使用DEVPKEY\_KsAudio\_PacketSize\_Constraints设备属性来表示其缓冲区大小功能。此属性允许用户定义驱动程序支持的绝对最小缓冲区大小，以及每种信号处理模式的特定缓冲区大小约束（特定于模式的约束必须大于驱动程序的最小缓冲区大小，否则被音频堆栈忽略）。例如，以下代码片段显示了驱动程序如何声明支持的绝对最小缓冲区大小为1ms，但是默认模式支持128帧（如果我们假设为48 kHz采样率，则对应于3 ms）。

// Describe constraints for small buffers

static struct

{

KSAUDIO\_PACKETSIZE\_CONSTRAINTS TransportPacketConstraints;

KSAUDIO\_PACKETSIZE\_PROCESSINGMODE\_CONSTRAINT AdditionalProcessingConstraints[1];

} SysvadWaveRtPacketSizeConstraintsRender =

{

{

1 \* HNSTIME\_PER\_MILLISECOND, // 1 ms minimum processing interval

FILE\_256\_BYTE\_ALIGNMENT, // 256 byte packet size alignment

0, // reserved

1, // 1 processing constraint below

{

STATIC\_AUDIO\_SIGNALPROCESSINGMODE\_DEFAULT, // constraint for default processing mode

128, // 128 samples per processing frame

0, // N/A hns per processing frame

},

},

};

请参阅以下主题，以获取有关这些结构的更深入的信息：

* [**KSAUDIO\_PACKETSIZE\_CONSTRAINTS结构**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudio_packetsize_constraints)
* [**KSAUDIO\_PACKETSIZE\_PROCESSINGMODE\_CONSTRAINT结构**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudio_packetsize_signalprocessingmode_constraint)

此外，sysvad示例（<https://github.com/Microsoft/Windows-driver-samples/tree/master/audio/sysvad>）显示了如何使用这些属性，以便驱动程序声明每种模式的最小缓冲区。

### 改善驱动程序和操作系统之间的协调。

本节中描述的DDI使驱动程序能够：

* 明确指出缓冲区的哪一半（包）可用于操作系统，而不是让操作系统根据编解码器链接位置猜测。这有助于操作系统更快地从音频卡顿(glitch)中恢复。
* 可以选择优化或简化其在WaveRT缓冲区中的数据传输。收益的大小取决于WaveRT缓冲区与（可能是DSP）硬件之间的DMA引擎设计或其他数据传输机制。
* 如果驱动程序在内部累积了捕获的数据，则“突发”捕获的数据将比实时更快。这主要用于语音激活方案，但也可以在正常流传输中应用。
* 提供有关其当前流位置的时间戳信息，而不是提供操作系统猜测的信息，从而有可能获得极其准确的位置信息。

在使用DSP的情况下，此DDI非常有用。但是，标准的HD音频驱动程序或其他简单的循环DMA缓冲区设计可能不会在此处列出的这些新DDI中带来太多好处。

* [IMiniportWaveRTInputStream](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iminiportwavertinputstream)
* [IMiniportWaveRTOutputStream](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iminiportwavertoutputstream)

一些驱动程序例程(routines)返回Windows性能计数器时间戳，以反映设备捕获或渲染样本的时间。

在具有复杂DSP流水线和信号处理的设备中，计算准确的时间戳可能具有挑战性，应谨慎进行。时间戳不应仅反映样本从OS传输到DSP或从OS转移到DSP的时间。

为了计算性能计数器值，驱动器和DSP可能采用以下一些方法。

* 在DSP内，使用一些内部DSP时钟跟踪采样时间戳。
* 在驱动程序和DSP之间，计算Windows性能计数器和DSP内部时钟之间的相关性。此过程的范围可以从非常简单（但不太精确）到相当复杂或新颖（但更精确）。
* 除非有其他原因，否则请考虑由于信号处理算法，管线或硬件传输而导致的任何恒定延迟。

sysvad示例（<https://github.com/Microsoft/Windows-driver-samples/tree/master/audio/sysvad>）显示了如何使用上述DDI。

### 注册驱动程序资源

为了确保无故障运行，音频驱动程序必须在portcls中注册其流资源。这允许OS管理资源，以避免音频流和其他子系统之间的干扰。

流资源是音频驱动程序用来处理音频流或确保音频数据流的任何资源。目前，仅支持两种类型的流资源：中断和驱动程序拥有的线程。音频驱动程序应在创建资源后注册资源，并在删除资源之前先注销资源。

音频驱动程序可以在加载驱动程序时的初始化时或运行时（例如，在I/O资源重新平衡时）注册资源。Portcls使用全局状态来跟踪所有音频流资源。

在某些用例中，例如需要极低延迟音频的用例，操作系统试图将音频驱动程序的注册资源与其他操作系统，应用程序和硬件活动的干扰隔离开来。操作系统和音频子系统无需与音频驱动程序交互即可按需执行此操作，除了音频驱动程序的资源注册。

注册流资源的要求意味着流管道路径中的所有驱动程序必须直接或间接向Portcls注册其资源。音频微型端口驱动程序具有以下选项：

* 音频微型端口驱动程序是其堆栈的底部驱动程序（直接与h / w接口），在这种情况下，该驱动程序知道其流资源，并且可以向Portcls注册它们。
* 音频微型端口驱动程序正在其他驱动程序（例如音频总线驱动程序）的帮助下传输音频。这些其他驱动程序还使用必须向Portcls注册的资源。这些并行/总线驱动程序堆栈可以公开的公共接口（或私有接口，如果单个供应商拥有所有驱动程序）,以便音频微型端口驱动程序可以通过这个接口用来收集此信息。
* 音频微型端口驱动程序正在其他驱动程序（例如hdaudbus）的帮助下传输音频。这些其他驱动程序还使用(必须是向Portcls注册的) 资源。这些并行/总线驱动程序可以与Portcls链接并直接注册其资源。请注意，音频微型端口驱动程序必须让Portcls知道它们依赖于这些其他并行/总线设备（PDO）的资源。高清音频基础结构使用此选项，即高清音频总线驱动程序与Portcls链接并自动执行以下步骤：
  + 注册其总线驱动程序的资源，并
  + 通知Portcls子资源依赖于父资源。在HD音频体系结构中，音频微型端口驱动程序仅需要注册其自己的驱动程序拥有的线程资源。

注意事项：

* 内置HDAudio总线驱动程序hdaudbus.sys枚举的HDAudio微型端口功能驱动程序不需要注册HDAudio中断，因为hdaudbus.sys已经完成了此操作。但是，如果微型端口驱动程序创建了自己的线程，则需要注册它们。
* 仅为了注册流媒体资源而与Portcls链接的驱动程序必须更新其INF，以包含/需要wdmaudio.inf并复制portcls.sys（和相关文件）。在wdmaudio.inf中定义了一个新的INF复制部分，以仅复制那些文件。
* 仅在Windows 10中运行的音频驱动程序可以硬链接到：
  + [**PcAddStreamResource**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-pcaddstreamresource)
  + [**PcRemoveStreamResource**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-pcremovestreamresource)
* 必须在下层OS上运行的音频驱动程序可以使用以下接口（微型端口可以为IID\_IPortClsStreamResourceManager接口调用QueryInterface并仅在PortCls支持该接口时注册其资源）。
  + [IPortClsStreamResourceManager](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iportclsstreamresourcemanager)
    - [**AddStreamResource**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclsstreamresourcemanager-addstreamresource)
    - [**RemoveStreamResource**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclsstreamresourcemanager-removestreamresource)
* 这些DDI使用以下枚举和结构：
  + [**PcStreamResourceType**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ne-portcls-_pcstreamresourcetype)
  + [**PCSTREAMRESOURCE\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ns-portcls-_pcstreamresource_descriptor)

最后，仅为了注册资源而链接PortCls的驱动程序必须在其inf的DDInstall部分中添加以下两行inf。音频微型端口驱动程序不需要这些inf，因为它们在wdmaudio.inf中已经包含/需要。

[<install-section-name>]

Include=wdmaudio.inf

Needs=WDMPORTCLS.CopyFilesOnly

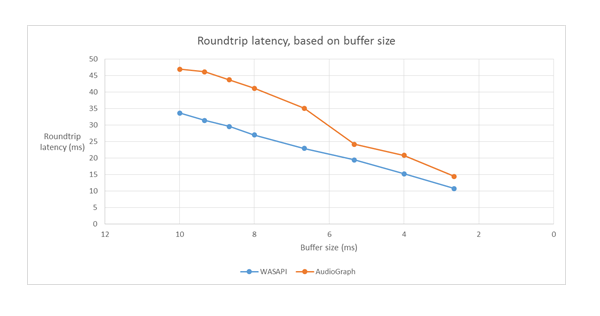
上面的几行inf确保已安装PortCls及其相关文件。

## 3.7. 测量工具

为了测量往返延迟，用户可以使用通过扬声器播放脉冲并通过麦克风捕获脉冲的工具。它们测量以下路径的延迟：

1. 该应用程序调用渲染API（AudioGraph或WASAPI）来播放脉冲
2. 通过扬声器播放音频
3. 从麦克风捕获音频
4. 捕获API（AudioGraph或WASAPI）检测到脉冲。为了测量不同缓冲区大小的往返延迟，用户需要安装支持小缓冲区的驱动程序。内置HDAudio驱动程序已更新，以支持128个样本（2.66ms@48kHz）和480个样本（10ms @ 48kHz）之间的缓冲区大小。以下步骤显示了如何安装内置HDAudio驱动程序（它是所有Windows 10 版本(SKU)的一部分）：

* 启动设备管理器。
* 在“ **声音视频和游戏控制器”下**，双击与您的内置扬声器相对应的设备。
* 在下一个窗口中，转到“ **驱动程序”**选项卡。
* 选择**更新驱动程序** ->在**计算机上浏览驱动程序软件** -> **让我从此计算机上的设备驱动程序列表中进行选择** -> **选择高清晰度音频设备**，然后单击**下一步**。
* 如果出现标题为“更新驱动程序警告”的窗口，请单击“ **是”**。
* 选择**关闭**。
* 如果要求您重新引导系统，请选择“ **是”**重新引导。
* 重新启动后，系统将使用内置Microsoft HDAudio驱动程序，而不是第三方编解码器驱动程序。记住以前使用的是哪个驱动程序，以便在要为音频编解码器使用最佳设置时回退到该驱动程序。



WASAPI和AudioGraph之间的延迟差异是由于以下原因造成的：

* AudioGraph在捕获端添加了一个延迟缓冲，以便同步渲染和捕获（WASAPI不提供）。此附加功能简化了使用AudioGraph编写的应用程序的代码。
* 当系统使用> 6ms的缓冲区时，AudioGraph的渲染端还有一个额外的延迟缓冲区。
* AudioGraph没有选项来禁用捕获音频效果

## 3.8. 示例程序

* WASAPI音频示例: <https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/WindowsAudioSession>
* AudioCreation示例（AudioGraph）: <https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/AudioCreation>
* Sysvad驱动程序样本：<https://github.com/Microsoft/Windows-driver-samples/tree/master/audio/sysvad>

## 3.9. 常问问题

**3.9.1.如果所有应用程序都使用新的API来降低延迟，那会更好吗？低延迟不是总能为用户带来更好的用户体验吗？**

不必要。低延迟有其折衷：

* 低延迟意味着更高的功耗。如果系统使用10ms缓冲区，则意味着CPU将每10ms唤醒一次，填充数据缓冲区并进入睡眠状态。但是，如果系统使用1ms缓冲区，则意味着CPU将每1ms唤醒一次。在第二种情况下，这意味着CPU将更频繁地唤醒，并且功耗会增加。这会缩短电池寿命。
* 大多数应用程序依靠音频效果来提供最佳的用户体验。例如，媒体播放器想要提供高保真音频。而通信应用程序希望将回声和噪声降至最低。将这些类型的音频效果添加到流中会增加其延迟。这些应用程序对音频质量比对音频延迟更感兴趣。

总而言之，每种应用程序类型在音频延迟方面都有不同的需求。如果应用程序不需要低延迟，则不应使用新的API来降低延迟。

**3.9.2.是否会自动更新所有更新到Windows 10的系统以支持小缓冲区？另外，所有系统都支持相同的最小缓冲区大小吗？**

否。为了使系统支持小缓冲区，它需要具有更新的驱动程序。由OEM决定要更新哪些系统以支持小型缓冲区。而且，较新的系统比旧的系统更可能支持较小的缓冲区（即，新系统中的延迟很可能会比旧系统低）。

**3.如果驱动程序支持较小的缓冲区大小（<10ms缓冲区），Windows 10中的所有应用程序是否会自动使用较小的缓冲区来呈现和捕获音频？**

否。默认情况下，Windows 10中的所有应用程序都将使用10ms缓冲区来呈现和捕获音频。如果应用程序需要使用较小的缓冲区，则需要使用新的AudioGraph设置或WASAPI IAudioClient3接口。但是，如果Windows 10中的一个应用程序请求使用小缓冲区，则音频引擎将开始使用该特定缓冲区大小来传输音频。在这种情况下，所有使用相同端点和模式的应用程序将自动切换到该较小的缓冲区大小。当低延迟应用程序退出时，音频引擎将再次切换到10ms缓冲区。

# 4. 语音激活及多语音助手

译注:原文两个章节中有大量重复, 此处做了合并处理.

2013年，Microsoft BUILD开发者大会上首次演示了个人助理技术Cortana。Windows语音平台用于支持Windows 10中的所有语音体验，例如Cortana和Dictation。语音激活是一项功能，使用户可以通过说出特定的短语“ Hey Cortana”来从各种设备电源状态调用语音识别引擎。要创建支持语音激活技术的硬件，请查看本主题中的信息。

**注意:** 实施语音激活是一个重要的项目，并且是SoC供应商完成的任务。OEM可以联系其SoC供应商，以获取有关其SoC语音激活实施的信息。

多重语音助手平台为Cortana之外的其他语音助手提供了支持。这样，其他语音助手就可以在Windows设备（例如PC）和可穿戴设备（例如HoloLens）上使用。使用一组支持的关键字模式，可以在同一设备上激活多个语音助手。

有关实现Windows Cortana的信息，请参阅语音激活章节。

**注意:** 从Windows 10 1903版开始支持多语音助手。

Microsoft提供了操作系统默认关键字检测器（软件关键字检测器），以在无法使用硬件关键字检测的情况下提供语音助手体验。尽管当前可用于Cortana，但可能需要其他Microsoft配置来安装其他语音助手以进行两阶段关键字检测。有关更多信息，请联系AskMVA@Microsoft.com。

## 4.1. Cortana最终用户体验

要了解Windows中可用的语音交互体验，请查看这些主题。

|  |  |
| --- | --- |
| **话题** | **描述** |
| [什么是Cortana？](https://support.microsoft.com/help/17214/cortana-what-is) | 提供Cortana的概述和使用方向 |
| [让Cortana成为您的](https://support.microsoft.com/help/17178/windows-10-make-cortana-yours) | 描述可通过Cortana的“设置”屏幕进行的自定义。 |

## 4.2.“嗨，Cortana”语音激活和“学习我的声音”简介

### “嘿Cortana”语音激活

“ Hey Cortana”语音激活（VA）功能允许用户通过使用Ta的声音在Ta的活动上下文（即当前屏幕上显示的内容）之外快速参与Cortana体验。用户通常希望能够立即访问体验，而无需与设备进行物理交互。对于电话用户而言，这可能是由于正在驾驶并且注意力和手都在操纵车辆上的。对于Xbox用户，这可能是不想查找和连接控制器。对于PC用户，这可能是想无需进行多次鼠标、触摸和/或键盘操作（例如厨房中的计算机）即可快速获得体验。

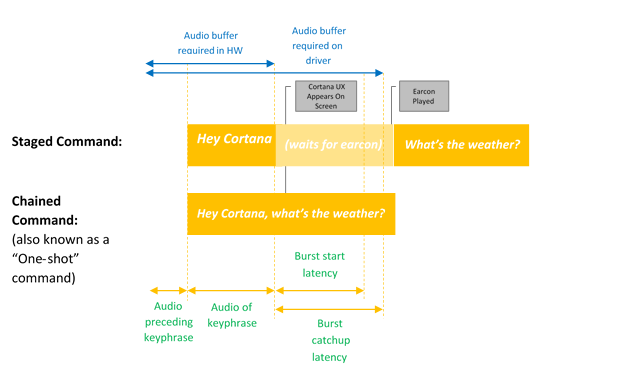
语音激活始终在线的语音监听预定义的关键短语或“激活短语”。关键短语（“Hey Cortana”）可以作为阶段性命令，也可以加上语音操作编程*链接命令*，例如“Hey Cortana，我的下一次会议在哪里？”。

术语“ 关键字检测（Keyword Detection）”，描述了通过硬件或软件对关键字的检测。

仅说Cortana关键字时，仅激活关键字，Cortana启动并播放应答（EarCon）声音以表明它已进入聆听模式。

链接命令中关键词后面紧接着有命令（如“Hey Cortana，打电话给约翰”），会启动Cortana（如果尚未启动），并服从指令操作（打电话给约翰）。

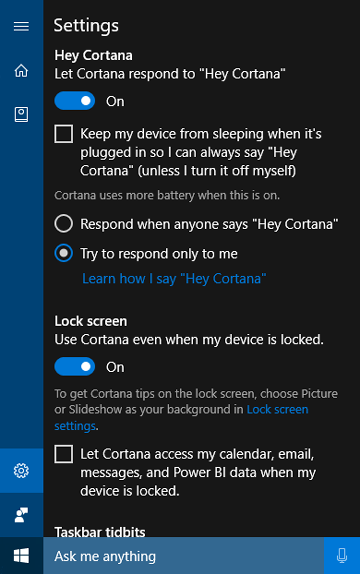
此图说明了链式激活和仅关键字激活。



Microsoft提供了操作系统默认关键字检测器（软件关键字检测器），该检测器用于确保硬件关键字检测的质量并在缺少或不提供硬件关键字检测的情况下提供Hey Cortana体验。

### “学习我的声音”功能

“学习我的声音”功能使用户可以训练Cortana识别其独特的声音。这是通过用户在Cortana设置屏幕中单击“ 了解我如何说“嘿Cortana””来实现的。然后，用户重复六个精心选择的短语，这些短语可提供足够多种语音模式来识别用户语音的独特属性。



当语音激活与“学习我的语音”配对时，这两种算法将一起工作以减少错误的激活。这对于会议室场景特别有价值，在会议室场景中，一个人在装满设备的房间中说“嘿Cortana”。此功能仅适用于Windows 10 1903和更早版本。

语音激活由关键字检测器（KWS）激发，如果检测到关键短语，它就会做出反应。如果KWS要从低功耗状态唤醒设备，则该解决方案称为语音唤醒（WoV）。有关更多信息，请参阅后续语音唤醒章节。

## 4.3. 专业术语

本词汇表总结了与语音激活相关的术语。

|  |  |
| --- | --- |
| 分阶段命令 | 示例：Hey Cortana <暂停，等待应答>天气如何？有时称为“两次命令（Two-shot command）”或“仅关键字（Keyword-only）” |
| 连锁命令 | 例子：Hey，Cortana，天气如何？有时称为“单次命令” |
| 语音激活 | 提供预定义激活关键字短语的关键字检测的场景。例如，“Hey Cortana”是Microsoft语音激活方案。 |
| WoV | 语音唤醒–使语音激活从屏幕关闭、低功耗状态切换到屏幕处于全功耗状态的技术。 |
| Modern Standby 的WoV | 从Modern Standby（S0ix）屏幕关闭状态到全功率（S0）状态屏幕上的语音唤醒。 |
| Modern Standby | Windows低功耗空闲基础结构-Windows 10中的Connected Standby（CS）的后继产品。Modern Standby的第一种状态是屏幕关闭时。最深的睡眠状态是处于DRIPS / Resiliency中。有关更多信息，请参阅[Modern Standby](https://docs.microsoft.com/windows-hardware/design/device-experiences/modern-standby) |
| KWS | 关键字搜寻器（Keyword spotter）–提供“ Hey Cortana”检测的算法 |
| SW KWS | 软件关键字检测器–在主机（CPU）上运行的KWS的实现。对于“ Hey Cortana”，SW KWS是Windows的一部分。 |
| HW KWS | 硬件分担的（Hardware-offloaded）关键字监视程序–在硬件上分担运行的KWS的实现。 |
| 突发（burst）Buffer | 一个环形buffer，用于存储在检测到KWS时可以“bursted up”的PCM数据，以便包括所有触发KWS检测的音频。 |
| 关键字检测器OEM适配器(Keyword Detector OEM Adapter) | 驱动程序级别的组件(shim)，使启用了WoV的硬件能够与Windows和Cortana堆栈进行通信。 |
| 模型 | KWS算法使用的声学模型数据文件。数据文件是静态的。模型已本地化，每个语言环境一个。 |
| MVA | 多个语音代理-支持多个代理的新HWKWS DDI |
| SVA | 单个语音代理-以前的HWKWS DDI，仅支持单个代理（Cortana） |

## 4.4. 集成硬件关键字检测器

要实施硬件关键字检测器（HW KWS），SoC供应商必须完成以下任务。

* 根据本主题后面介绍的SYSVAD示例创建一个自定义关键字检测器。您将在COM DLL中实现这些方法，如后面关键字检测器OEM适配器接口章节中所述。
* 实现后续WAVERT增强章节中描述的WAVE RT增强功能。
* 提供INF文件条目以描述用于关键字检测的任何自定义APO。
  + [PKEY\_FX\_KeywordDetector\_StreamEffectClsid](https://docs.microsoft.com/windows-hardware/drivers/audio/pkey-fx-keyworddetector-streameffectclsid)
  + [PKEY\_FX\_KeywordDetector\_ModeEffectClsid](https://docs.microsoft.com/windows-hardware/drivers/audio/pkey-fx-keyworddetector-modeeffectclsid)
  + [PKEY\_FX\_KeywordDetector\_EndpointEffectClsid](https://docs.microsoft.com/windows-hardware/drivers/audio/pkey-fx-keyworddetector-endpointeffectclsid)
  + [PKEY\_SFX\_KeywordDetector\_ProcessingModes\_Supported\_For\_Streaming](https://docs.microsoft.com/windows-hardware/drivers/audio/pkey-sfx-keyworddetector-processingmodes-supported-for-streaming)
  + [PKEY\_MFX\_KeywordDetector\_ProcessingModes\_Supported\_For\_Streaming](https://docs.microsoft.com/windows-hardware/drivers/audio/pkey-mfx-keyworddetector-processingmodes-supported-for-streaming)
  + [PKEY\_EFX\_KeywordDetector\_ProcessingModes\_Supported\_For\_Streaming](https://docs.microsoft.com/windows-hardware/drivers/audio/pkey-efx-keyworddetector-processingmodes-supported-for-streaming)
* 查看[音频设备建议中](https://docs.microsoft.com/windows-hardware/design/component-guidelines/audio)的硬件建议和测试指南。本主题为旨在与Microsoft语音平台一起使用的音频输入设备的设计和开发提供指导和建议。
* 支持分段命令和链接命令。
* 对于每个受支持的Cortana语言环境，都支持“ Hey Cortana”。
* APO（音频处理对象）必须提供以下效果：
  + AEC
  + 自动增益控制
  + NS
* 语音处理模式的音效必须由MFX APO报告。
* APO可以在MFX中做格式转换 (The APO may perform format conversion as MFX) 。
* APO必须输出以下格式：
  + 16 kHz，单声道，FLOAT。
* （可选）设计任何自定义APO，以增强音频捕获过程。有关更多信息，请参见[Windows音频处理对象](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/windows-audio-processing-objects)。

硬件分担的关键字搜寻器（HW KWS）WoV要求

* 在S0工作状态和S0睡眠状态（也称为Modern Standby）期间，支持HW KWS WoV。
* S3不支持HW KWS WoV。

硬件KWS的AEC要求 (?)

* 对于Windows版本1709
  + 为了在S0睡眠状态（Modern Standby）支持HW KWS WoV，不要求AEC。
  + Windows版本1709不支持S0工作状态下的HW KWS WoV。
* 对于Windows版本1803
  + 支持S0工作状态下的HW KWS WoV。
  + 要在S0工作状态下支持HW KWS WoV，APO必须支持AEC。

**译注:以下两小段来自原文的”多语音助手”章节**

**AEC**

AEC可以在捕获突发音频时由DSP执行，也可以稍后通过软件APO完成。为了使用KWS突发数据执行软件AEC，必须从捕获突发数据的时间起具有相应的回送音频。为此，创建了用于突发输出的自定义音频格式，该格式将回送音频插入突发音频数据中。Microsoft AEC APO知道这种交错格式，可以使用它来执行AEC。有关更多信息，请参见[KSPROPERTY\_INTERLEAVEDAUDIO\_FORMATINFORMATION](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/ksproperty-interleavedaudio-formatinformation)。

**验证方式**

使用HLK测试:[语音激活管理器2](https://docs.microsoft.com/en-us/windows-hardware/test/hlk/testref/5119a80f-8aae-49bb-aa59-8eaa7e7b1fad)(Audio Codec - Voice Activation Manager 2 Tests)来验证对[KSPROPSETID\_SoundDetector2](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kspropsetid-sounddetector2)属性的硬件支持。

## 示例代码概述

作为SYSVAD虚拟音频适配器例子的一部分，有一个音频驱动程序的示例代码，该代码在GitHub上实现了语音激活。建议使用此代码作为起点。该代码在此位置可用。

<https://github.com/Microsoft/Windows-driver-samples/blob/master/audio/sysvad/>

有关SYSVAD示例音频驱动程序的更多信息，请参见[示例音频驱动程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/sample-audio-drivers)。

## 4.5. 关键字识别系统信息

### 语音激活 - 音频堆栈支持

为了使能语音激活, 音频堆栈的外部接口, 被用作语音平台和音频驱动程序的通信管道。外部接口分为三个部分。

* 关键字检测器DDI（Keyword detector Device Driver Interface）。关键字检测器DDI负责配置和提供硬件关键字检测器（KWS）。驱动还使用它来通知系统检测事件。
* 关键字检测器OEM适配器DLL(Keyword Detector OEM Adapter DLL)( 译注: 在2019/09/26更新的多语音助手章节, 此处已经改为[IEvent Detector OEM适配器DLL](https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/eventdetectoroemadapter/nn-eventdetectoroemadapter-ieventdetectoroemadapter))。该DLL实现COM接口，以适应特定于驱动程序的不透明数据，以供OS使用以协助关键字检测。
* WaveRT流增强。增强功能使音频驱动程序可以从关键字检测中突发传输缓冲的音频数据。

### 音频端点属性

音频端点图构建正常进行。该图准备比实时捕获更快地处理。捕获的缓冲区上的时间戳保持真实。具体来说，时间戳将正确反映过去捕获并缓存的数据，现在正在“突发”。(Audio endpoint graph building occurs normally. The graph is prepared to handle faster than real time capture. Timestamps on captured buffers remain true. Specifically, the timestamps will correctly reflect data that was captured in the past and buffered, and is now “bursting”.)

### 操作理论

驱动程序照常为捕获设备公开KS filter。该filter支持多个KS属性和一个KS事件，以配置、启用和发信号通知检测事件。该过滤器还包括一个附加的引脚工厂（pin factory），该工厂被标识为关键字搜索器（KWS）引脚。此引脚用于从关键字检测器流式传输音频。

这些属性是：

* 支持的关键字类型[**-KSPROPERTY\_SOUNDDETECTOR\_PATTERNS**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-sounddetector-patterns)。操作系统设置此属性以配置要检测的关键字。
* 关键字模式GUID的列表[**-KSPROPERTY\_SOUNDDETECTOR\_SUPPORTEDPATTERNS**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-sounddetector-supportedpatterns)。此属性用于获取标识所支持模式类型的GUID列表。
* 提供[**-KSPROPERTY\_SOUNDDETECTOR\_ARMED**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-sounddetector-armed)。此读/写属性是一个简单的布尔状态，指示检测器是否已提供。操作系统对此进行设置以启用关键字检测器。操作系统可以清除此内容以使其停用。设置关键字模式时以及检测到关键字后，驱动程序会自动清除此内容。（操作系统必须重新提供此内容。）
* 匹配结果[**-KSPROPERTY\_SOUNDDETECTOR\_MATCHRESULT**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-sounddetector-matchresult)。在检测发生后，此可读属性保存着结果数据。

当检测到关键字时，触发的事件是[**KSEVENT\_SOUNDDETECTOR\_MATCHDETECTED**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksevent-sounddetector-matchdetected)事件。

## 4.6. 关键字检测器OEM适配器接口

OEM提供了一个COM对象实现，该实现充当OS和驱动程序之间的中介，有助于计算或解析通过[**KSPROPERTY\_SOUNDDETECTOR\_PATTERNS**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-sounddetector-patterns)和[**KSPROPERTY\_SOUNDDETECTOR\_MATCHRESULT**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-sounddetector-matchresult)写入和读取到音频驱动程序的不透明数据。

COM对象的CLSID是[**KSPROPERTY\_SOUNDDETECTOR\_SUPPORTEDPATTERNS**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-sounddetector-supportedpatterns)返回的检测器模式类型GUID 。操作系统调用CoCreateInstance传递模式类型GUID来实例化适当的COM对象，它是与关键字模式类型兼容，并调用该对象的IKeywordDetectorOemAdapter接口中的方法。

### COM线程模型要求

OEM的实现可以选择任何COM线程模型。

### IKeywordDetectorOemAdapter

接口设计试图使对象实现保持无状态。换句话说，该实现不应要求在方法调用之间存储任何状态。实际上，除了一般实现COM对象所需的成员变量外，内部C ++类可能不需要任何成员变量。

### 方法

实现以下方法。

* [**IKeywordDetectorOemAdapter :: BuildArmingPatternData**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/nf-keyworddetectoroemadapter-ikeyworddetectoroemadapter-buildarmingpatterndata)
* [**IKeywordDetectorOemAdapter :: ComputeAndAddUserModelData**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/nf-keyworddetectoroemadapter-ikeyworddetectoroemadapter-computeandaddusermodeldata)
* [**IKeywordDetectorOemAdapter :: GetCapabilities**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/nf-keyworddetectoroemadapter-ikeyworddetectoroemadapter-getcapabilities)
* [**IKeywordDetectorOemAdapter :: ParseDetectionResultData**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/nf-keyworddetectoroemadapter-ikeyworddetectoroemadapter-parsedetectionresultdata)
* [**IKeywordDetectorOemAdapter :: VerifyUserKeyword**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/nf-keyworddetectoroemadapter-ikeyworddetectoroemadapter-verifyuserkeyword)

### 关键字ID

该[**KEYWORDID**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/ne-keyworddetectoroemadapter-__midl_ikeyworddetectoroemadapter_0002)枚举标识关键字的短语文本/功能，在Windows生物识别服务适配器(Windows Biometric Service adapters)也可使用。有关更多信息，请参见[生物识别框架概述-核心平台组件。](https://docs.microsoft.com/windows/desktop/SecBioMet/biometric-framework-overview)

typedef enum {

KwInvalid = 0,

KwHeyCortana = 1,

KwSelect = 2

} KEYWORDID;

### 关键字选择器

该[**KEYWORDSELECTOR**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/ns-keyworddetectoroemadapter-__midl_ikeyworddetectoroemadapter_0003)结构是一组ID, 可以唯一地选择特定的关键字和语言。

typedef struct

{

KEYWORDID KeywordId;

LANGID LangId;

} KEYWORDSELECTOR;

### 处理模型数据

*静态用户独立模型(*Static user independent model*)* -OEM DLL通常会包含一些内置于DLL或DLL随附的单独数据文件中的静态用户独立模型数据。GetCapabilities例程返回的一组受支持的关键字ID将取决于此数据。例如，如果GetCapabilities返回的受支持的关键字ID列表包括KwHeyCortana，则与用户无关的静态模型数据将包括所有受支持语言的“ Hey Cortana”（或其翻译）数据。

*动态依赖用户的模型(*Dynamic user dependent model*)* -IStream提供了随机访问存储模型。操作系统将IStream接口指针传递给IKeywordDetectorOemAdapter接口上的许多方法。操作系统通过适当的存储支持IStream实现，最多可存储1MB的数据。

该存储中数据的内容和结构由OEM定义。预期目标是永久存储由OEM DLL计算或检索的用户相关模型数据。

OS可以使用空的IStream调用接口方法，尤其是在用户从未训练过关键字的情况下。操作系统为每个用户创建一个单独的IStream存储。换句话说，给定的IStream为一个且仅一个用户存储模型数据。

OEM DLL开发人员决定如何管理独立于用户和依赖于用户的数据。但是，它将永远不会在IStream之外的任何地方存储用户数据。一种可能的OEM DLL设计, 根据当前方法的参数在内部访问IStream和访问独立于用户的静态数据之间进行切换。另一种设计可能是在每个方法调用开始时检查IStream，并将静态用户独立数据添加到IStream（如果尚不存在），从而允许该方法的其余部分仅访问所有模型数据的IStream。

## 4.7. 训练和执行音频处理

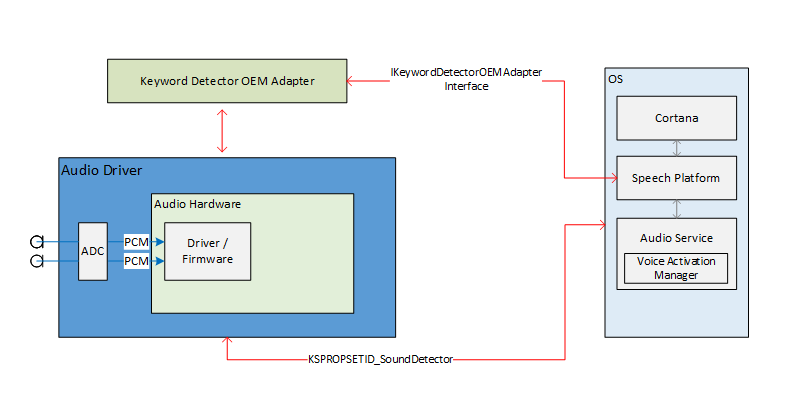
如前所述，训练UI流程会在音频流中提供丰富的音节完整的句子。每个句子分别传递给[**IKeywordDetectorOemAdapter :: VerifyUserKeyword**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/nf-keyworddetectoroemadapter-ikeyworddetectoroemadapter-verifyuserkeyword)以验证它包含预期的关键字并具有可接受的质量。在UI收集并验证了所有句子之后，它们全部通过一次调用传递给[**IKeywordDetectorOemAdapter :: ComputeAndAddUserModelData**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/nf-keyworddetectoroemadapter-ikeyworddetectoroemadapter-computeandaddusermodeldata)。

音频以独特的方式处理，用于语音激活训练。下表总结了语音激活训练与常规语音识别用法之间的区别。

|  |  |  |
| --- | --- | --- |
|  | **语音训练** | **语音识别** |
| **模式** | Raw | Raw or Speech |
| **pin** | 正常 | KWS |
| **音频格式** | 32位浮点型（类型=音频，子类型= IEEE\_FLOAT，采样率= 16 kHz，位= 32） | 由操作系统音频堆栈管理 |
| **麦克风** | 麦克风0 | 阵列中的所有麦克风，或单声道 |

## 4.8. 关键字识别系统概述

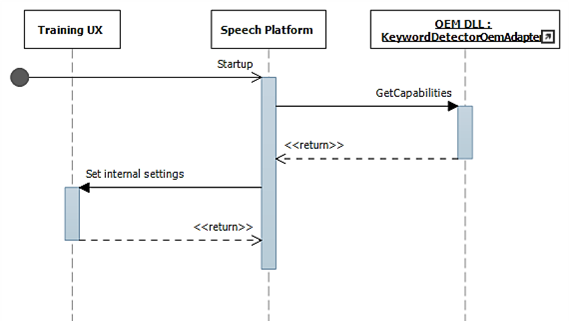
此图概述了关键字识别系统。



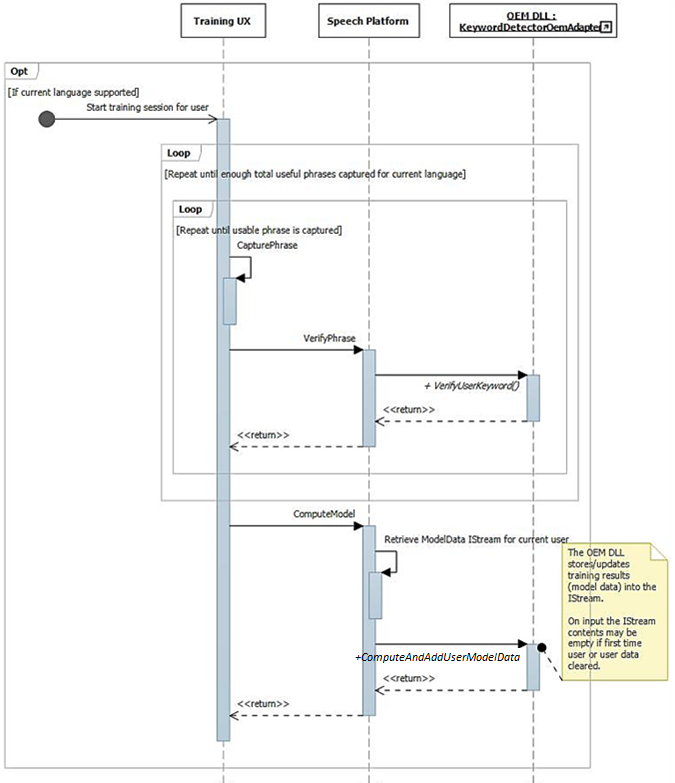
## 4.9. 关键字识别顺序图

在这些图中，语音运行时模块(runtime module)显示为“语音平台”。如前所述，Windows语音平台用于支持Windows 10中的所有语音体验，例如Cortana和听写。

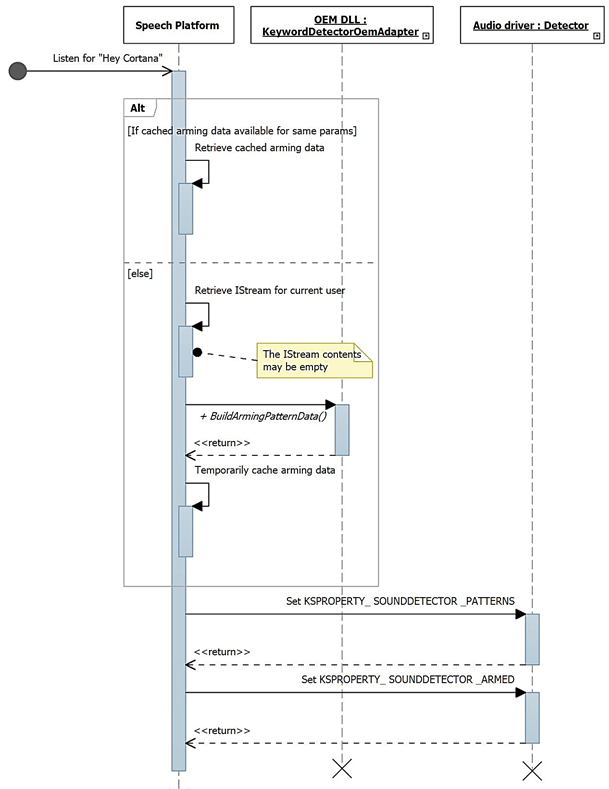
在启动期间，使用[**IKeywordDetectorOemAdapter :: GetCapabilities**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/keyworddetectoroemadapter/nf-keyworddetectoroemadapter-ikeyworddetectoroemadapter-getcapabilities)收集功能。



稍后，当用户选择“学习我的声音”时，将调用训练流程。



此图描述了用于关键字检测的提供过程。



## 4.10. WAVERT增强功能

微型端口接口定义为由WaveRT微型端口驱动程序实现。这些接口提供了方法, 来简化音频驱动程序，提高OS音频管道性能和可靠性, 或支持新方案。定义了一个新的PnP设备接口属性，该属性允许驱动程序向OS提供其缓冲区大小约束的静态表达式。

### 缓冲区大小

在OS，驱动程序和硬件之间移动音频数据时，驱动程序会在各种约束下运行。这些约束可能是由于在存储器和硬件之间移动数据的物理硬件传输，和/或由于硬件或关联的DSP中的信号处理模块。

HW-KWS解决方案必须支持至少100ms至200ms的音频捕获大小。

对于具有KS流引脚的KS filter, 驱动程序通过设置KSCATEGORY\_AUDIO PnP设备接口上的设备属性DEVPKEY\_KsAudio\_PacketSize\_Constraints来表达缓冲区大小约束。启用KS filter接口后，此属性应保持有效和稳定。操作系统可以随时读取此值，而无需打开驱动程序的句柄并调用驱动程序。

### DEVPKEY\_KsAudio\_PacketSize\_Constraints

DEVPKEY\_KsAudio\_PacketSize\_Constraints属性值包含一个描述物理硬件约束的[**KSAUDIO\_PACKETSIZE\_CONSTRAINTS**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudio_packetsize_constraints)结构（即，取决于将数据从WaveRT缓冲区传输到音频硬件的机制）。该结构包括一个0或多个[**KSAUDIO\_PACKETSIZE\_PROCESSINGMODE\_CONSTRAINT**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudio_packetsize_signalprocessingmode_constraint)结构的数组，[**这些**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/ksmedia/ns-ksmedia-_ksaudio_packetsize_signalprocessingmode_constraint)结构描述了特定于任何信号处理模式的约束。驱动程序在调用[**PcRegisterSubdevice**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-pcregistersubdevice)或为流引脚启用KS filter接口之前设置此属性。

### IMiniportWaveRTInputStream

驱动程序应实现此接口，以更好地协调从驱动程序到OS的音频数据流。如果该接口在捕获流上可用，则OS使用该接口上的方法访问WaveRT缓冲区中的数据。有关更多信息，请参见[**IMiniportWaveRTInputStream :: GetReadPacket**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavertinputstream-getreadpacket)

### IMiniportWaveRTOutputStream

WaveRT微型端口可以选择实现此接口，以告知OS写入进度并返回精确的流位置。有关更多信息，请参见[**IMiniportWaveRTOutputStream :: SetWritePacket**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavertoutputstream-setwritepacket)，[**IMiniportWaveRTOutputStream :: GetOutputStreamPresentationPosition**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavertoutputstream-getoutputstreampresentationposition)和[**IMiniportWaveRTOutputStream :: GetPacketCount**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavertoutputstream-getpacketcount)。

### 性能计数器时间戳

一些驱动程序例程返回Windows性能计数器时间戳，以反映设备捕获或呈现样本的时间。

在具有复杂DSP流水线和信号处理的设备中，计算准确的时间戳可能具有挑战性，应谨慎进行。时间戳不应仅反映样本从OS传输到DSP或从OS转移到DSP的时间。

* 在DSP内，使用一些DSP内部时钟跟踪采样时间戳。
* 在驱动程序和DSP之间，计算Windows性能计数器和DSP时钟之间的相关性。此过程的范围可以从非常简单（但不太精确）到相当复杂或新颖（但更精确）。
* 除非有其他原因，否则请考虑由于信号处理算法，管线或硬件传输而导致的任何恒定延迟。

### 突发读取操作(Burst Read Operation)

本节介绍操作系统和驱动程序在突发读取时的交互。只要驱动程序支持基于数据包的流WaveRT模型，包括[**IMiniportWaveRTInputInputStream :: GetReadPacket**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavertinputstream-getreadpacket)函数，突发读取就可以在语音激活方案之外发生。

讨论了两个突发示例读取方案。在一种情况下，如果微型端口支持引脚类别为[**KSNODETYPE\_AUDIO\_KEYWORDDETECTOR**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-audio-keyworddetector)的引脚，则驱动程序将在检测到关键字时开始捕获并在内部缓冲数据。在另一种情况下，如果操作系统无法通过调用[**IMiniportWaveRTInputInput :: GetReadPacket**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavertinputstream-getreadpacket)足够快地读取数据，则驱动程序可以在内部在WaveRT缓冲区之外缓冲数据。

要使在过渡到[KSSTATE\_RUN](https://docs.microsoft.com/en-us/windows-hardware/drivers/stream/streaming-states)之前已捕获的数据突发，驱动程序必须保留准确的采样时间戳信息以及缓冲的捕获数据。时间戳标识捕获的样本的采样时刻。

1. 在流转换为KSSTATE\_RUN之后，驱动程序会立即设置缓冲区通知事件，因为该事件已具有可用数据。
2. 在此事件上，操作系统将调用GetReadPacket（）以获取有关可用数据的信息。

a. 驱动程序返回有效捕获数据的数据包编号（从KSSTATE\_STOP转换为KSSTATE\_RUN后的第一个数据包为0），操作系统可以从中得出WaveRT缓冲区中的数据包位置以及相对于流开始的数据包位置。

b. 驱动程序还返回性能计数器值，该值对应于数据包中第一个样本的采样时刻。请注意，此性能计数器值可能相对较旧，具体取决于在硬件或驱动程序（WaveRT缓冲区之外）中缓冲了多少捕获数据。

c. 如果有更多未读取的缓冲数据可用，驱动程序将：

i. 立即将数据传输到WaveRT缓冲区的可用空间（即，GetReadPacket返回的数据包未使用的空间），为MoreData返回true，并在从此例程返回之前设置缓冲区通知事件。

ii. 或对硬件进行编程，以将下一个数据包突发到WaveRT缓冲区的可用空间中，为MoreData返回false，然后在传输完成时设置缓冲区事件。

1. 操作系统使用GetReadPacket（）返回的信息从WaveRT缓冲区读取数据。
2. 操作系统等待下一个缓冲区通知事件。如果驱动程序在步骤（2c）中设置了缓冲区通知，则等待可能会立即终止。
3. 如果驱动程序未在步骤（2c）中立即设置事件，则驱动程序在将更多捕获的数据传输到WaveRT缓冲区中并使其可供操作系统读取后设置事件。
4. 转到（2）。对于[**KSNODETYPE\_AUDIO\_KEYWORDDETECTOR**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-audio-keyworddetector)关键字检测器引脚，驱动程序应为至少5000ms的音频数据分配足够的内部猝发缓冲。如果操作系统未能在缓冲区溢出之前在引脚上创建流，则驱动程序可能会结束内部缓冲活动并释放关联的资源。

## 4.11. 唤醒声音

唤醒语音（WoV）使用户能够通过说出某个关键字（例如“Hey·Cortana”）来从屏幕关闭的低功耗状态到屏幕打开的全功耗状态激活和查询语音识别引擎。

此功能允许设备在低功耗状态下（包括屏幕关闭和设备空闲时）始终听用户的声音。它通过使用监听模式来做到这一点，与普通麦克风录制过程中看到的更高功耗相比，它的功耗更低。低功耗语音识别功能使用户可以简单地说出“ Hey Cortana”之类的预定义关键字词，然后说出诸如“ when's my nextappointment”之类的链接式语音词组，以免提方式做语音触发。无论设备在使用中还是在屏幕关闭的情况下处于空闲状态，此功能都可以工作。

音频堆栈负责传达唤醒数据（扬声器ID，关键字触发器，置信度），并通知感兴趣的客户端已检测到关键字。

## 4.12. 在Modern Standby系统上的验证

可以这样使用[HLK](https://docs.microsoft.com/windows-hardware/test/hlk/)中的” [插电](https://docs.microsoft.com/en-us/windows-hardware/test/hlk/testref/69df7cf2-6024-4eee-92ee-1506480614ee)和[电池](https://docs.microsoft.com/windows-hardware/test/hlk/testref/614ffb93-eced-45ab-bf7b-e09291a97fd2)情况下的WoV基本测试”来验证从系统空闲状态([Modern Standby](https://docs.microsoft.com/windows-hardware/design/device-experiences/modern-standby))WOV。这些测试检查系统是否具有硬件关键字检测器（HW-KWS），是否能够进入最深睡眠状态（Deepest Runtime Idle Platform State, DRIPS）以及是否能够通过语音命令从Modern Standby中唤醒，并且系统恢复延迟应该小于或等于一秒钟。

# 5. [USB Audio 2.0 驱动](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/usb-2-0-audio-drivers)

从Windows 10版本1703开始，Windows随附USB Audio 2.0驱动程序。它旨在支持USB Audio 2.0设备类。该驱动程序是WaveRT音频端口类微型端口。有关USB Audio 2.0设备类的更多信息，请参见<https://www.usb.org/documents?search=&type%5B0%5D=55&items_per_page=50>。

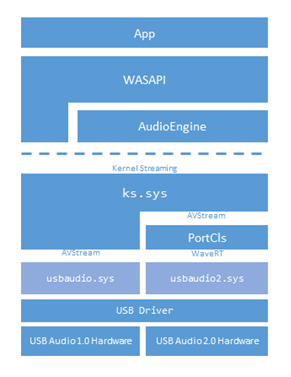
该驱动程序名为：*usbaudio2.sys*，关联的inf文件为*usbaudio2.inf*。

驱动程序将在设备管理器中标识为“ USB Audio Class 2 Device”。如果可用，此名称将被USB产品字符串覆盖。

将兼容设备连接到系统后，将自动启用该驱动程序。但是，如果系统或Windows Update上存在第三方驱动程序，则将安装该驱动程序并覆盖类驱动程序。

## 架构

如图所示，USBAudio.Sys存在于Windows USB Audio的大体系结构中。



## 相关USB规格

以下USB规范定义了USB音频，并且在本主题中进行了引用。

* USB-2指: 通用串行总线规范2.0修订版
* ADC-2是指: 音频设备的USB设备类定义, 2.0版。
* FMT-2指的是音频数据格式规范， 2.0版。

USB-IF是一个特殊的组织，它维护[官方的USB规范](https://www.usb.org/documents)，测试规范和工具。

## 音频格式

该驱动程序支持以下格式。如果指定为FMT-2中定义的另一种格式或未知格式, 该替代设置将被忽略。

I型格式（FMT-2 2.3.1）：

* PCM格式，每个样本具有8..32位（FMT20 2.3.1.7.1）
* PCM8格式（FMT-2 2.3.1.7.2）
* IEEE\_FLOAT格式（FMT-2 2.3.1.7.3）

III型格式（FMT-2 2.3.3和A.2.3）：

* IEC61937\_AC-3
* IEC61937\_MPEG-2\_AAC\_ADTS
* IEC61937\_DTS-I
* IEC61937\_DTS-II
* IEC61937\_DTS-III
* TYPE\_III\_WMA

## 功能说明

本节介绍USB Audio 2.0驱动程序的功能。

### 音频功能拓扑

该驱动程序支持ADC-2 3.13中定义的所有实体类型。

在兼容的USB Audio 2.0硬件中, 每个终端实体必须具有有效的时钟连接。时钟路径可以选择包括时钟乘法器和时钟选择器单元，并且必须以时钟源实体结尾。

该驱动程序仅支持一个时钟源。如果设备实现多个时钟源实体和一个时钟选择器，则驱动程序将使用默认选择的时钟源，并且不会修改时钟选择器的位置。

不支持具有多个输入引脚的处理单元（ADC-2 3.13.9）。

不支持具有多个输入引脚的扩展单元（ADC-2 3.13.10）。

拓扑中不允许存在循环路径。

### 音频流

驱动程序支持以下端点同步类型（USB-2 5.12.4.1）：

* 异步输入和输出
* 同步输入和输出
* 自适应输入和输出

对于异步输出，驱动程序仅支持显式反馈。反馈端点必须在异步接口的相应备用设置中实现。驱动程序不支持隐式反馈。

当前对在多端点设备中共享时钟的支持有限。

对于自适应输入，驱动程序不支持前馈端点。如果替代设置中存在此类端点，则它将被忽略。驱动程序以与异步输入流相同的方式处理自适应输入流。

设备创建的同步数据包的大小必须在FMT-2.0第2.3.1.1节中指定的限制内。这意味着实际数据包大小与标称大小的偏差不得超过+/-1个音频插槽（音频插槽=通道计数样本）。

## 描述符

音频功能必须恰好实现一个AudioControl接口描述符（ADC-2 4.7）和一个或多个AudioStreaming接口描述符（ADC-2 4.9）。不能只有控制接口而没有流接口。

该驱动程序支持ADC20第4节中定义的所有描述符类型。以下小节提供有关某些特定描述符类型的注释。

### 特定于类的AS接口描述符

有关此规范的详细信息，请参见ADC-2 4.9.2。

AS接口描述符必须从可用设置 (alternate setting, 译注: USB的一个术语, 相当于一个配置)零(表示无端点设备)开始（不消耗带宽），并且必须在兼容的USB Audio 2.0硬件中以升序指定其他可用设置。

驱动程序不支持的其他设置格式将被忽略。

每个非零可用设置都必须指定一个同步数据端点，以及一个可选的反馈端点。不支持没有任何端点的非零可用设置。

bTerminalLink字段必须引用拓扑中的终端实体，并且其值在AS接口的所有可用设置中必须相同。

AS接口描述符中的bFormatType字段必须与格式类型描述符（FMT-2 2.3.1.6）中指定的bFormatType相同。

对于类型I格式，必须在AS接口描述符的bmFormats字段中将一个比特位正确设置为1。否则，驱动程序将忽略该格式。

为了节省总线带宽，一个AS接口可以使用相同的格式（根据bNrChannel和AS格式类型描述符）实现多个可用设置，但在同步数据端点描述符中使用不同的wMaxPacketSize值。对于给定的采样率，驱动程序选择可以满足数据速率要求的最小wMaxPacketSize的可用设置。

### Type I格式类型描述符

有关此规范的详细信息，请参阅FMT-2 2.3.1.6。

适用以下限制：

|  |  |  |
| --- | --- | --- |
| 格式名称 | bSubslotSize | bBitResolution |
| I型PCM格式： | 1 ~4 | 8 ~32 |
| I型PCM8格式： | 1 | 8 |
| I型 IEEE\_FLOAT格式： | 4 | 32 |
| III型IEC61937格式： | 2 | 16 |

### 特定类的AS等时音频数据端点描述符

有关此规范的详细信息，请参见ADC-2 4.10.1.2。

不支持bmAttributes字段中的MaxPacketsOnly标志，将被忽略。

字段bmControls，bLockDelayUnits和wLockDelay将被忽略。

## 类请求和中断数据消息

驱动程序支持ADC-2第5.2节中定义的控制请求的子集，并支持某些控件的中断数据消息（ADC-2 6.1）。下表显示了在驱动程序中实现的子集。

| **实体** | **控制** | **获取cur** | **SET CUR** | **获取范围** | **打断** |
| --- | --- | --- | --- | --- | --- |
| 时钟源 | 采样频率控制 | X | X | X |  |
| 时钟选择器 | 时钟选择器控制 | X |  |  |  |
| 时钟倍频器 | 分子控制 | X |  |  |  |
|  | 分母控制 | X |  |  |  |
| 端点 | 连接器控制 | X |  |  | X |
| 混音器单元 | 混音器控制 | X | X | X |  |
| 选择器单元 | 选择器控制 | X | X |  |  |
| 功能单元 | 静音控制 | X | X |  | X |
|  | 音量控制 | X | X | X | X |
|  | 自动增益控制 | X | X |  |  |
| 效果器 | – |  |  |  |  |
| 处理单元 | – |  |  |  |  |
| 扩展单元 | – |  |  |  |  |

以下各节提供了有关控件和请求的其他信息。

### 时钟源实体

有关此规范的详细信息，请参见ADC-2 5.2.5.1。

时钟源实体至少必须在兼容的USB Audio 2.0硬件中实现采样频率控制GET RANGE和GET CUR请求（ADC-2 5.2.5.1.1）。

采样频率控制GET RANGE请求返回一个子范围列表（ADC-2 5.2.1）。每个子范围描述一个离散频率或一个频率范围。必须通过将MIN和MAX字段设置为相应的频率并将RES设置为零来表示离散采样频率。各个子范围不得重叠。如果某个子范围与先前的范围重叠，则驱动程序将忽略该子范围。

仅实现一个固定频率的时钟源实体不需要实现采样频率控制SET CUR。它实现了返回固定频率的GET CUR，并且实现了报告单个离散频率的GET RANGE。

### 时钟选择器实体

有关此规范的详细信息，请参见ADC-2 5.2.5.2

USB Audio 2.0驱动程序不支持时钟选择。驱动程序使用默认情况下选择的时钟源实体，并且从不发出时钟选择器控件SET CUR请求。时钟选择器控件GET CUR请求（ADC-2 5.2.5.2.1）必须在兼容的USB Audio 2.0硬件中实现。

### 功能单元

有关此规范的详细信息，请参见ADC-2 5.2.5.7。

该驱动程序仅支持一个单一的音量范围。如果“音量控制GET RANGE”请求返回多个范围，则后续范围将被忽略。

MIN和MAX字段表示的音量间隔应为RES字段中指定的步长的整数倍。

如果功能部件实现了单通道控件以及“静音”或“音量”的主控件，则驱动程序将使用单通道控件，而忽略该主控件。

## OEM和IHV的其他信息

OEM和IHV应根据提供的内置驱动程序测试其现有和新设备。

没有任何特定的合作伙伴自定义项与内置USB Audio 2.0驱动程序关联。

此INF文件条目（在Windows版本1703的更新中提供），用于标识内置驱动程序是通用设备驱动程序。

GenericDriverInstalled,,,,1

内置驱动程序向usbaudio2.inf注册以下兼容ID。

USB\Class\_01&SubClass\_00&Prot\_20

USB\Class\_01&SubClass\_01&Prot\_20

USB\Class\_01&SubClass\_02&Prot\_20

USB\Class\_01&SubClass\_03&Prot\_20

有关子类类型，请参见USB音频2.0规范。

带有MIDI的USB Audio 2.0设备（上面的子类0x03）将把MIDI功能枚举为单独的多功能设备，并装有usbaudio.sys（USB音频1.0驱动程序）。

USB Audio 1.0类驱动程序将此兼容的ID注册到wdma\_usb.inf。

USB\Class\_01

并具有以下排除条件：

USB\Class\_01&SubClass\_00&Prot\_20

USB\Class\_01&SubClass\_01&Prot\_20

USB\Class\_01&SubClass\_02&Prot\_20

USB\Class\_01&SubClass\_03&Prot\_20

由于Windows音频堆栈的限制，在共享模式下不支持任意数量的通道（大于八个）。

## IHV USB Audio 2.0驱动程序和更新

对于IHV提供的第三方驱动程序USB Audio 2.0驱动程序，这些驱动程序将继续比其内置驱动程序更受其设备的欢迎，除非他们更新其驱动程序以显式覆盖此行为并使用内置驱动程序。

## 音频插孔注册表说明

从Windows 10版本1703开始，创建具有一个或多个插孔的USB Audio Class 2.0设备的IHV可以将这些插孔描述为内置Audio Class 2.0驱动程序。内置驱动程序在处理此设备的KSPROPERTY\_JACK\_DESCRIPTION时会使用提供的插孔信息。

插孔信息存储在注册表中的设备实例键值（HW键值）中。

下面介绍了注册表中的音频插孔信息设置：

REG\_DWORD T<tid>\_NrJacks # of the jack on this device

REG\_DWORD T<tid>\_J<n>\_ChannelMapping Channel mask. The value is defined in ksmedia.h. e.g. SPEAKER\_FRONT\_RIGHT or KSAUDIO\_SPEAKER\_5POINT1\_SURROUND

REG\_DWORD T<tid>\_J<n>\_ConnectorType The enum value is define in EPcxConnectionType.

REG\_DWORD T<tid>\_J<n>\_GeoLocation The enum value is define in EPcxGeoLocation.

REG\_DWORD T<tid>\_J<n>\_GenLocation The enum value is define in EPcxGenLocation.

REG\_DWORD T<tid>\_J<n>\_PortConnection The enum value is define in EPxcPortConnection.

REG\_DWORD T<tid>\_J<n>\_Color The color needs to be represent by RGB like this: 0x00RRGGBB (NOT a COLORREF).

<tid> =终端ID（在描述符中定义）

<n> =插孔编号（1〜n）。

<tid>和<n>的约定为：

* 以10为底（8、9、10而不是8、9，a）
* 无前导零
* n基于1（第一个插孔为插孔1而不是插孔0）

例如：

T1\_NrJacks，T1\_J2\_ChannelMapping，T1\_J2\_ConnectorType

有关其他音频插孔的信息，请参见[KSJACK\_DESCRIPTION结构](https://docs.microsoft.com/windows-hardware/drivers/audio/ksjack-description)。

可以通过多种方式设置这些注册表值：

* 使用用于包装内置INF的自定义INF来设置这些值。
* 由硬件设备直接通过用于USB设备的Microsoft OS描述符（请参见下面的示例）。有关创建这些描述符的更多信息，请参见[USB设备的Microsoft OS描述符](https://docs.microsoft.com/windows-hardware/drivers/usbcon/microsoft-defined-usb-descriptors)。

### USB示例的Microsoft OS描述符

以下用于USB的Microsoft OS描述符示例包含一个插孔的通道映射和颜色。该示例适用于具有单个特征描述符的非复合设备。

IHV供应商应将其扩展为包含有关插口说明的任何其他信息。

## 故障排除

如果驱动程序没有启动，则应检查系统事件日志。驱动程序记录指示失败原因的事件。同样，可以按照[此博客条目中](https://blogs.msdn.microsoft.com/matthew_van_eerde/2017/01/09/collecting-audio-logs-the-old-fashioned-way/)描述的步骤手动收集音频日志。如果故障可能表明驱动程序有问题，请使用下面描述的“反馈中心”进行报告，并包括日志。

有关如何使用补充TMF文件读取USB Audio 2.0类驱动程序日志的信息，请参阅[此博客条目](https://blogs.msdn.microsoft.com/matthew_van_eerde/2017/10/23/how-to-gather-and-read-logs-for-microsofts-usb-audio-2-0-class-driver/)。有关使用TMF文件的一般信息，请参阅[显示](https://docs.microsoft.com/windows-hardware/drivers/devtest/displaying-a-trace-log-with-a-tmf-file)带有TMF文件[的跟踪日志](https://docs.microsoft.com/windows-hardware/drivers/devtest/displaying-a-trace-log-with-a-tmf-file)。

## 反馈中心

如果您遇到此驱动程序问题，请收集音频日志，然后按照[此博客条目中](https://blogs.msdn.microsoft.com/matthew_van_eerde/2016/09/26/report-problems-with-logs-and-suggest-features-with-the-feedback-hub/)概述的步骤通过“反馈中心”通知我们。

## 驱动程序开发

此USB Audio 2.0类驱动程序由[Thesycon](https://www.thesycon.de/eng/home.shtml)开发，并得到Microsoft的支持。

# 6. 音频硬件资源管理

Windows 10包含使用和XML文件表示并发约束的功能。在资源受限的移动设备上，为特定音频流指定优先级的能力可以增强客户体验。

**注意: 此机制仅在手机和平板电脑中可用。**

在低成本移动设备上创建良好音频体验的一个挑战是，某些设备具有各种并发限制。例如，设备可能最多只能同时播放6个音频流，并且仅支持2个分担(offload)流。当移动设备上有正在进行的电话通话时，该设备可能仅支持2个音频流。设备正在捕获音频时，设备最多只能播放4个音频流。

Windows 10包含一种表达并发约束的机制，以确保能够播放高优先级的音频流和蜂窝电话。如果系统没有足够的资源，则终止低优先级流。**仅在台式机或笔记本电脑上的电话和平板电脑中可以使用此机制。**

要指定约束，请完成以下两个步骤。

* 如**指定并发资源约束**章节中所述，创建并发约束XML文件。
* 如**注册表配置**章节中所述，配置注册表项以使用自定义并发约束XML文件。

## 6.1. 指定并发资源约束

XML约束文件由三部分组成。第一个必需部分由<Limits> </ Limits>定义。本部分最多可用于定义十五种资源限制。例如，您可以为最大渲染流数和可以分担(offload)的最大流数定义约束。

<?xml version="1.0" encoding="utf-8"?>

<ConstraintModel>

<Limits>

<Resource>

<ID>MaxRender</ID>

<Consumption>6</Consumption>

</Resource>

<Resource>

<ID>MaxOffLoad</ID>

<Consumption>2</Consumption>

</Resource>

...

</Limits>

XML文件的下一部分定义一个或多个互斥端点列表，每个列表包含两个或多个端点。这些端点不能同时处于活动状态。本部分是可选的。

例如，如果音频硬件将HandsetSpeaker和WiredHeadsetSpeaker都连接到同一DAC，但无法同时激活，则它们应位于同一ExclusiveEndpoints列表中。

此部分可以具有多个<ExclusiveEndpoints>节点。每个ExclusiveEndpoints节点包含两个或多个Endpoint节点。每个端点节点都包含HWID，TopologyName和PinId。

<ExclusiveEndpoints>

<Endpoint>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

</Endpoint>

<Endpoint>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetSpeaker</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

</Endpoint>

</ExclusiveEndpoints>

XML文件的最后一个必需部分定义了各种资源使用者。文件的此部分包含多个<ResourceConsumer>条目。每个条目标识有关资源使用者及其相关资源使用的信息。必须在<Limits>部分中预先定义所使用的每个资源。

<ResourceConsumer>

<!-- Active Phone call -->

<ConsumerInfo>

<PhoneCall>

<CallState>Active</CallState>

</PhoneCall>

</ConsumerInfo>

<Resource>

<ID>MaxRender</ID>

<Consumption>2</Consumption>

</Resource>

<Resource>

<ID>MaxOffLoad</ID>

<Consumption>2</Consumption>

</Resource>

...

</ResourceConsumer>

使用音频资源时，音频服务会跟踪资源。当没有足够的资源可用时，如果现有资源使用方的优先级较高，则终止优先级较低的流或当前资源请求失败。

这些是有效的<ConsumerInfo>条目。

* <PhoneCall>-<Phonecall>节点包含一个带有CallState子节点，可以为“活动”或“保持”。
* <Stream>-音频流。<Stream>节点包含以下子节点。

<HWID-在驱动程序的INF文件中指定的资源使用者的硬件ID（hw-id）。

<TopologyName>-资源使用方的拓扑过滤器参考字符串。

<PinId>-资源使用者的引脚ID。

<Mode>-关联模式的GUID。有关更多信息，请参见[音频信号处理模式](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-signal-processing-modes)。

<ConnectorType>-资源使用者的连接器类型。有效值为：主机(Host)，环回(loopback)或分担(offload)。

* <FM>-FM收音机。
* <KeywordDetector>-用于支持Cortana语音交互的关键字检测器。

下表总结了渲染音频流优先级，从最高优先级到最低优先级列出。

| 名称 | Name | 优先级 |
| --- | --- | --- |
| 通讯技术 | Communications | 1 |
| 游戏聊天 | Game Chat | 2 |
| 屏幕阅读器 | Screen Reader | 3 |
| 相机快门 | Camera Shutter | 4 |
| 语音消息, 对讲机 | Push To Talk | 5 |
| 通话中通知 | In Call Notification | 6 |
| 个人助手 | Personal Assistant | 6 |
| 语音 | Speech | 7 |
| 铃声 | Ringtone | 8 |
| 闹钟 | Alarm | 9 |
| 电影 | Movie | 10 |
| 前端媒体 | Foreground Only Media | 10 |
| 后台媒体 | Background Capable Media | 11 |
| 媒体 | Media | 11 |
| 声音特效 | Sound Effects | 12 |
| 双音多频 | DTMF | 12 |
| 游戏媒体 | Game Media | 12 |
| 系统 | System | 12 |
| 游戏特效 | Game Effects | 12 |
| 其他 | Other | 13 |
| 提示音 | Alerts | 14 |

下表总结了捕获音频流的优先级，从最高优先级到最低优先级列出。

| 名称 | Name | 优先级 |
| --- | --- | --- |
| 通讯技术 | Communications | 1 |
| 游戏聊天 | Game Chat | 2 |
| 语音信息,对讲机 | Push To Talk | 4 |
| 个人助理 | Personal Assistant | 6 |
| 语音 | Speech | 7 |
| 后台媒体 | Background Capable Media | 8 |
| 媒体 | Media | 8 |
| 其他 | Other | 13 |
| 游戏媒体 | Game Media | 15 |
| 屏幕阅读器 | Screen Reader | 15 |
| 提示音 | Alerts | 15 |
| 前端媒体 | Foreground Only Media | 15 |
| 游戏特效 | Game Effects | 15 |
| 声音特效 | Sound Effects | 15 |
| 双音多频 | DTMF | 15 |
| 通话中通知 | In Call Notification | 15 |
| 闹钟 | Alarm | 15 |
| 相机快门 | Camera Shutter | 15 |
| 电影 | Movie | 15 |
| 铃声 | Ringtone | 15 |
| 系统 | System | 15 |

**例子**

* 示例1：用户正在使用Communications 渲染和捕获流通过Skype进行交谈。他们开始游戏，尝试创建游戏效果流。如果没有足够的可用资源，Game Effects流的创建将失败。
* 示例2：用户正在播放音乐。如果他们启动一个创建语音流的应用程序, 且没有足够的可用资源，则音乐流将终止，并且语音流创建将成功。

## 6.2. 注册表项配置

需要在以下注册表项中指定并发约束XML文件的完整路径。

HKR\SYSTEM\MultiMedia\DeviceCapability\ResourceSettings\XMLConfig

该路径是相对于驱动程序安装的。在驱动程序INF安装中，需要复制约束XML文件，并将添加以下行以将其注册到系统：

HKR,SYSTEM\MultiMedia\DeviceCapability\ResourceSettings\XMLConfig,<Name of the constraint>,,<Path to the constraint>

在此注册表项中，提供一个包含XML路径的值。建议XML文件的名称和regkey值名称是唯一的，因为其他子系统/音频设备有可能在XML文件中提供自己的约束集。可以在音频驱动程序INF文件中设置regkey。

## 6.3. 示例XML约束文件

这是SYSVAD虚拟音频驱动程序样本中的XML约束文件示例。

<?xml version="1.0" encoding="utf-8"?>

<ConstraintModel>

<Limits>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>3</Consumption>

</Resource>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>2</Consumption>

</Resource>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>2</Consumption>

</Resource>

<Resource>

<ID>MaxOneLoopback</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>127</Consumption>

</Resource>

</Limits>

<ExclusiveEndpoints>

<Endpoint>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

</Endpoint>

<Endpoint>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetSpeaker</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

</Endpoint>

</ExclusiveEndpoints>

<ResourceConsumer>

<!-- Phone call -->

<ConsumerInfo>

<PhoneCall>

<CallState>Active</CallState>

</PhoneCall>

</ConsumerInfo>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>2</Consumption>

</Resource>

<Resource>

<ID>MaxOneLoopback</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>26</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- FM -->

<ConsumerInfo>

<FM />

</ConsumerInfo>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- Keyword Detector -->

<ConsumerInfo>

<KeywordDetector />

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, default mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<!--Signal processing mode default-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, Communications mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{98951333-B9CD-48B1-A0A3-FF40682D73F7}</Mode>

<!--Signal processing mode Communications-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, Speech mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{FC1CFC9B-B9D6-4CFA-B5E0-4BB2166878B2}</Mode>

<!--Signal processing mode Speech-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, Notification mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{9CF2A70B-F377-403B-BD6B-360863E0355C}</Mode>

<!--Signal processing mode Notification-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, Media mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{4780004E-7133-41D8-8C74-660DADD2C0EE}</Mode>

<!--Signal processing mode Media-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, Movie mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{B26FEB0D-EC94-477C-9494-D1AB8E753F6E}</Mode>

<!--Signal processing mode Movie-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, raw mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<!--Signal processing mode raw-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, default mode, offload -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<!--Signal processing mode default-->

<ConnectorType>Offload</ConnectorType>

<!-- Offload -->

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, Media mode, Offload -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{4780004E-7133-41D8-8C74-660DADD2C0EE}</Mode>

<!--Signal processing mode Media-->

<ConnectorType>Offload</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, Movie mode, offload -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{B26FEB0D-EC94-477C-9494-D1AB8E753F6E}</Mode>

<!--Signal processing mode Movie-->

<ConnectorType>Offload</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to speaker, default mode, loopback -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<!--Signal processing mode default-->

<ConnectorType>Loopback</ConnectorType>

<!-- Loopback -->

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneLoopback</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, default mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, Communications mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{98951333-B9CD-48B1-A0A3-FF40682D73F7}</Mode>

<!--Signal processing mode Communications-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, Speech mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{FC1CFC9B-B9D6-4CFA-B5E0-4BB2166878B2}</Mode>

<!--Signal processing mode Speech-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, Notification mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{9CF2A70B-F377-403B-BD6B-360863E0355C}</Mode>

<!--Signal processing mode Notification-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, Media mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{4780004E-7133-41D8-8C74-660DADD2C0EE}</Mode>

<!--Signal processing mode Media-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, Movie mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{B26FEB0D-EC94-477C-9494-D1AB8E753F6E}</Mode>

<!--Signal processing mode Movie-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, raw mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode raw-->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, default mode, offload -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<!-- Offload -->

<ConnectorType>Offload</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, Media mode, Offload -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{4780004E-7133-41D8-8C74-660DADD2C0EE}</Mode>

<!--Signal processing mode Media-->

<ConnectorType>Offload</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, Movie mode, offload -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{B26FEB0D-EC94-477C-9494-D1AB8E753F6E}</Mode>

<!--Signal processing mode Movie-->

<ConnectorType>Offload</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to wired headset, default mode, loopback -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologySpeakerHeadset</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<!-- Loopback -->

<ConnectorType>Loopback</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneLoopback</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to BT speaker, raw mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyBthHfpSpeaker</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode raw-->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to BT speaker, raw mode, offload -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyBthHfpSpeaker</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode raw-->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<!-- Offload -->

<ConnectorType>Offload</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxTwoOffload</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to BT speaker, default mode, loopback -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyBthHfpSpeaker</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<!-- Loopback -->

<ConnectorType>Loopback</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneLoopback</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to handset speaker, default mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetSpeaker</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to handset speaker, Communications mode, host -->

<ConsumerInfo>

<Stream>

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<TopologyName>TopologyHandsetSpeaker</TopologyName>

<!-- Topology filter reference string-->

<PinId>1</PinId>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<Mode>{98951333-B9CD-48B1-A0A3-FF40682D73F7}</Mode>

<!--Signal processing mode Communications-->

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to handset speaker, raw mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetSpeaker</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode raw-->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream to handset speaker, default mode, loopback -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetSpeaker</TopologyName>

<!-- KSPIN\_TOPO\_LINEOUT\_DEST -->

<PinId>1</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<!-- Loopback -->

<ConnectorType>Loopback</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxThreeRender</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneLoopback</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic, default mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicIn</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic, communications mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicIn</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode communications-->

<Mode>{98951333-B9CD-48B1-A0A3-FF40682D73F7}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic, speech mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicIn</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode speech-->

<Mode>{FC1CFC9B-B9D6-4CFA-B5E0-4BB2166878B2}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic, notification mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicIn</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode notification-->

<Mode>{9CF2A70B-F377-403B-BD6B-360863E0355C}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic, raw mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicIn</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode raw-->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from wired headset mic, default mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicHeadset</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from wired headset mic, communications mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicHeadset</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode communications-->

<Mode>{98951333-B9CD-48B1-A0A3-FF40682D73F7}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from wired headset mic, speech mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicHeadset</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode speech-->

<Mode>{FC1CFC9B-B9D6-4CFA-B5E0-4BB2166878B2}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from wired headset mic, notification mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicHeadset</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode notification-->

<Mode>{9CF2A70B-F377-403B-BD6B-360863E0355C}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from wired headset mic, raw mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicHeadset</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode raw-->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic array, default mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicArray1</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic array, communications mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicArray1</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode communications-->

<Mode>{98951333-B9CD-48B1-A0A3-FF40682D73F7}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic array, speech mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicArray1</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode speech-->

<Mode>{FC1CFC9B-B9D6-4CFA-B5E0-4BB2166878B2}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic array, notification mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicArray1</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode notification-->

<Mode>{9CF2A70B-F377-403B-BD6B-360863E0355C}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from mic array, raw mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyMicArray1</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode raw-->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from BT mic, default mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyBthHfpMic</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from BT mic, raw mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyBthHfpMic</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode raw-->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from handset mic, default mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetMic</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode default-->

<Mode>{C18E2F7E-933D-4965-B7D1-1EEF228D2AF3}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from handset mic, communications mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetMic</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode communications-->

<Mode>{98951333-B9CD-48B1-A0A3-FF40682D73F7}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from handset mic, speech mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetMic</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode speech-->

<Mode>{FC1CFC9B-B9D6-4CFA-B5E0-4BB2166878B2}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from handset mic, notification mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetMic</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode notification-->

<Mode>{9CF2A70B-F377-403B-BD6B-360863E0355C}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>2</Consumption>

</Resource>

</ResourceConsumer>

<ResourceConsumer>

<!-- AudioStream from handset mic, raw mode, host -->

<ConsumerInfo>

<Stream>

<!-- Example of h/w id specified in phoneaudiosample.inf -->

<HWID>Root\sysvad\_PhoneAudioSample</HWID>

<!-- Topology filter reference string-->

<TopologyName>TopologyHandsetMic</TopologyName>

<!-- KSPIN\_TOPO\_MIC\_ELEMENTS -->

<PinId>0</PinId>

<!--Signal processing mode raw-->

<Mode>{9E90EA20-B493-4FD1-A1A8-7E1361A956CF}</Mode>

<ConnectorType>Host</ConnectorType>

</Stream>

</ConsumerInfo>

<Resource>

<ID>MaxTwoCapture</ID>

<Consumption>1</Consumption>

</Resource>

<Resource>

<ID>MaxOneRawStreamInPhoneCall</ID>

<Consumption>1</Consumption>

</Resource>

</ResourceConsumer>

</ConstraintModel>

# 7. 音频驱动的蓝牙旁路设计指南

这部分内容较多, 将另外采用一篇文章来记录(尚未完成或未更新到此).

# 8. 硬件分担音频处理

这部分已经在其他文档实现, 请参考: <https://blog.csdn.net/danteLiujie/article/details/100701660>

# 9. [WDM音频的平台差异](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/wdm-audio-platform-differences)

本节讨论各种Windows版本中提供的Microsoft Windows驱动程序模型（WDM）音频驱动程序的支持级别。

下列主题描述了可用的不同Windows版本中对WDM音频的支持：

Windows Vista支持以下WDM音频功能：

* WDM版本1.40
* Windows XP支持的所有功能，但以下情况除外。
  + 支持硬件峰值计，但Windows Vista混合器API在每个应用程序模式下运行时除外。
  + Wave和MIDI NT4驱动程序可以正常运行，但是新的音频用户界面不支持它们。
  + 不支持AUX设备，并且Mmsystem.h中的[**auxGetNumDevs**](https://docs.microsoft.com/previous-versions/dd756713(v=vs.85))函数将始终返回零计数。
  + 不支持Windows NT4样式的混合器驱动程序（DRIVERS32）。

Windows Driver Kit（WDK）文档中的其他位置提供了有关WDM平台差异的其他信息：

* 有关Windows版本之间对PortCls功能和接口的支持如何不同的信息，请参阅《[操作系统对PortCls的支持》](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/portcls-support-by-operating-system)。
* 有关PortCls适配器驱动程序的版本号在Windows版本之间如何变化的信息，请参阅[“音频驱动程序的版本号”](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/version-numbers-for-audio-drivers)。
* 有关Windows版本之间一般对WDM驱动程序的支持如何有所不同的信息，包括对驱动程序浮点问题的讨论，请参阅[WDM版本中的差异](https://docs.microsoft.com/windows-hardware/drivers/kernel/differences-in-wdm-versions)。

# 10. [WDM音频组件](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/wdm-audio-components)

这部分内容较多, 将另外采用一篇文章来记录(尚未完成或未更新到此).

# 11. 典型音频配置

这部分内容较多, 将另外采用一篇文章来记录(尚未完成或未更新到此).

# 12. 影响波形输出流([Wave-Output)延迟的因素](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/factors-governing-wave-output-latency)

本节描述了波形输出流的延迟源。影响延迟的一些因素包括：

* 流是输出到WaveCyclic还是WavePci设备
* 流是通过DirectSound还是**waveOut** API 生成的

讨论了以下主题：

## 12.1. WaveCyclic延迟

如果您的WaveCyclic微型端口驱动程序提供了音频播放流的硬件混音器，则DirectSound会将IRP提交给WaveCyclic端口驱动程序，该驱动程序在单个循环缓冲区中包含整个DirectSound波流。WaveCyclic端口驱动程序接收IRP，并将波形数据逐个导入驱动程序公开的DMA缓冲区。WaveCyclic尝试将DMA缓冲区的写指针保持在读指针之前约40毫秒。即使您的驱动程序正在使用DirectSound进行硬件混音，它也可以预期DMA缓冲区中大约有40毫秒的额外数据。

WaveCyclic端口驱动程序尝试在循环缓冲区中累积多达40毫秒的数据这一事实并不意味着WaveCyclic端口驱动程序会使流的延迟增加40毫秒。实际上，端口驱动程序几乎没有增加延迟。就在新的流开始播放之前，当端口驱动程序在将初始数据写入循环缓冲区的开头时，端口驱动程序会持续写入，直到没有更多数据可用或缓冲区包含完整的40毫秒数据为止。但是，如果可用数据少于此数量，端口驱动程序不会强制微型端口驱动程序等待。相反，它允许微型端口驱动程序立即开始播放已经缓冲的数据。随着更多数据可用，端口驱动程序继续将数据写入缓冲区，直到没有更多数据可用或读写指针之间缓冲的数据量达到40毫秒为止。

经过一段时间的数据荒后，KMixer流可能包含静音片段。如果WaveCyclic从KMixer接收到的波数据仅够在DMA缓冲区中保留30而不是40毫秒的额外数据，则WaveCyclic将, 在DMA缓冲区中的KMixer的有效数据后, 写入静音数据。此策略可确保如果发生数据饥荒并且设备读取的数据越界(超出有效数据的末尾)，则音频设备将呈现静音状态，而不是旧数据或未初始化的数据。

写入DMA缓冲区的静音量应保持很小，如果在播放静音之前KMixer向WaveCyclic端口驱动程序提供了新的数据，则新数据将覆盖缓冲区中的静音。在没有数据荒的情况下，音频设备会接收连续的混音数据流，而不会出现强制静音片段。但是，在调试驱动程序时，即使音频渲染器没有数据荒，您也可能会看到微型端口驱动程序的[**IMiniportWaveCyclicStream :: Silence**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavecyclicstream-silence)方法被调用。

## 12.2. WavePci延迟

如果您的WavePci微型端口驱动程序提供硬件混音，则DirectSound会将IRP提交给WavePci端口驱动程序，该驱动程序将整个DirectSound波流包含在单个循环缓冲区中。DirectSound将缓冲区分配为虚拟内存的连续块。为避免复制DirectSound缓冲区，内核流传输层将缓冲区映射到内核模式虚拟内存，并生成一个MDL（内存描述符列表），该MDL指定循环缓冲区中内存页面的虚拟和物理地址。WavePci端口驱动程序将循环缓冲区划分为一系列分配器帧 (allocator frame, 请参阅[KS分配器](https://docs.microsoft.com/windows-hardware/drivers/stream/ks-allocators)）。微型端口驱动程序在其[**IMiniportWavePciStream :: GetAllocatorFraming**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavepcistream-getallocatorframing)时指定其首选的分配器帧大小。在流初始化期间，端口驱动程序将调用该方法。但是，系统图音频构建器SysAudio可以覆盖微型端口驱动程序的首选项，以适应音频过滤器图中其他组件的要求。

WavePci端口驱动程序以一系列映射的方式将循环缓冲区呈现给微型端口驱动程序。映射可以是整个分配帧，也可以是帧的一部分。如果特定的分配帧完全位于一个页面内，则端口驱动程序将该帧作为单个映射呈现给微型端口驱动程序。如果分配帧跨越一个或多个页面边界，则端口驱动程序将在每个页面边界处拆分该帧，并将其呈现为两个或多个映射。每次对[**IPortWavePciStream :: GetMapping的**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportwavepcistream-getmapping)调用都会在序列中产生一个连续映射。

与WaveCyclic情况相反，在该情况下，微型端口驱动程序几乎无法控制在硬件上缓冲多少毫秒的数据，而WavePci微型端口驱动程序可以随时控制打开的映射数量。每次调用[**GetMapping**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportwavepcistream-getmapping)时打开的映射数增加一个，而每次调用[**ReleaseMapping**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportwavepcistream-releasemapping)时打开的映射数减少一个。（当然, **GetMapping**调用也可能失败，因此驱动程序对映射数量的还说不上是完全。）通过控制打开的映射数量并跟踪映射的累积大小，微型端口驱动程序可以确定（在取决于映射大小的公差范围内）硬件可用的缓冲时间（以毫秒为单位）。WavePci微型端口驱动程序应请求足够的页面映射，以将数据荒的概率降低到可接受的水平。

如果微型端口驱动程序的策略是, 例如在读写指针之间, 最多缓冲50毫秒的数据，请记住，该限制表示驱动程序将累积的最大数据量，但并不表示驱动程序会引入这么多的流延迟(stream latency)。驱动程序的设计应使时延(latency)尽可能小。当微型端口驱动程序在开始播放新流之前获得其初始映射集时，微型端口驱动程序可以继续请求映射，直到达到其缓冲区限制（在此示例中为50毫秒）或不再有可用的映射为止。但是，在后一种情况下，微型端口驱动程序不必等到有更多映射可用后才开始播放。而应立即开始播放已经获得的映射。以后，随着更多的映射可用，驱动程序可以继续获取其他映射，直到达到其缓冲区大小限制或没有更多的映射立即可用。

通常，WavePci设备的DMA硬件应设计为直接访问音频帧, 以任意字节对齐方式存储, 并能跨越物理内存不连续页面之间边界。如果您的设备要求映射是整数个音频帧，则该设备受其支持的音频格式种类的限制。当然，具有此限制的设备仍应能够处理为2的幂的音频帧大小。

例如，具有四个通道和16位样本大小的设备需要八个字节的音频帧大小。整数个音频帧可以整齐地放在一个页面中（或任何其他分配帧大小，为八个字节的倍数）。但是，在具有16位样本的5.1通道流的情况下，音频帧大小为12字节，并且超出单个页面大小的流必定包含跨越页面边界的音频帧。（[波形filter驱动中](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/wave-filters)的数字说明了此问题。）无法处理任意字节对齐和任意字节长度映射的硬件，必须依靠驱动程序来执行中间复制，这会降低性能。

Microsoft Windows驱动程序工具包（WDK）中的Ac97示例适配器驱动程序实现了[**GetAllocatorFraming**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iminiportwavepcistream-getallocatorframing)方法。微型端口驱动程序使用此方法来传达其首选的帧分配大小。在Windows 2000和Windows Me中，仅当在输出引脚上方实例化[Splitter系统驱动程序](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/kernel-mode-wdm-audio-components#splitter_system_driver)（Splitter.sys）时，端口驱动程序才调用此方法。在Windows XP和更高版本中，端口驱动程序也为输入流调用此方法。请记住，在决定帧分配大小时，SysAudio可能会选择忽略微型端口驱动程序的首选项。

## 12.3. 避免数据拷贝

您可以通过设计音频硬件来避免不必要的数据复制，从而提高驱动程序性能。

通过实现硬件以执行真正的分散/收集DMA并编写WavePci微型端口驱动程序来管理硬件，可以达到最佳效果。然后，您的设备可以直接访问播放数据缓冲区或空记录缓冲区，无论它们位于系统内存中的任何位置。这消除了许多不必要的软件干预和费时的数据拷贝。

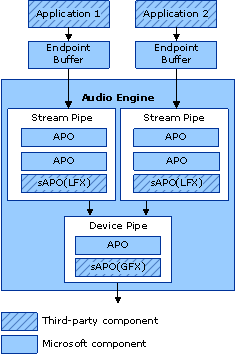
但是，如果要设计WaveCyclic设备，则可以通过直接将其硬件缓冲区作为系统内存进行访问来提高其性能。这消除了从系统内存中的中间缓冲区复制数据的开销。

另外，如果您的设备要求使用的音频格式的通道顺序与标准WDM音频格式不兼容，则驱动程序可能必须在中间缓冲区中对每个音频帧执行就地转换，然后硬件才能对其进行处理。这会降低性能。有关更多信息，请参见[音频技术](https://go.microsoft.com/fwlink/p/?linkid=8751)网站上标题为“[多通道音频数据和WAVE文件](https://docs.microsoft.com/en-us/previous-versions/windows/hardware/design/dn653308(v=vs.85))”的白皮书。

# 13. 探索[Windows Vista音频引擎](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/exploring-the-windows-vista-audio-engine)

本主题概述了Windows Vista音频引擎。它侧重于可帮助您理解APO和sAPO (System Effects Audio Processing Objects ) 如何协同工作的概念。

下图显示了音频引擎内部结构的简化布局。



如图所示，系统提供的APO和sAPO是音频引擎的基本构建块。音频引擎将系统提供的APO和sAPO配置为称为管道的组件。音频引擎中有两种管道：

* 流管道由APO和sAPO组成，它们执行来自单个应用程序的本地流数字音频处理。此类管道中的sAPO称为局部效果sAPO（LFX sAPO）。
* 设备管道由执行数字音频处理的APO和sAPO组成，这些处理会影响全局所有流。此类管道中的sAPO称为全局效果sAPO（GFX sAPO）。

下表显示了Windows Vista音频引擎中可用的sAPO及其应用的系统效果类型。

| **Windows Vista sAPO** | **系统效果类型** |
| --- | --- |
| 低音增强 | LFX |
| 低音管理 | LFX |
| 响度均衡 | LFX |
| 低频保护 | LFX |
| 声道扩展(Speaker Fill) | LFX |
| 扬声器幻影 (声道补全, Speaker Phantoming) | LFX |
| 虚拟环绕 | LFX |
| 耳机虚拟环绕声 | LFX |
| 便携式计算机的声音增强 | GFX |
| 房间校正 | GFX |

音频应用程序启动音频处理时，音频引擎会将系统提供的APO和sAPO配置为音频图，以处理数字音频数据。音频引擎用于构建音频图的机制是系统细节，将不再讨论。

音频应用程序可以以共享模式或独占模式启动连接。尽管Windows Vista安装了一组默认的sAPO，但是sAPO并不被视为系统组件，因此可以自定义。

## 13.1. 共享模式

在共享模式下，音频应用程序与在其他进程中运行的其他音频应用程序共享音频硬件。音频引擎混合来自这些应用程序的流，并通过硬件播放混合结果。在共享模式下打开流的任何应用程序都必须选择音频引擎使用的混合格式。使用共享模式的优点是Windows Vista音频引擎提供了内置的音频处理对象（APO），以提供必要的支持功能。使用共享模式的缺点是音频流时延比独占模式下长。以下代码示例显示了在共享模式下初始化音频流的语法。

hResult = pAudioClient->Initialize(

AUDCLNT\_SHAREMODE\_SHARED,

0,

0,

0,

pWfx, // mix format

&m\_SubmixGuid);

## 13.2. 独占模式

相反，当应用程序以独占模式打开流时，该应用程序具有对音频硬件的独占访问权。在这种模式下，应用程序可以选择端点支持的任何音频格式。使用独占模式的优点是音频流等待时间比共享模式下的等待时间短。使用独占模式的缺点是，您必须提供自己的APO才能处理音频引擎的支持功能。只有少数专业级别的应用程序需要这种操作模式。以下代码示例显示了以独占模式初始化音频流的语法。

hResult = pAudioClient->Initialize(

AUDCLNT\_SHAREMODE\_EXCLUSIVE,

0,

0,

0,

pWfxEx,

&m\_SubmixGuid);

应用程序启动音频处理后，音频图构建器将sAPO配置为音频图，并初始化sAPO。然后，音频服务与LFX sAPO协商以在sAPO的输入和输出处建立音频数据的格式。有关更多信息，请参见格式协商章节。

# 14. 格式协商

应用程序启动音频处理后，音频图构建器将sAPO配置到音频图，并初始化sAPO。然后，音频服务与LFX sAPO协商以在sAPO的输入和输出处设定音频数据格式。此协商过程称为格式协商。

为Windows Vista提供音频系统效果的所有sAPO必须具有某些接口和方法。所使用的SAPO和音频引擎进行协商的数据格式的方法有：在**IsInputFormatSupported**所述的方法**IAudioProcessingObject**接口和**LockForProcess**和**UnlockForProcess**所述的方法**IAudioProcessingObjectConfiguration**接口。

为了启动格式协商，音频服务首先将LFX sAPO的输出设置为默认的基于float32的格式。音频服务然后调用**IAudioProcessingObject :: IsInputFormatSupported**LFX sAPO的方法，建议使用默认格式，并监视此方法的HRESULT响应。如果LFX sAPO可以支持建议的格式，则它将返回S\_OK以及对所支持格式的引用。如果LFX sAPO无法支持建议的格式，它将返回S\_FALSE以及对与建议格式最接近的格式的引用。如果LFX sAPO无法支持建议的格式并且不具有近似匹配项，则它将返回APOERR\_FORMAT\_NOT\_SUPPORTED。GFX sAPO使用LFX sAPO的输出格式。因此，GFX sAPO不参与格式协商过程。

选择一种数据格式来处理音频数据后，音频图构建器将调用sAPO 的**IAudioProcessingObjectConfiguration :: LockForProcess**方法，从而完成格式选择。

如果Windows Vista sAPO响应对**LockForProcess**方法的调用而向包裹它的自定义sAPO返回一个错误，则自定义sAPO必须以, 与实例化sAPO的尝试失败时处理**CoCreateInstance中**的错误的方式相同的方式, 处理该错误。有关如何覆盖系统提供的LockForProcess方法的详细信息，请参考Spkrfill.cpp文件 。(?)

由于音频服务的运行方式，LFX和GFX sAPO必须能够彼此独立地响应来自音频服务的有关数据格式的查询。

**重要说明：**    当您实现包裹Windows Vista LFX sAPO的自定义sAPO时，请勿在自定义sAPO的注册属性中指定APO\_FLAG\_FRAMESPERSECOND\_MUST\_MATCH标志。如果指定此标志，则Windows Vista LFX sAPO将无法执行扬声器填充，耳机虚拟化或虚拟环绕。此外，您的自定义sAPO将无法缩混任何音频流。例如，您的自定义sAPO将无法将5.1音频流混合为两个声道的立体声音频流。

# 15. 动态格式更改

动态格式更改是Windows 7和Windows操作系统的更高版本中的一项功能，该功能允许动态更改用于在音频应用程序和音频适配器之间流音频数据的格式。动态格式更改可适应高清多媒体接口（HDMI）设备中音频流的行为。本主题概述了动态格式更改并描述了其工作方式。

以下列表显示了使用动态格式更改的方案。

* **HDMI设备具有新功能。**当HDMI设备流传输音频或视频数据或同时传输这两种数据时，用于音频和视频传输的HDMI总带宽是固定的，并且视频信号在容量分配中优先。这意味着，如果您将HDMI显示设备连接到计算机，并且更改了显示分辨率，则这会影响音频数据传输到计算机的剩余带宽大小。

例如，假设您的HDMI设备最初配置为将数据格式设置为192 KHz，具有特定显示模式的16位立体声。当您更改为其他显示模式时，流音频数据的剩余带宽可能不足以支持192 KHz格式。因此，设备驱动程序会将显示模式的更改通知已连接计算机的音频服务，这将导致音频驱动程序和音频服务重新协商音频数据格式。如果当前选择的192 KHz格式不能在剩余带宽内传输，则选择新格式。有关格式协商过程的更多信息，请参见[格式协商](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/format-negotiation)。

在另一种与HDMI相关的动态格式更改方案中，拔出音频设备，并插入新的具有HDMI功能的设备。HDMI设备的设备驱动程序生成格式更改事件，并且音频服务与音频设备驱动程序重新协商音频数据格式。

* **某些独立的音频设备提供了硬件控件，用户可以使用这些控件来更改音频数据格式。**在这种情况下，用户可以操纵环绕声放大器上的控制旋钮，例如，选择音频数据格式。如果有一台计算机连接到独立音频设备，则此新选择的数据格式将导致所连接计算机上的音频驱动程序重新协商该数据格式，并可能进行更改。
* **“控制面板”中“声音”小程序的第三方UI提供了用于启用或禁用系统效果的选项。**在开发自己的系统效果音频处理对象（sAPO）时，还可以在“控制面板”中为“ **声音”**小程序提供自定义UI 。此自定义UI可以包括对“ **声音”**小应用程序的“ **增强”**或“ **高级”**选项卡的修改，或者对这两者都进行修改。在这种情况下，用户选中**增强选项卡中**的复选框标签以启用或禁用全局系统效果（GFX）功能会导致音频数据格式的更改。用户所做的选择使HDMI驱动程序生成格式更改事件。音频服务接收有关此事件的通知，并与音频驱动程序重新协商以选择音频数据的新格式。

为了提供对HDMI和IEC61937兼容的压缩音频格式（例如杜比数字和数字影院声音（DTS））的支持，Windows 7和更高版本的Windows操作系统提供了一组新的子类型GUID，供内核流（KS）属性和结构使用。 。国际电工委员会（IEC）标准IEC 61937适用于传输非线性[PCM](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/pcm-stream-data-format)编码的比特流的数字音频接口。有关子类型GUID的更多信息，请参见Ksmedia.h中的KSDATAFORMAT\_SUBTYPE\_IEC61937\_Xxx GUID。

**注意:** 当音频端点构建器收到动态格式更改通知，并且设备驱动程序不支持建议的数据格式时，端点构建器将重新计算新的默认设备数据格式。

在重新设计音频驱动程序来支持新格式的情况下，它可以强制端点构建器选择新格式作为设备的默认格式。要强制将新格式转换为设备的默认格式，音频驱动程序收到对旧格式支持情况的查询时, 应该让查询失败(来表示不支持)。失败的格式支持查询将触发格式更改通知，然后端点构建器将为设备计算新的默认格式。