Windows 硬件分担(Hardware-Offloaded )音频处理

翻译自: <https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/hardware-offloaded-audio-processing> 201704/20 版本

备注:

* 此文档较为久远, 架构图仅包含LFX和GFX, 应该存在较多问题.
* 行文中, 我们将结合使用术语”硬件分担音频处理”和”offloaded音频”

硬件分担(Hardware-offloaded)音频处理允许主要音频处理任务在计算机的主CPU外部执行.

音频处理可能非常耗费MIPS。 因此，在许多场景中，允许专用处理器处理诸如混合和叠加音效之类的处理任务可能更划算。 但Windows 7及更早版本的Windows并未提供对硬件分担音频处理的支持。

对于Windows 8及更高版本的操作系统，音频驱动程序模型已更新，以便为硬件分担音频处理提供支持，以下各节描述了如何开发支持offloaded音频处理的驱动程序。

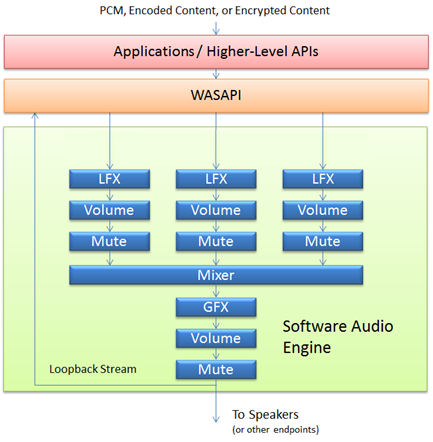
# 1. 架构预览

本节概述了Windows 8中引入的音频体系结构，以提供对组合硬件/软件音频引擎的支持。

## 软件音频引擎

Windows 7和一些早期版本的Windows支持软件音频引擎，允许第三方开发人员在处理管道中的特定位置插入软件解码器，编码器和其他通用音频效果。

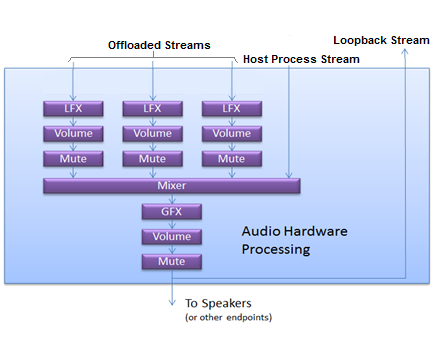
下图显示了Windows 8软件音频引擎:



如上图所示, pcm或其他编码的音频数据流，从应用层通过Windows音频会话API（WASAPI）进入音频引擎, 也可能通过更高级别的API（如Media Foundation）到达. 在引擎中, 先叠加局部效果（LFX），如音量控制和静音; 然后混合(Mix), 在叠加全局效果（GFX）, 然后发送到扬声器.

## 硬件音频引擎

硬件音频引擎在音频适配器中实现，并且在很大程度上镜像了软件音频引擎的功能。虽然Windows 8支持硬件分担音频处理，但音频驱动程序负责实现下图中所示的拓扑和功能。



如图所示，硬件音频引擎必须接受一个host流(主机进程流, Host Process Stream)和最多n个offloaded流。这些offloaded的流直接来自应用层。换句话说，offloaded的流将不会经过软件音频引擎。该图显示了一个可以处理最多三个offload流的实现。host流是软件音频引擎中处理的所有流的最终输出。硬件音频引擎应该包含硬件混音器(Mixer)。

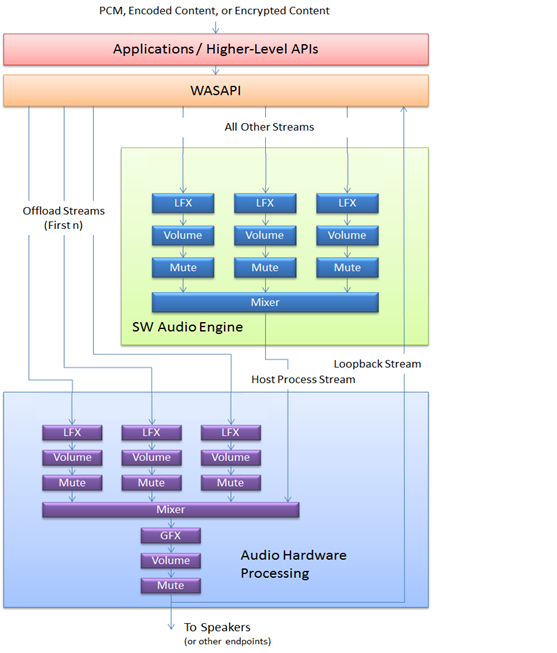
为了保持与软件音频引擎及WASAPI接口的 协同(parity)，硬件音频引擎必须以环回流(loopback stream)的形式将最终音频输出流发回音频堆栈。这对于依赖AEC(Acoustic Echo Cancellation, 回声消除)的应用和场景尤其重要，AEC要求了解最终输出流以消除回声并防止反馈造成啸叫。

为了实现环回流的路径，音频驱动程序应提供环回引脚(loopback pin)。

如果数据被编码为PCM格式，该引脚将从最终音频引擎输出返回音频数据。如果不是PCM格式，将返回后端混合后（Post-Mixing）但在编码前(Pre-Encoding)的结果。这意味着如果硬件GFX在数据处理后输出非PCM格式编码，环回流应该在硬件混音器之后在GFX之前直接提取。有关代表硬件音频引擎的新KS过滤器拓扑的信息，请参阅[Implementation Overview](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/implementation-overview)(本文第二部分第一节).

## 整体架构

下图显示了硬件音频引擎与Windows 8软件音频引擎协同工作时的体系结构。



这意味着在音频驱动程序已表示支持offloaded 音频处理的情况下，最早初始化的n个（在本例中为3个）流将直接从WASAPI层路由到硬件音频引擎，绕过软件音频引擎。硬件音频引擎支持的n之后的任何新音频流将通过软件音频引擎进行路由以进行处理。然后，来自软件音频引擎的结果流作为主机进程流被发送到硬件音频引擎。主进程流与前n个流混合，应用GFX处理，然后将得到的流发送到扬声器。

注意: 通常在音量调整之前应用GFX处理。但是在GFX被编码为非PCM格式的情况下例外; 在这种情况下，GFX /音量控制顺序被反转，以便在GFX处理发生之前可以将音量控制应用于未压缩的数据。 Windows 7软件音频引擎遵循相同的模型。

# 2. 驱动实现

为offloaded音频实现驱动程序时，开发能够处理offloaded音频流的驱动程序，并将该功能展示给Windows音频系统。

本节中的以下主题讨论了,在实现带硬件音频引擎(支持offloaded音频流)的音频适配器驱动程序时, 在UI方面应注意的事项、应用程序影响和其他问题。

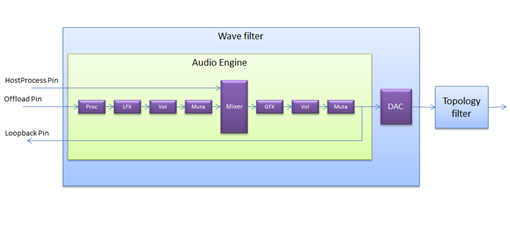
## 2.1. 实现概览

本主题概述了,能够处理硬件分担音频流的音频适配器驱动程序, 的实现要点。

### 2.1.1. 新KS过滤器(filter)拓扑

在Windows 8及更高版本的操作系统中，已经支持使用板载硬件音频引擎来处理音频流的音频适配器。开发此类音频适配器时，相关的音频驱动程序必须以特定方式将此情况展示给用户模式音频系统，以便音频系统可以发现、使用并正确地展示此适配器及其驱动程序的功能。

为了达到此目的，Windows 8引入了一种新的KS过滤器（KS-Filter）拓扑(类型名为Audio Engine)，并要求驱动必须使用:



如上图所示，KS过滤器拓扑表示通过硬件的数据路径，还显示了这些路径上可用的功能。对于可以处理offloaded音频的音频适配器，KS滤波器上有以下输入和输出（称为引脚）：

* 一个主机进程引脚（Host Process Pin）。这表示从软件音频引擎输入KS滤波器。
* 一个Loopback引脚。这表示从硬件音频引擎到Windows音频会话API（WASAPI）层的输出。
* 一些Offloaded-audio引脚。虽然该图仅显示了这种类型的一个引脚，但IHV(independent hardware vendor, 独立硬件供应商)可以自由地实现任意数量（n）的引脚。

有关此新型KS滤波器拓扑中引脚的更多信息，请参阅 [Architectural Overview](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/architectural-overview).

Windows服务AudioEndpointBuilder是用户模式音频系统中“引导”发现音频适配器及其驱动程序的关键部分。AudioEndpointBuilder服务监视KSCATEGORY\_AUDIO类以捕获设备接口插拔动作。当音频设备驱动程序注册KSCATEGORY\_AUDIO设备接口类的新实例时，将触发设备接口插入通知。 AudioEndpointBuilder服务将使用算法检查系统中音频设备的拓扑，以便它可以采取适当的操作。

因此，当您开发支持Offloaded-Audio的适配器驱动时，您的驱动程序必须使用新定义的[**KSNODETYPE\_AUDIO\_ENGINE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-audio-engine) 音频端点来公开硬件音频引擎的功能。有关音频端点发现过程的详细信息，请参阅音频端点生成器算法([Audio Endpoint Builder Algorithm](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/audio-endpoint-builder-algorithm))。

### 2.1.2. 用户界面注意事项

您开发了音频驱动程序来控制能够处理offloaded音频的音频适配器的底层硬件功能。这意味着您的驱动程序具有如何控制适配器功能的最佳知识。因此，您必须开发一个UI，以便以可供选择，启用和/或禁用的选项的形式向最终用户公开适配器的功能。

但是，如果您已经有一个用于控制您开发的音频处理对象（APO）的UI，则可以扩展此UI以使用新的音频适配器。在这种情况下，您对UI的扩展将为APO提供软件控制，并为适配器提供硬件控制。

### 2.1.3. 应用影响

UWP应用程序可以通过WASAPI，Media Foundation，Media Engine或HTML 5 <audio>标签使用此新型音频适配器及其相关驱动程序所描述的功能。请注意，无法使用Wave和DSound，因为UWP应用无法使用它们。另请注意，桌面应用程序(Desktop applications)无法使用支持Hardware-offload音频的音频适配器的offload功能。这些应用程序仍然可以渲染音频，但只能通过使用软件音频引擎的主机引脚。

如果UWP应用程序流式传输媒体内容并使用Media Foundation，Media Engine或HTML 5 <audio>标签，则只要为流设置了正确的音频类别，该应用就会自动选择加载硬件。选择硬件offloaded是基于每个流。

使用WASAPI或流式通信的UWP应用必须明确选择加入硬件。

## 2.2. 驱动实现细节

本主题介绍支持硬件分担音频驱动程序的实现细节。

换句话说，本主题解释了Microsoft已经做了什么（从Windows 8开始）以支持与能够处理硬件分担音频的音频适配器一起工作的驱动程序。在以下部分中，本主题还向您展示了驱动程序必须能够支持此类适配器的功能。

### 2.2.1. 新节点描述符的GUID

如果音频适配器能够处理offloaded的音频流，则适配器的音频驱动程序通过在适配器的KS过滤器中使用新引入的节点来公开此功能。

音频流路径中的每个节点都有一个节点描述符，因此对于这个新节点，驱动程序必须将Type GUID设置为**[KSNODETYPE\_AUDIO\_ENGINE](https://docs.microsoft.com/windows-hardware/drivers/audio/ksnodetype-audio-engine)**.以下是驱动程序如何为此新节点配置节点描述符的示例：

typedef struct \_KSNODE\_DESCRIPTOR {

const KSAUTOMATION\_TABLE \*AutomationTable; // drv specific

const GUID \*Type; // must be set to KSNODETYPE\_AUDIO\_ENGINE

const GUID \*Name; // drv specific (KSNODETYPE\_AUDIO\_ENGINE?)

} KSNODE\_DESCRIPTOR, \*PKSNODE\_DESCRIPTOR;

如果Name GUID设置为**KSNODETYPE\_AUDIO\_ENGINE**，则必须为此节点创建默认名称字符串。然后，将该字符串添加到ks.inf，以便在安装驱动程序期间，该字符串可用于填充HKEY\_LOCAL\_MACHINE \ SYSTEM \ CurrentControlSet \ Control \ MediaCategories注册表项

新节点类型的GUID定义为KSNODETYPE\_AUDIO\_ENGINE，如下所示：

#define STATIC\_KSNODETYPE\_AUDIO\_ENGINE\

0x35caf6e4, 0xf3b3, 0x4168, 0xbb, 0x4b, 0x55, 0xe7, 0x7a, 0x46, 0x1c, 0x7e

DEFINE\_GUIDSTRUCT("35CAF6E4-F3B3-4168-BB4B-55E77A461C7E", KSNODETYPE\_AUDIO\_ENGINE);

#define KSNODETYPE\_AUDIO\_ENGINE DEFINE\_GUIDNAMED(KSNODETYPE\_AUDIO\_ENGINE)

有关更多信息，请参阅ksmedia.h头文件。

并且基于前面的信息，微型端口节点(miniport node)的描述符可能如下所示：

PCNODE\_DESCRIPTOR MiniportNodes[] =

{

// KSNODE\_WAVE\_AUDIO\_ENGINE

{

0, // Flags

NULL, // AutomationTable

&KSNODETYPE\_AUDIO\_ENGINE, // Type KSNODETYPE\_AUDIO\_ENGINE

NULL // Name

}

};

### 2.2.2. 音频引擎的新KS属性集

从Windows 8开始，引入了[KSPROPSETID\_AudioEngine](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-audioengine)属性集以支持硬件音频引擎和硬件分担的音频处理。因此，可以处理Offloaded的音频流的适配器的驱动程序必须支持此新属性集中的属性。

新属性集KSPROPSETID\_AudioEngine定义如下:

#define STATIC\_KSPROPSETID\_AudioEngine\

0x3A2F82DCL, 0x886F, 0x4BAA, 0x9E, 0xB4, 0x8, 0x2B, 0x90, 0x25, 0xC5, 0x36

DEFINE\_GUIDSTRUCT("3A2F82DC-886F-4BAA-9EB4-082B9025C536", KSPROPSETID\_AudioEngine);

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

此新属性集中的属性名称在**[KSPROPERTY\_AUDIOENGINE](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine)**枚举中定义，驱动程序必须支持这些名称。

以下是**KSPROPSETID\_AudioEngine**属性集中的新属性：

[**KSPROPERTY\_AUDIOENGINE\_BUFFER\_SIZE\_RANGE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine-buffer-size-limits)

[**KSPROPERTY\_AUDIOENGINE\_DESCRIPTOR**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine-descriptor)

[**KSPROPERTY\_AUDIOENGINE\_DEVICEFORMAT**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine-deviceformat)

[**KSPROPERTY\_AUDIOENGINE\_GFXENABLE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine-gfx-enable)

[**KSPROPERTY\_AUDIOENGINE\_LFXENABLE**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine-lfx-enable)

[**KSPROPERTY\_AUDIOENGINE\_LOOPBACK\_PROTECTION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine-loopback-protection)

[**KSPROPERTY\_AUDIOENGINE\_MIXFORMAT**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine-mixformat)

[**KSPROPERTY\_AUDIOENGINE\_SUPPORTEDDEVICEFORMATS**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine-supporteddeviceformats)

[**KSPROPERTY\_AUDIOENGINE\_VOLUMELEVEL**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audioengine-volumelevel)

### 2.2.3. KSPROPSETID\_ Audio属性集的更新

除了支持新**KSPROPSETID\_AudioEngine**属性集中的属性外，驱动程序还必须支持 [KSPROPSETID\_Audio](https://docs.microsoft.com/windows-hardware/drivers/audio/kspropsetid-audio)属性集中的以下现有属性：

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

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

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

为了完成对硬件分担音频处理的驱动程序支持的实现，添加一些新属性到KSPROPSETID\_ Audio属性集中。

以下是新的KSPROPSETID\_音频属性：

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

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

[**KSPROPERTY\_AUDIO\_WAVERT\_CURRENT\_WRITE\_POSITION**](https://docs.microsoft.com/windows-hardware/drivers/audio/ksproperty-audio-wavert-current-write-position)

### 2.2.4. 端口级驱动程序(Port-class driver)更新和故障报告

除了前面部分中描述的硬件分担音频处理支持之外，Windows端口类驱动程序还使用“辅助接口(helper interfaces)”进行了更新，以便开发可以处理offloaded音频流的驱动程序。当这样的驱动程序检测到毛刺(glitching)时，有一种机制可以让驱动程序报告毛刺错误。以下两节提供了有关辅助接口和故障报告的更多详细信息.

## 2.3. 音频处理的辅助接口

本主题介绍Microsoft已添加到其音频端口类驱动程序（port-class driver, PortCls）的帮助程序接口，可以简化支持Offloaded音频处理的驱动程序的实现。

当您开发WaveRT微型端口驱动程序(miniport driver)，以支持能够处理硬件分担音频流的音频适配器，您的微型端口驱动程序可与PortCls配合使用，流式传输和/或处理音频数据。

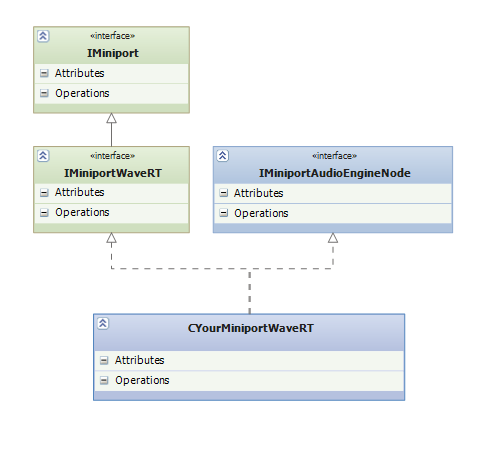
PortCls已经更新，可以处理所有与offloaded相关的内核流（KS）属性，这使得开发WaveRT微型端口驱动程序以支持处理硬件分担音频流变得简单。 作为更新的结果，PortCls仅通过两个新定义的接口调用底层微型端口驱动程序以进行硬件和/或驱动程序特定的操作：

* [**IMiniportAudioEngineNode**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iminiportaudioenginenode)
* [**IMiniportStreamAudioEngineNode**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iminiportstreamaudioenginenode)

您必须开发两个类来使用这些接口，每个接口一个。

### 2.3.1. 使用IMiniportAudioEngineNode

您开发使用**IMiniportAudioEngineNode**的类也必须从[IMiniportWaveRT](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iminiportwavert)继承。 **IMiniportAudioEngineNode**中定义的方法允许您的驱动程序使用KS属性, 这些属性通过KS过滤器句柄访问音频引擎。类/接口层次结构如下：



因此，举个例子，如果您开发了一个名为CYourMiniportWaveRT的类，那么从上图中可以看出，CYourMiniportWaveRT必须实现为两个父接口定义的所有方法（显示为Operations）。

这种类的骨架模板将包含以下代码：

class CMiniportWaveRT :

public IMiniportWaveRT,

public IMiniportAudioEngineNode,

public CUnknown

{

...

IMP\_IMiniportWaveRT;

IMP\_IMiniportAudioEngineNode;

...

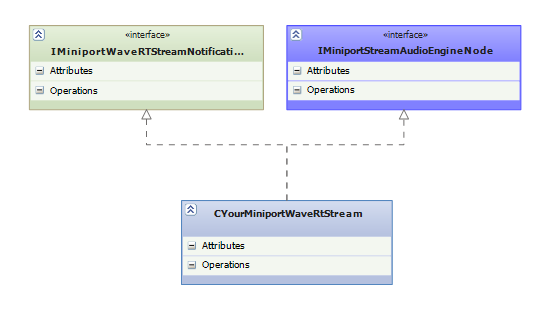
};

头文件Portcls.h定义了这些接口

### 2.3.2. 使用IMiniportStreamAudioEngineNode

The class that you develop to work with the second interface, **IMiniportStreamAudioEngineNode**, must also inherit from [IMiniportWaveRTStreamNotification](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iminiportwavertstreamnotification). The methods defined in **IMiniportStreamAudioEngineNode** allow your driver to use KS properties that access the audio engine via a pin instance handle. The class/interface hierarchy is as follows:

您开发使用第二个接口**IMiniportStreamAudioEngineNode**的类也必须继承自 [IMiniportWaveRTStreamNotification](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iminiportwavertstreamnotification)。 **IMiniportStreamAudioEngineNode**中定义的方法允许您的驱动程序使用KS属性, 这些属性通过引脚实例句柄访问音频引擎。类/接口层次结构如下：



因此，例如，如果您开发了一个名为CYourMiniportWaveRTStream的类，那么从上图中可以看出，CYourMiniportWaveRTStream必须实现为两个父接口定义的所有方法。

这种类的骨架模板将包含以下代码：

class CMiniportWaveRTStream :

public IMiniportWaveRTStreamNotification,

public IMiniportStreamAudioEngineNode,

public CUnknown

{

...

IMP\_IMiniportWaveRTStream;

IMP\_IMiniportWaveRTStreamNotification;

IMP\_IMiniportStreamAudioEngineNode;

...

};

Portcls.h头文件定义了这些接口。有关如何开发可处理硬件分担音频流的驱动程序的更多信息，请参阅驱动程序实现详细信息([Driver Implementation Details](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/driver-implementation-details)).

## 2.4. 针对Offloaded音频的故障报告

本主题说明了音频驱动程序在报告与硬件分担的音频流相关的毛刺(glitching)错误时必须使用的机制。

当音频驱动程序检测到毛刺错误时，它必须引发Windows事件跟踪（ETW）事件以报告错误。此事件应包括毛刺的原因，以及有关用于音频流的DMA缓冲区的信息。

以下枚举显示已为音频驱动程序定义的用于毛刺错误报告的事件。

typedef enum

{

eMINIPORT\_IHV\_DEFINED = 0,

eMINIPORT\_BUFFER\_COMPLETE,

eMINIPORT\_PIN\_STATE,

eMINIPORT\_GET\_STREAM\_POS,

eMINIPORT\_SET\_WAVERT\_BUFFER\_WRITE\_POS,

eMINIPORT\_GET\_PRESENTATION\_POS,

eMINIPORT\_PROGRAM\_DMA,

eMINIPORT\_GLITCH\_REPORT

} EPcMiniportEngineEvent;

有关此枚举的更多信息，请参阅**[EPcMiniportEngineEvent](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ne-portcls-epcminiportengineevent)**.

有关如何开发可处理硬件分担音频流的驱动程序的更多信息，请参阅驱动程序实现详细信息([Driver Implementation Details](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/driver-implementation-details))。

# 3. PortCls 为支持SoC所做的电源管理更新

以下主题讨论了Windows 8中在端口级驱动程序（PortCls）中引入的电源管理更新，以支持片上系统（system-on-a-chip, SoC）平台。

## 3.1. PortCls注册表电源设置

本主题说明Windows 8的PortCls注册表电源设置。

在Windows 8中，（PortCls）微型端口驱动程序可以,使用驱动程序的相关注册表值来,执行以下操作：

* 确定PortCls是否启用空闲电源管理
* 确定电池节能模式的空闲等待值(idle timeout values), 即空闲后等多久才从高性能模式进入节能模式

默认情况下，Windows 8提供了电源设置, 当运行时电源框架(runtime power framework) 指示不再需要电源时， PortCls可以根据相关设置，来决定是否要向电源管理单元注册“设备空闲”检测。用于描述电源设置的参数定义如下。

| **注册表值** | **数据类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| ConservationIdleTime | REG\_BINARY | 0 | 0秒超时 |
| IdlePowerState | REG\_BINARY | 3 (D3)  Valid values:  1 - D1 2 - D2 3 - D3 | 指定当不再需要电源时设备将进入的电源状态 |
| PerformanceIdleTime | REG\_BINARY | 0 | 0秒超时 |

以下Windows注册表片段显示用于提供电源设置信息的语法。

[HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E96C-E325-11CE-BFC1-08002BE10318}\0000\PowerSettings]

"ConservationIdleTime"=hex:1e,00,00,00

"PerformanceIdleTime"=hex:00,00,00,00

“IdlePowerState”=hex:03,00,00,00

前面的片段显示了:

* ConservationIdleTime的十六进制值"1e" ,这相当于30秒的空闲超时(idle timeout).
* PerformanceIdleTime的十六进制值“0”表示已禁用空闲管理。
* IdlePowerState取值为“03”的值意味着当不再需要电源时，与该电源设置配置相关联的设备将进入D3电源状态。

## 3.2. 切换到立即关电状态(Immediate Idle Timeout Opt-in )

本主题讨论注册表值ImmediateIdle, 在不再需要电源时，Windows 8驱动程序可用它, 来决定如何切换到立即关电状态(immediate power down state)。

除了PortCls注册表电源设置中讨论的默认电源设置外，Windows 8还引入了一个新的注册表值，该值也位于相关驱动程序的PowerSettings注册表项中。

例如，如果您的驱动程序的键为<UVXYZ>，则可以在Windows注册表中的以下路径中找到该驱动程序的电源设置信息：

HKLM\System\CurrentControlSet\Control\Class\{4D36E96C-E325-11CE-BFC1-08002BE10318}\<UVXYZ>\PowerSettings.

除了[PortCls注册表电源设置](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/portcls-registry-power-settings)中显示的默认电源设置值之外，还可以像下行一样加上ImmediateIdle的值：

"ImmediateIdle"=hex:00,00,00,00

ImmediateIdle的数据类型为REG\_DWORD，其默认值为“0”，相当于FALSE。在前面的语句中，十六进制值“0”表示当不再需要电源时，设备不会立即断电。

For your driver to opt-in to an immediate power down state, when power is no longer needed, you must use the following syntax:

为了让您的驱动程序在不再需要电源时选择立即关闭电源，您需要使用以下语法：

"ImmediateIdle"=hex:01,00,00,00

这里的值"1" 相当于TRUE, 这表示当不再需要电源时，设备将立即断电.

当运行时电源管理框架调用回调函数**DevicePowerRequired**，指示设备不再需要电源时，PortCls为D状态(D-State, 用IdlePowerState注册表值表示)请求设备电源IRP(Device Power IRP)。如果未提供任何状态，则使用默认值D3。

如果驱动程序选择立即进行空闲电源管理，则必须确保系统的电源引擎插件（Power Engine Plug-in , PEP）包含必要的逻辑, 来避免因为要接收后续的IRP而对适配器不必要的反复的上下电。应该应用一些驻留(residency)规则，以便为批量的I / O请求保持设备通电。

此外，Windows 7中引入的新接口允许驱动程序以编程方式启用或禁用空闲电源管理，当驱动程序未选择立即进行空闲电源管理时，这一接口仍将受到尊重。这是通过**[IPortClsPower::SetIdlePowerManagement](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclspower-setidlepowermanagement)** 方法完成的，并且会覆盖注册表中的设置，除非ImmediateIdle设置为1（TRUE）。

## 3.3. PortCls 私有电源引擎插件(Power Engine Plug-in ,PEP) 上下文共享

从Windows 8开始，微型端口驱动程序(miniport driver)可以使用IPortClsRuntimePower（一种新接口）与电源引擎插件（PEP）进行私有上下文共享。

已经更新了音频端口类驱动程序（PortCls）来在WaveRT端口上支持这个新接口。 为了将私有电源控制发送到操作系统的PEP，微型端口驱动程序首先必须访问其相关端口的IPortClsRuntimePower接口。 然后，微型端口驱动程序注册在适当的时间调用的回调，允许微型端口驱动程序发送私有电源控制

### 3.3.1. 访问 IPortClsRuntimePower接口

微型端口驱动程序通过以下事件序列访问其端口的IPortClsRuntimePower接口:

1. 微型端口驱动程序调用**[PcNewPort](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-pcnewport)**并提供IID\_IPortWaveRT作为REFID。
2. **PcNewPort**创建 [IPortWaveRT](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iportwavert)类型的端口接口（Pport）。
3. 然后，微型端口驱动程序在新创建的**IPortWaveRT**端口接口中调用QueryInterface，并将IID\_IPortClsRuntimePower指定为接口GUID。
4. **IPortWaveRT**端口接口为微型端口驱动程序提供指向其 **[IPortClsRuntimePower](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iportclsruntimepower)**接口的指针。

Portcls.h头文件定义IPortClsRuntimePower的GUID，如下所示：

// {E057C351-0430-4DBC-B172-C711D40A2373}

DEFINE\_GUID(IID\_IPortClsRuntimePower,

0xe057c351, 0x430, 0x4dbc, 0xb1, 0x72, 0xc7, 0x11, 0xd4, 0xa, 0x23, 0x73);

### 3.3.2. 注册回调

微型端口驱动程序使用**[IPortClsRuntimePower::RegisterPowerControlCallback](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclsruntimepower-registerpowercontrolcallback)**方法来注册回调。 当PEP发起私有请求或响应由微型端口驱动程序本身发起的私有请求时，将调用此方法。 通常应在驱动程序处理IRP\_MN\_START\_DEVICE PNP Irp时执行回调注册。

除了回调中提供的Context指针之外，其他参数的定义与运行时电源框架的PowerControlCallback的定义完全相同。 此外，微型端口的回调必须是PCPFNRUNTIME\_POWER\_CONTROL\_CALLBACK类型，如Portcls.h头文件中的以下片段中所定义。

typedef

NTSTATUS

\_IRQL\_requires\_max\_(DISPATCH\_LEVEL)

(\*PCPFNRUNTIME\_POWER\_CONTROL\_CALLBACK)

(

\_In\_ LPCGUID PowerControlCode,

\_In\_opt\_ PVOID InBuffer,

\_In\_ SIZE\_T InBufferSize,

\_Out\_opt\_ PVOID OutBuffer,

\_In\_ SIZE\_T OutBufferSize,

\_Out\_opt\_ PSIZE\_T BytesReturned,

\_In\_opt\_ PVOID Context

);

When the driver is stopped or removed, it must use the **[IPortClsRuntimePower::UnregisterPowerControlCallback](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclsruntimepower-unregisterpowercontrolcallback)** method to unregister any registered callbacks.

当驱动程序被停止或删除时，它必须使用**[IPortClsRuntimePower::UnregisterPowerControlCallback](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclsruntimepower-unregisterpowercontrolcallback)**方法注销任何已注册的回调.

### 3.3.3. 发送私有电源控制

在微型端口建立对**IPortClsRuntimePower**接口的访问权限并使用接口的**RegisterPowerControlCallback**方法注册回调之后，现在它已准备好发送私有电源控制。 调用回调方法时，微型端口驱动程序使用[**IPortClsRuntimePower::SendPowerControl**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nf-portcls-iportclsruntimepower-sendpowercontrol)方法将私有电源控件发送到Windows PEP。

除DeviceObject参数外，所有其他参数的定义与运行时电源框架的[PoFxPowerControl](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/wdm/nf-wdm-pofxpowercontrol)方法的参数相同。

## 3.4. PortCls从D3状态退出的时延要求

本主题讨论Windows端口类驱动程序（PortCls）如何使用新的Windows 8接口来操作从D3睡眠状态退出的时延要求。

当系统进入除完全工作之外的平台电源状态（例如，睡眠或连接待机）时，可以放宽音频适配器返回D0（完全工作）状态所需的退出延迟。 这使得音频适配器可以使用比D3更深的睡眠状态，即使这些更深的状态可能导致更长的退出时延。

PortCls现在可以使用新的电源管理接口生成新的D3退出时延容差，然后动态地和音频微型端口驱动程序(audio miniport driver)沟通其值。 这些容差表示为[**PC\_EXIT\_LATENCY**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/ne-portcls-_pc_exit_latency)枚举值。

有关新电源管理接口的更多信息，请参阅[**IAdapterPowerManagement3**](https://docs.microsoft.com/windows-hardware/drivers/ddi/content/portcls/nn-portcls-iadapterpowermanagement3)