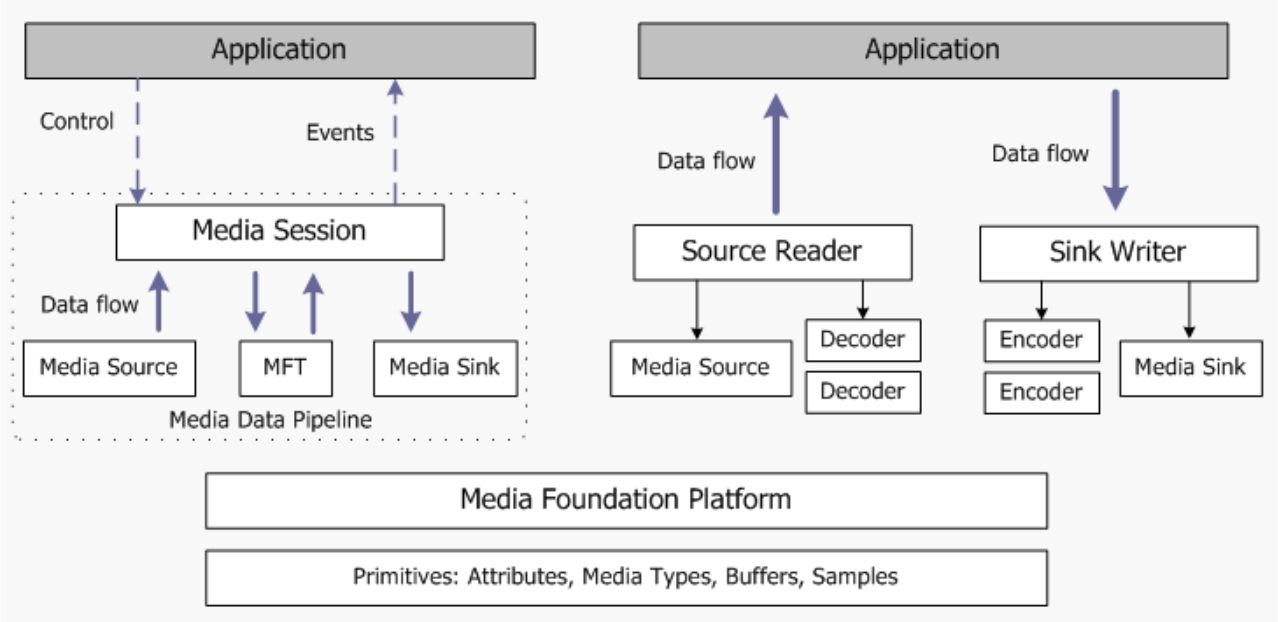
在 Windows 上，组件对象模型 (COM) 提供了一种将 API 与其实现分离的标准方法，这是任何插件模型的要求。由于这个原因（除其他外），Media Foundation 使用 COM 接口。 From here

MF基础架构

From here



MF提供两种不同的编程模型。第一个模型（如图左侧所示）使用端到端管道来处理媒体数据。应用程序初始化管道（例如，通过提供要播放的文件的 URL），然后调用方法来控制流式传输。在第二个模型中，如图右侧所示，应用程序要么从源提取数据，要么将其推送到目标（或两者）。如果您需要处理数据，此模型特别有用，因为应用程序可以直接访问数据流。

属性

属性是键/值对，其中键是 GUID，值是 PROPVARIANT。属性值仅限于以下数据类型：

- 无符号 32 位整数 (UINT32)。
- 无符号 64 位整数 (UINT64)。
- 64 位浮点数。
- GUID。
- 以 Null 结尾的宽字符字符串。
- 字节数组。
- IUnknown 指针。

要设置或检索属性值，请使用 IMFAttributes 接口。属性键在对象内是唯一的。

有关媒体基础属性的完整列表，请参阅Media Foundation Attributes

Implementing IMFAttributes提供了一个IMFAttributes序列化的封装

媒体类型

参考 Media Type Attributes

媒体类型描述媒体流的格式。在 Microsoft Media Foundation 中，媒体类型由 IMFMediaType 接口表示。该接口继承了IMFAttributes接口。媒体类型的详细信息被指定为属性。

要创建新的媒体类型，请调用 MFCreateMediaType 函数。该函数返回一个指向 IMFMediaType 接口的指针。媒体类型最初没有属性。要设置格式的详细信息，请设置相关属性。

主类型是 GUID，它定义媒体流中数据的总体类别。主要类型包括视频和音频。要指定主要类型，请设置 MF_MT_MAJOR_TYPE 属性。IMFMediaType::GetMajorType 方法返回此属性的值。

子类型进一步定义了格式。例如，在视频主要类型中，有 RGB-24、RGB-32、YUY2 等子类型。在音频中，有 PCM 音频、IEEE 浮点音频等。子类型比主要类型提供更多信息，但它没有定义有关格式的所有信息。例如，视频子类型不定义图像大小或帧速率。要指定子类型，请设置 MF_MT_SUBTYPE 属性。

所有媒体类型都应具有主要类型 GUID 和子类型 GUID。有关主要类型和子类型 GUID 的列表，请参阅媒体类型 GUID。

与 DirectShow 和 Windows Media Format SDK 等先前技术中使用的格式结构相比，属性具有多个优点。

视频格式

视频格式通常由 FOURCC 或 D3DFORMAT 值表示。保留一系列 GUID 用于将这些值表示为子类型。这些 GUID 的格式为 XXXXXXXX-0000-0010-8000-00AA00389B71，其中 XXXXXXXX 是 4 字节 FOURCC 代码或 D3DFORMAT 值。

可以使用这里的代码在调试时查看媒体类型的内容。

Media Buffers 媒体缓冲区

媒体缓冲区是一个管理内存块的 COM 对象，通常用于保存媒体数据。媒体缓冲区用于将数据从一个管道组件移动到下一个管道组件。**大多数应用程序不直接使用媒体缓冲区**，因为媒体会话处理管道对象之间的所有数据流。如果您正在编写自己的管道组件，或者如果您在没有媒体会话的情况下直接使用管道组件，则必须使用媒体缓冲区。媒体缓冲区暴漏一个 IMFMediaBuffer 接口。该接口设计用于读取或写入任何类型的数据。未压缩的视频帧需要特殊处理，因为它们可能存储在视频内存中的 Direct3D 表面中。

Media Samples 媒体样本

媒体样本是包含零个或多个缓冲区的有序列表的对象。媒体样本暴漏一个 IMFSample 接口。一个样本中包含的数据量取决于创建该样本的组件以及缓冲区中的数据类型。对于未压缩的视频，样本通常包含单个视频帧。对于未压缩的音频，数据量可能会有所不同，但通常音频帧不会跨越两个样本。对于压缩数据，这些准则可能不适用。

要创建新的媒体样本，请调用 MFCreateSample 函数

Media Foundation Platform APIs

在使用任何 Microsoft Media Foundation 对象或接口之前，必须调用 MFStartup 函数。传入常量 MF_VERSION。MFStartup 函数初始化 Media Foundation 平台。如果 MFStartup 返回 MF_E_BAD_STARTUP_VERSION，则意味着您的应用程序是使用与系统上的 Media Foundation DLL 不匹配的 Media Foundation 标头版本进行编译的。

对于每次调用 MFStartup，您的应用程序都必须调用 MFShutdown。

Microsoft Media Foundation 使用 COM 结构的混合，但不是完全基于 COM 的 API。

在 Media Foundation 中，异步处理和回调由工作队列处理。工作队列始终具有多线程单元 (MTA) 线程，因此如果应用程序也在 MTA 线程上运行，那么它的实现将会更简单。因此，建议使用 COINIT_MULTITHREADED 标志调用 CoInitializeEx。

Media Foundation 不会将单线程单元 (STA) 对象编组到工作队列线程。它也不保证 STA 不变量得到维护。因此，STA 应用程序必须小心，不要将 STA 对象或代理传递给 Media Foundation API。Media Foundation 不支持仅 STA 的对象。

如果您有 MTA 或自由线程对象的 STA 代理，则可以使用工作队列回调将该对象封送到 MTA 代理。CoCreateInstance 函数可以返回原始指针或 STA 代理，具体取决于注册表中为该 CLSID 定义的对象模型。如果返回 STA 代理，则不得将指针传递给 Media Foundation API。

尽管并非所有 Media Foundation 对象都是 COM 对象，但所有 Media Foundation 接口均派生自 IUnknown。因此，所有媒体基础对象都必须根据 COM 规范实现 IUnknown，包括引用计数和 QueryInterface 的规则。所有引用计数对象还应确保 DllCanUnloadNow 不允许在对象仍然存在时卸载模块。

媒体基础组件不能是 STA 对象。许多媒体基础对象根本不需要是 COM 对象。但如果是的话，它们就不能在 STA 中运行。所有媒体基础组件都必须是线程安全的。一些媒体基础对象也必须是自由线程或单元中立的。

异步回调方法

在 Media Foundation 中，许多操作都是异步执行的。异步操作可以提高性能，因为它们不会阻塞调用线程。Media Foundation 中的异步操作由一对具有命名约定 Begin... 和 End..... 的方法定义，这两个方法一起定义流上的异步读取操作。
要启动异步操作，请调用 Begin 方法。如果Begin方法返回失败代码，则意味着操作立即失败，并且不会调用回调。如果 Begin 方法成功，则意味着操作正在挂起，操作完成时将调用回调。回调方法为 IMFAsyncCallback::Invoke。它有一个参数 pAsyncResult，它是指向 IMFAsyncResult 接口的指针。End 方法的返回值指示异步操作的状态。
一个异步回调的实现代码，参考这里。
如果调用多个异步方法，则每个方法都需要单独实现 IMFAsyncCallback::Invoke。但是，您可能希望在单个 C++ 类中实现回调。该类只能有一个 Invoke 方法，因此一种解决方案是提供一个帮助器类，将 Invoke 调用委托给容器类上的另一个方法。参考这里。

Work Queues工作队列

在 Microsoft Media Foundation 中，工作队列是在另一个线程上执行异步操作的有效方法。从概念上讲，您将工作项放入队列中，队列有一个线程从队列中提取每个项目并调度它。使用 IMFAsyncCallback 接口将工作项实现为回调。
Media Foundation 创建了多个标准工作队列，称为平台工作队列。应用程序还可以创建自己的工作队列，称为私有工作队列。有关平台工作队列的列表，请参阅工作队列标识符。要创建私有工作队列，请调用MFAllocateWorkQueue。该函数返回新工作队列的标识符。要将项目放入队列中，请调用 MFPutWorkItem 或 MFPutWorkItemEx。在这两种情况下，您都必须指定回调接口。
工作队列线程始终在调用者的进程中创建。在每个工作队列中，回调都是序列化的。如果使用同一工作队列调用 MFPutWorkItem 两次，则在第一个回调返回之前不会调用第二个回调。
细节可以参考Work Queues

Media Event Generators媒体事件生成器

MF为对象发送事件提供了一致的方式。对象可以使用事件来指示异步方法的完成，或者通知应用程序对象状态的更改。
如果对象发送事件，它暴漏了一个 IMFMediaEventGenerator 接口。IMFMediaEventGenerator的实现可以参考这里。
事件由 IMFMediaEvent 接口表示。

Service Interfaces服务接口

Media Foundation 中的某些接口必须通过调用 IMFGetService::GetService 来获取，而不是通过调用 QueryInterface。GetService 方法的工作方式与 QueryInterface 类似，但有以下区别：

- 除了接口标识符之外，它还需要一个服务标识符 GUID。
- 它可以返回指向实现该接口的另一个对象的指针，而不是返回指向被查询的原始对象的指针。

MFGetService 函数是一个辅助函数，它在对象中查询 IMFGetService，然后调用该对象的 GetService 方法。

Activation Objects激活对象

激活对象是一个辅助对象，用于创建另一个对象，有点类似于类工厂。激活对象暴漏一个 IMFActivate 接口。

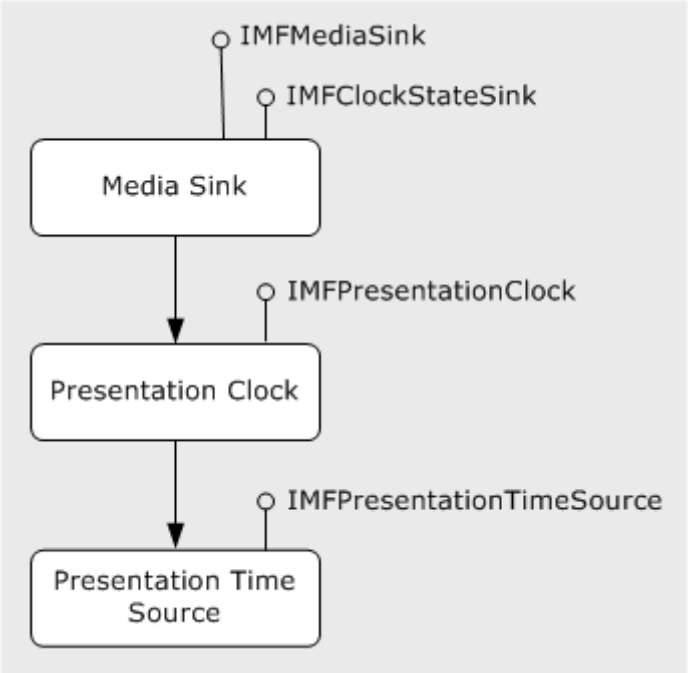
激活对象允许您**推迟目标对象的创建**，因为您可以保留 IMFActivate 指针而不创建目标对象。激活对象也可以被序列化，从而用于在另一个进程中创建目标对象。例如，激活对象用于将管道组件从应用程序进程编组到受保护的媒体路径 (PMP) 进程。某些返回 IMFActivate 指针列表的枚举函数也使用激活对象。在应用程序创建目标对象之前，它可以通过检查激活对象的属性来获取有关该对象的信息。

Presentation Clock 演示时钟 / 渲染时钟

演示时钟是生成演示时钟时间的对象。演示时钟报告的时间称为演示时间。演示中的所有流都与演示时间同步。

要创建演示时钟，请调用 MFCreatePresentationClock。要关闭时钟，请查询 IMFShutdown 接口并调用 IMFShutdown::Shutdown。MFCreatePresentationClock的调用者负责调用Shutdown；在大多数情况下，这是媒体会话而不是应用程序。

演示时钟实际上并没有实现时钟。相反，它从另一个对象（称为演示时间源）获取时钟时间。时间源可以是生成准确时钟滴答并公开 IMFPresentationTimeSource 接口的任何对象。下图显示了此过程。



首次创建演示时钟时，它没有时间源。要设置时间源，请使用指向时间源的 IMFPresentationTimeSource 接口的指针调用 IMFPresentationClock::SetTimeSource。时间源支持与演示时钟相同的状态（运行、暂停和停止），并且必须实现 IMFClockStateSink 接口。演示时钟使用此接口通知时间源何时更改状态。这样，时间源提供时钟滴答声，但呈现时钟启动时钟中的状态变化。

某些媒体接收器可以访问准确的时钟，因此公开 IMFPresentationTimeSource 接口。特别地，音频渲染器可以使用声卡的频率作为时钟。在音频播放中，音频渲染器充当时间源非常有用，以便视频与音频播放速率同步。这通常比尝试将音频与外部时钟匹配产生更好的结果。

Media Foundation 还提供基于系统时钟的演示时间源。要创建此对象，请调用 MFCreateSystemTimeSource。当没有媒体接收器提供时间源时，可以使用系统时间源。

通常，媒体接收器必须使用提供给它的呈现时钟，无论演示时钟使用哪个时间源。即使媒体接收器实现 IMFPresentationTimeSource，此规则也适用。如果演示时钟使用其他时间源，则媒体接收器必须遵循该时间源，而不是其自己的内部时钟。

在两种情况下媒体接收器不会遵循演示时钟：

- 有些媒体接收器是无速率的。如果媒体接收器是无速率的，它会尽快消耗样本，而不根据呈现时钟来调度它们。通常，无速率接收器将数据写入文件，因此希望尽快完成操作。无速率接收器在其 IMFMediaSink::GetCharacteristics 方法中返回 MEDIASINK_RATELESS 标志。当拓扑中的所有接收器都是无速率时，媒体会话会尽快通过管道推送数据。
- 某些媒体接收器无法将速率与除自身之外的时间源相匹配。如果是这样，接收器将在其 GetCharacteristics 方法中返回 MEDIASINK_CANNOT_MATCH_CLOCK 标志。管道仍然可以使用另一个时间源，但结果将不是最佳的。接收器可能会落后并在播放过程中导致故障。

Media Foundation Pipeline MF管道

Media Sources媒体来源

媒体源是在MF管道中生成媒体数据的对象。

Media Source Object Model媒体源对象模型

Microsoft Media Foundation 中媒体源的对象模型。媒体源必须实现两个对象：

- 演示描述符，描述源的内容，包括流的数量和每个流的格式。有关表示描述符的更多信息，请参阅演示描述符。
- 一个或多个媒体流，生成源数据。

在播放开始之前，源不会创建流。

媒体源（Media Source）必须通过 QueryInterface 暴漏以下接口：**IMFMediaSource**， **IMFMediaEventGenerator**（IMFMediaSource 接口继承此接口）

媒体源可以选择性地实现 IMFGetService 接口并将以下服务接口：

- **IMFRateControl** 控制播放速率。
- **IMFRateSupport** 报告支持的播放速率范围。
- **IMFQualityAdvise** 调整音频或视频质量。
- **IMFMetadataProvider** 提供元数据。

如果媒体源可以以正常速度 (1.0) 以外的速率播放，则应公开速率控制服务（IMFRateControl 和 IMFRateSupport）。否则，假定源仅支持正常速度播放。有关详细信息，请参阅实现速率控制。

媒体流（Media streams）必须实现以下接口。

- **IMFMediaStream**
- **IMFMediaEventGenerator**

目前还没有为媒体流定义服务接口。

Presentation Descriptors 演示描述符

一个Presentation是公用一个演示时间的一组相关媒体流。例如，Presentation可能包含电影中的音频和视频流。Presentation Descriptors是包含特定呈现的描述的对象。用于配置媒体源和一些媒体接收器。

每个Presentation Descriptors包含一个或多个流描述符（stream descriptors）的列表。这些描述了Presentation中的流。可以选择或取消选择流。只有选定的流才会产生数据。取消选择的流不活动并且不产生任何数据。每个流描述符都有一个媒体类型处理程序，用于更改流的媒体类型或获取流的当前媒体类型。（有关媒体类型的更多信息，请参阅媒体类型。）

- **IMFPresentationDescriptor** Presentation Descriptors 演示描述符
- **IMFStreamDescriptor** 流描述符
- **IMFMediaTypeHandler** 媒体类型处理程序

Media Source Presentations 媒体源描述符

每个媒体源都提供一个表示描述符，用于描述源的默认配置。在默认配置中，至少选择一个流，并且每个选定的流都有一种媒体类型。要获取演示描述符，请调用 IMFMediaSource::CreatePresentationDescriptor。该方法返回 IMFPresentationDescriptor 指针。

您可以修改源的表示描述符以选择不同的流集。除非媒体源停止，否则不要修改呈现描述符。当您调用 IMFMediaSource::Start 启动源时，将应用更改。

要获取流的数量，请调用 IMFPresentationDescriptor::GetStreamDescriptorCount。要获取流的流描述符，请调用 IMFPresentationDescriptor::GetStreamDescriptorByIndex 并传入流的索引。流从零开始索引。GetStreamDescriptorByIndex 方法返回 IMFStreamDescriptor 指针。它还返回一个布尔标志，指示是否选择了流。如果选择了该流，则媒体源会为该流生成数据。否则，源不会为该流生成任何数据。要选择流，请使用流的索引调用 IMFPresentationDescriptor::SelectStream。要取消选择流，请调用 IMFPresentationDescriptor::DeselectStream。

Media Type Handlers媒体类型处理程序

要更改流的媒体类型或获取流的当前媒体类型，请使用流描述符的媒体类型处理程序。调用 IMFStreamDescriptor::GetMediaTypeHandler 获取媒体类型处理程序。此方法返回 IMFMediaTypeHandler 指针。

如果您只是想知道流中的数据类型，例如音频或视频，请调用 IMFMediaTypeHandler::GetMajorType。此方法返回主要媒体类型的 GUID。例如，播放应用程序通常将音频流连接到音频渲染器，将视频流连接到视频渲染器。如果您使用媒体会话或拓扑加载器来构建拓扑，则主要类型 GUID 可能是您需要的唯一格式信息。

如果您的应用程序需要有关当前格式的更多详细信息，请调用 IMFMediaTypeHandler::GetCurrentMediaType。此方法返回指向媒体类型的 IMFMediaType 接口的指针。使用此接口获取格式的详细信息。

媒体类型处理程序还包含流支持的媒体类型列表。要获取列表的大小，请调用 IMFMediaTypeHandler::GetMediaTypeCount。要从列表中获取媒体类型，请使用媒体类型的索引调用 IMFMediaTypeHandler::GetMediaTypeByIndex。媒体类型按照大致的偏好顺序返回。例如，对于音频格式，较高的采样率优于较低的采样率。但是，没有明确的规则来管理排序，因此您应该将其简单地视为指南。

支持的类型列表可能不包含流支持的所有可能的媒体类型。要测试是否支持特定媒体类型，请调用 IMFMediaTypeHandler::IsMediaTypeSupported。要设置媒体类型，请调用 IMFMediaTypeHandler::SetCurrentMediaType。如果该方法成功，流将包含符合指定格式的数据。SetCurrentMediaType 方法不会更改首选类型列表。

Media Source Events媒体源事件

媒体源和媒体流发送的事件。具体参考这里。

Writing a Custom Media Source编写自定义媒体源

参考这里

一个实例：Case Study: MPEG-1 Media Source

Source Resolver源解析器

源解析器获取 URL 或字节流并为该内容创建适当的媒体源。源解析器是应用程序创建媒体源的标准方法。

在内部，源解析器使用处理程序对象：

- **Scheme handlers** 方案处理程序采用 URL 并创建媒体源或字节流
- **Byte stream handlers** 字节流处理程序获取字节流并创建媒体源。

要创建源解析器，请调用 MFCreateSourceResolver。该函数返回 IMFSourceResolver 接口指针。

源解析器有同步和异步两种方法。如果您在主应用程序线程中使用源解析器，则异步方法将使您的用户界面响应更快。同步方法可能会阻塞相当长的时间，特别是在源解析器必须打开网络资源的情况下。参考这里。

Configuring a Media Source配置媒体源

当您使用源解析器创建媒体源时，您可以指定包含配置属性的属性存储。这些属性将用于初始化媒体源。支持的属性集取决于媒体源的实现。并非每个媒体源都定义配置属性。

要配置源，请执行以下步骤。

- 调用 PSCreateMemoryPropertyStore 创建新的属性存储。该函数返回一个 IPropertyStore 指针。
- 调用 IPropertyStore::SetValue 设置一个或多个配置属性。
- 调用源解析器的创建函数之一，例如 IMFSourceResolver::CreateObjectFromURL，并在 pProps 参数中传递 IPropertyStore 指针。

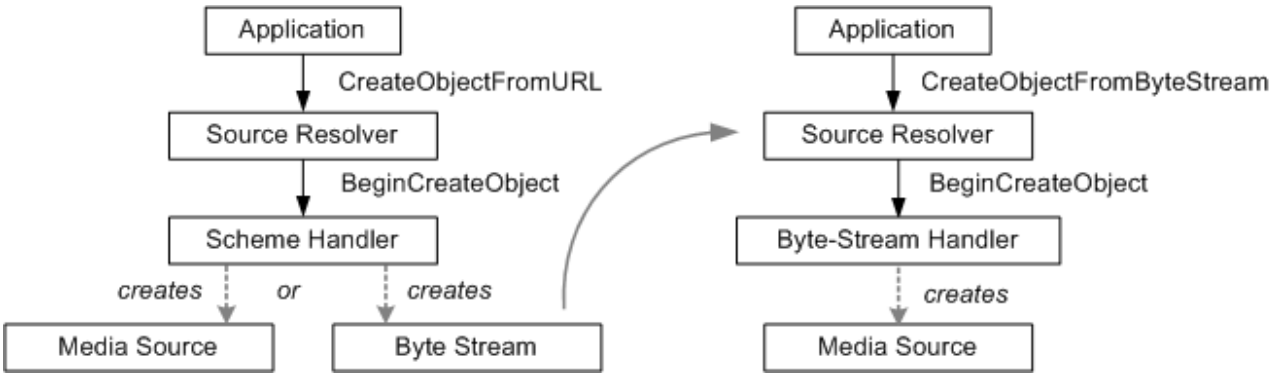
源解析器将 IPropertyStore 指针直接传递到创建源的方案处理程序或字节流处理程序。源解析器不会尝试验证属性。通常，这些属性用于高级设置。如果您不提供属性存储，媒体源仍应使用默认设置正常运行。

Scheme Handlers and Byte-Stream Handlers方案处理程序和字节流处理程序

源解析器如何创建媒体源的内部详细信息。

源解析器可以从 URL 或字节流（即 IMFByteStream 指针）创建媒体源。为此，它使用称为处理程序的辅助对象。对于 URL，源解析器使用方案处理程序。对于字节流，它使用字节流处理程序。

方案处理程序（scheme handler）将 URL 作为输入并创建媒体源或字节流。如果它创建字节流，源解析器会将其传递给字节流处理程序，该处理程序创建媒体源。下图说明了此过程。



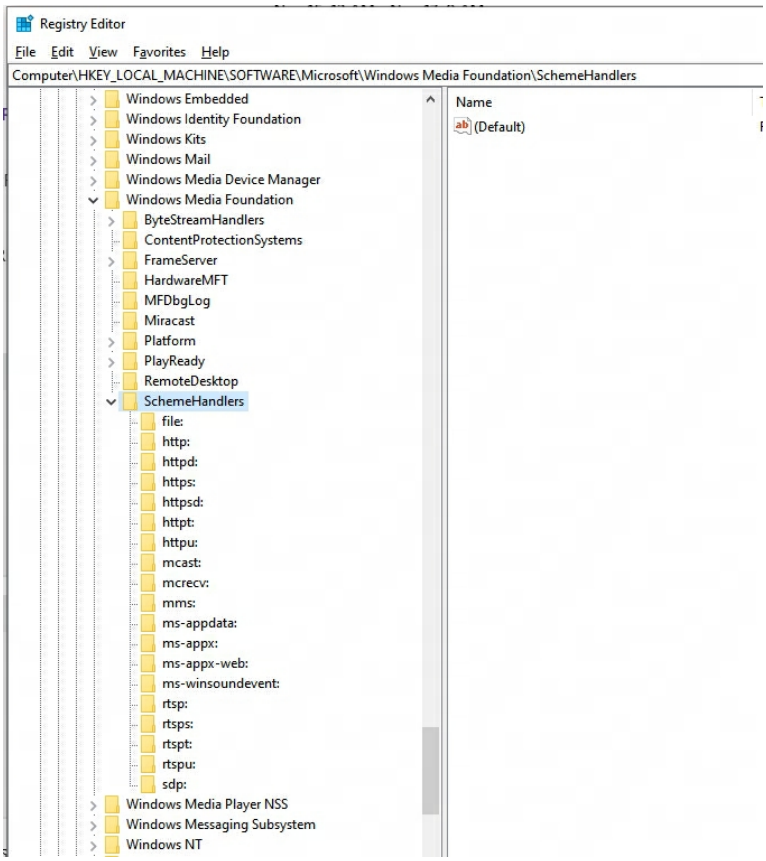
Scheme Handlers方案处理程序

当应用程序调用 IMFSourceResolver::CreateObjectFromURL 或其异步等效项 BeginCreateObjectFromURL 时，将使用方案处理程序。

源解析器在注册表中查找方案处理程序。方案处理程序按 URL 方案列出，位于以下键下：

HKEY_CURRENT_USER
  Software
    Microsoft
      Windows Media Foundation
        SchemeHandlers
          <scheme>
            {00000000-0000-0000-0000-000000000000} = REG_SZ
HKEY_LOCAL_MACHINE
  Software
    Microsoft
      Windows Media Foundation
        SchemeHandlers
          <scheme>
            {00000000-0000-0000-0000-000000000000} = REG_SZ

在**<scheme>那**里是处理程序要解析的 URL 方案。该方案包括尾随“:”字符；例如，“http：”。比如我的电脑上：



要注册新的方案处理程序，请添加一个名称为方案处理程序的 CLSID 的条目，采用规范字符串形式：{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}。该条目的值是一个字符串 (REG_SZ)，其中包含处理程序的简短描述，例如“My Scheme Handler.”。该条目的重要部分是 CLSID。源解析器通过使用此 CLSID 调用 CoCreateInstance 来创建处理程序。

方案处理程序暴漏一个 IMFSchemeHandler 接口。如果源解析器找到与 URL 方案匹配的方案处理程序，源解析器将调用 IMFSchemeHandler::BeginCreateObject，并传入原始 URL。方案处理程序将打开 URL 并尝试解析内容。此时，方案处理程序有两个选择：

- 创建媒体源。
- 创建字节流。

如果它创建了媒体源，则源解析器会将媒体源返回给应用程序。如果它创建字节流，源解析器会尝试查找适当的字节流处理程序，如下一节所述。

Byte-Stream Handlers 字节流处理程序

当应用程序调用 IMFSourceResolver::CreateObjectFromByteStream 或其异步等效方法 BeginCreateObjectFromByteStream 时，将使用字节流处理程序。当方案处理程序返回字节流时也会使用它们，如前所述。

与方案处理程序一样，字节流处理程序也在注册表中列出。它们按文件扩展名或 MIME 类型（或两者）列出，位于以下键下：

```
HKEY_CURRENT_USER
  Software
    Microsoft
```

Windows Media Foundation
ByteStreamHandlers
<ExtensionOrMimeType>
{00000000-0000-0000-0000-000000000000} = REG_SZ
HKEY_LOCAL_MACHINE
Software
Microsoft
Windows Media Foundation
ByteStreamHandlers
<ExtensionOrMimeType>
{00000000-0000-0000-0000-000000000000} = REG_SZ

在 **<ExtensionOrMimeType>**那里是文件扩展名或 MIME 类型。文件扩展名包括开头的“.”特点;例如，“.wmv”。比如我的电脑上：

文件扩展名是 URL 的一部分，由应用程序提供。MIME 类型可以通过字节流上的 MF_BYTESTREAM_CONTENT_TYPE 属性获得。
要注册新的字节流处理程序，请添加一个名称为处理程序的 CLSID（采用规范字符串形式）的条目。该条目的值是一个字符串 (REG_SZ)，其中包含处理程序的简短描述，例如“My Byte-Stream Handler”。源解析器调用 CoCreateInstance 从 CLSID 创建处理程序。您可以在多个扩展名或 MIME 类型下注册相同的处理程序。
字节流处理程序公开 IMFByteStreamHandler 接口。如果源解析器找到匹配的字节流处理程序，它将调用 IMFByteStreamHandler::BeginCreateObject。此方法的输入是指向字节流的指针，加上原始 URL（如果可用）。字节流处理程序从字节流中读取数据，直到解析足够的数据来创建媒体源。

Media Foundation Transforms 转换

About MFTs关于 MFT

MF Transform (MFT) 提供了处理媒体数据的通用模型。MFT 用于解码器、编码器和数字信号处理器 (DSP)。简而言之，位于媒体源和媒体接收器之间的媒体管道中的任何内容都是 MFT。
对于大多数应用程序，MFT 数据处理的细节被MF架构的更高层隐藏。许多MF应用程序永远不会直接调用 MFT。但是，当然可以直接在您的应用程序中托管 MFT。
MFT 是首次随 DirectX 媒体对象 (DMO) 引入的转换模型的演变。事实上，创建支持这两种模型的转换相对容易。与 DMO 相比，MFT 所需的行为更加明确，这使得编写正确的实现变得更加容易。此外，MFT 还可以支持硬件加速视频处理。
下面简要概述了 MFT 处理模型，重点关注**整体设计**而不是具体的方法调用。有关更详细的分步说明，请参阅基本 MFT 处理模型。

Streams流

MFT 具有输入流和输出流。输入流接收数据，输出流产生数据。例如，解码器具有一个接收编码数据的输入流和一个产生解码数据的输出流。

MFT 上的流不表示为不同的 COM 对象。相反，每个流都有一个指定的流标识符，并且 IMFTransform 接口中的方法将流标识符作为输入参数。

一些 MFT 具有固定数量的流。例如，解码器和编码器通常只有一个输入和一个输出。其他 MFT 具有动态数量的流。如果MFT支持动态流，则客户端可以添加新的输入流。客户端无法添加输出流，但 MFT 可能在处理过程中添加或删除输出流。例如，多路复用器通常允许客户端添加输入流并为多路复用流提供一个输出。解复用器则相反，具有一个输入，但具有动态数量的输出流，具体取决于输入流的内容。下图显示了多路复用器和多路分解器之间的区别。

Media Types媒体类型

首次创建 MFT 时，没有任何流具有已建立的格式。在 MFT 可以处理数据之前，客户端必须设置流的格式。例如，对于解码器，输入格式是原始源文件中使用的压缩格式，输出格式是未压缩格式，例如PCM音频或RGB视频。流格式使用媒体类型来描述。

根据 MFT 的内部状态，它可能会为每个流提供可能的媒体类型列表。您可以在设置媒体类型时使用此列表作为提示。在一个流上设置媒体类型可以更改另一流的可能类型列表。例如，解码器通常无法提供任何输出类型，直到客户端设置输入类型。输入类型包含解码器返回可能的输出类型列表所需的信息。

要设置流上的媒体类型，请调用 IMFTransform::SetInputType 或 IMFTransform::SetOutputType。要获取流的可能媒体类型列表，请调用 IMFTransform::GetInputAvailableType 或 IMFTransform::GetOutputAvailableType。

Processing Data处理数据

客户端在流上设置媒体类型后，MFT 就准备好处理数据。为了实现这一点，客户端交替向 MFT 提供输入数据并从 MFT 获取输出数据：

- 要将输入数据提供给 MFT，请调用 IMFTransform::ProcessInput。
- 要从 MFT 中提取输出数据，请调用 IMFTransform::ProcessOutput。

ProcessInput 方法传入指向客户端分配的媒体样本的指针。媒体样本包含一个或多个缓冲区，每个缓冲区都包含供 MFT 处理的输入数据。

ProcessOutput 方法支持两种不同的分配模型：MFT 分配输出缓冲区，或者客户端分配输出缓冲区。一些 MFT 支持这两种分配模型，但 MFT 并不需要同时支持这两种模型。例如，MFT 可能要求客户端分配输出缓冲区。IMFTransform::GetOutputStreamInfo 方法返回有关输出流的信息，包括 MFT 支持的分配模型。

MFT 旨在缓冲尽可能少的数据，以最大限度地减少管道中的延迟。因此，在任何给定时间，MFT 都可以发出以下条件之一的信号：

- MFT 需要更多的输入数据。在此状态下，MFT 无法生成输出，直到客户端至少调用一次 ProcessInput。
- 在客户端至少调用一次 ProcessOutput 之前，MFT 将不再接受任何输入。

例如，假设您正在使用视频解码器来解码包含关键帧和增量帧混合的视频流。最初，MFT 需要一些输入才能解码任何帧。客户端调用 ProcessInput 来传递第一帧。假设第一帧是增量帧（如下图中所示，“P”表示预测帧）。解码器保留该帧，但在获得下一个关键帧之前无法产生任何输出。

客户端继续调用 ProcessInput 并最终到达下一个关键帧（在下图中显示为帧内编码帧的“I”）。现在解码器有足够的帧来开始解码。此时它停止接受输入，客户端必须调用 ProcessOutput 来获取解码后的帧。

对于客户端来说，最简单的方法就是交替调用 ProcessInput 和 ProcessOutput。基本 MFT 处理模型主题中描述了更复杂的算法。

Basic MFT Processing Model基本MFT处理模型

客户端如何使用媒体基础转换 (MFT) 来处理数据。客户端是直接调用 MFT 上的方法的任何东西。这可能是应用程序或媒体基础管道。

创建 MFT

创建 MFT 有多种方法：

- 调用 MFTEnum 函数。
- 调用 MFTEnumEx 函数。
- 如果您已经知道 MFT 的 CLSID，只需调用 CoCreateInstance 即可。

一些 MFT 可能提供其他选项，例如专门的创建功能。

Get Stream Identifiers获取流标识符

MFT 具有一个或多个流。输入流接收输入数据，输出流生成输出数据。流不表示为不同的对象。相反，各种 MFT 方法将流标识符作为参数。

某些 MFT 允许客户端添加或删除输入流。在流式传输期间，MFT 可以添加或删除输出流。（客户端无法添加或删除输出流。）

- （可选）调用 IMFTransform::GetStreamLimits 获取 MFT 可以支持的最小和最大流数。如果最小值和最大值相同，则 MFT 具有固定数量的流。
- 调用 IMFTransform::GetStreamCount 获取初始流数。
- 调用 IMFTransform::GetStreamIDs 获取流标识符。如果该方法返回E_NOTIMPL，则表示MFT具有固定数量的流，并且流标识符从零开始连续。
- （可选。）如果 MFT 没有固定数量的流，请调用 IMFTransform::AddInputStreams 添加更多输入

流，或调用 IMFTransform::DeleteInputStream 删除输入流。（您无法添加或删除输出流。）

Set Media Types设置媒体类型

在 MFT 可以处理数据之前，客户端必须为 MFT 的每个流设置媒体类型。MFT 可能要求客户端在设置输出类型之前设置输入类型，或者可能需要相反的顺序（首先是输出类型）。有些 MFT 对订单没有要求。

MFT 可以提供流的首选媒体类型列表。此外，MFT 可以通过将此信息添加到注册表来指示它们支持的通用格式。

要设置媒体类型，请执行以下操作：

1. （可选）对于每个输入流，调用 IMFTransform::GetInputAvailableType 以获取该流的首选类型列表。如果该方法返回MF_E_TRANSFORM_TYPE_NOT_SET，则必须先设置输出类型；跳到步骤 3。如果该方法返回 E_NOTIMPL，则 MFT 没有首选输入类型列表；跳至步骤 2。
2. 对于每个输入流，调用 IMFTransform::SetInputType 设置输入类型。您可以使用步骤 1 中的媒体类型，或描述输入数据的类型。如果任何流返回 MF_E_TRANSFORM_TYPE_NOT_SET，请跳到步骤 3。
3. （可选）对于每个输出流，调用 IMFTransform::GetOutputAvailableType 以获取该流的首选类型列表。如果该方法返回MF_E_TRANSFORM_TYPE_NOT_SET，则必须先设置输入类型；返回步骤 1。如果任何流返回E_NOTIMPL，则MFT没有首选输出类型列表；跳至步骤 4。
4. 对于每个输出流，调用 IMFTransform::SetOutputType 设置输出类型。您可以使用步骤 3 中的媒体类型，或描述所需输出格式的类型。
5. 如果任何输入流没有媒体类型，请返回步骤 1。

Get Buffer Requirements获取缓冲区要求

客户端设置媒体类型后，应该获取每个流的缓冲区要求：

- 对于每个输入流，调用 IMFTransform::GetInputStreamInfo。
- 对于每个输出流，调用 IMFTransform::GetOutputStreamInfo。

Process Data处理数据

MFT 被设计为可靠的状态机。它不会向客户端发出任何回调。

- 使用 MFT_MESSAGE_NOTIFY_BEGIN_STREAMING 消息调用 IMFTransform::ProcessMessage。该消息请求 MFT 在流传输期间分配其所需的任何资源。
- 在至少一个输入流上调用 IMFTransform::ProcessInput 以将输入样本传送到 MFT。
- （可选）调用 IMFTransform::GetOutputStatus 查询MFT是否可以生成输出样本。如果该方法返回 S_OK，请检查 pdwFlags 参数。如果 pdwFlags 包含 MFT_OUTPUT_STATUS_SAMPLE_READY 标志，请转到步骤 4。如果 pdwFlags 为零，请返回到步骤 2。如果该方法返回 E_NOTIMPL，请转到步骤 4。
- 调用 IMFTransform::ProcessOutput 获取输出数据。

    - 如果该方法返回**MF_E_TRANSFORM_NEED_MORE_INPUT**，则表示MFT需要更多的输入数据；返回步骤2。
    - 如果该方法返回**MF_E_TRANSFORM_STREAM_CHANGE**，则表示输出流的数量发生了变化，或者输出格式发生了变化。客户端可能需要查询新的流标识符或设置新的媒体类型。有关更多信息，请参阅 ProcessOutput 的文档。
- 如果仍有输入数据需要处理，则转至步骤 2。如果 MFT 已消耗完所有可用的输入数据，则转至步骤 6。
- 使用 MFT_MESSAGE_NOTIFY_END_OF_STREAM 消息调用 ProcessMessage。
- 使用 MFT_MESSAGE_COMMAND_DRAIN 消息调用 ProcessMessage。
- 调用 ProcessOutput 获取剩余的输出。重复此步骤，直到该方法返回 MF_E_TRANSFORM_NEED_MORE_INPUT。该返回值表明所有输出已从 MFT 中耗尽。（不要将此视为错误情况。）

此处描述的流程在 MFT 中保留尽可能少的数据。每次调用 ProcessInput 后，客户端都会尝试获取输出。可能需要多个输入样本来产生一个输出样本，或者单个输入样本可能生成多个输出样本。客户端的最佳行为是从 MFT 中提取输出样本，直到 MFT 需要更多输入。

然而，MFT 应该能够处理客户端不同顺序的方法调用。例如，客户端可能只是交替调用 ProcessInput 和 ProcessOutput。当 MFT 有一些输出要产生时，MFT 应该通过从 ProcessInput 返回 MF_E_NOTACCEPTING 来限制它获得的输入量。

此处描述的方法调用顺序并不是唯一有效的事件顺序。例如，步骤 3 和 4 假设客户端从输入类型开始，然后尝试输出类型。客户端也可以颠倒此顺序并从输出类型开始。在任一情况下，如果 MFT 需要相反的顺序，则应返回错误代码 MF_E_TRANSFORM_TYPE_NOT_SET。

客户端可以在流式传输期间随时调用信息方法，例如 GetInputCurrentType 和 GetOutputStreamInfo。客户端还可以随时尝试更改媒体类型。如果这不是有效操作，MFT 应返回错误代码。简而言之，除了调用本身中记录的内容之外，MFT 应该对操作顺序做很少的假设。

下图显示了本主题中描述的过程的流程图。

Extensions to the Basic Model基本模型的扩展

MFT 可以支持基本流模型的一些扩展。

- 延迟读取流。如果 IMFTransform::GetOutputStreamInfo 方法返回输出流的 MFT_OUTPUT_STREAM_LAZY_READ 标志，则客户端不必从该输出流收集数据。MFT 继续接受输入，并且在某个时刻 MFT 将丢弃该流中的输出数据。如果所有输出流都有此标志，则 MFT 将永远不会无法接受输入。一个示例可能是可视化转换，其中客户端仅在有空闲 CPU 周期来绘制可视化时才获取输出。

- 可丢弃的流。如果 GetOutputStreamInfo 方法返回输出流的 MFT_OUTPUT_STREAM_DISCARDABLE 标志，则客户端可以请求 MFT 丢弃输出，但除非请求，否则 MFT 不会丢弃任何输出。当 MFT 达到其最大输入缓冲区时，客户端必须收集一些输出数据或请求 MFT 丢弃输出。
- 可选流。如果 GetOutputStreamInfo 方法返回输出流的 MFT_OUTPUT_STREAM_OPTIONAL 标志，或者 IMFTransform::GetInputStreamInfo 方法返回输入流的 MFT_INPUT_STREAM_OPTIONAL 标志，则该流是可选的。客户端不必在流上设置媒体类型。如果客户端未设置类型，则取消选择该流。取消选择的输出流不会生成样本，并且客户端在调用 ProcessOutput 时不会为该流提供缓冲区。取消选择的输入流不接受输入数据。MFT 可以将其所有输入和输出流标记为可选。然而，预计必须至少选择一个输入和一个输出才能使 MFT 工作。
- 异步处理。异步处理模型是在 Windows 7 中引入的。它在主题异步 MFT 中进行了描述。

IMF2DBuffer

如果 MFT 处理未压缩的视频数据，则应使用 IMF2DBuffer 接口来操作样本缓冲区。要获取此接口，请在任何输入或输出缓冲区上查询 IMFMediaBuffer 接口。当该接口可用时不使用该接口可能会导致额外的缓冲区副本。为了正确使用此接口，当 IMF2DBuffer 可用时，转换不应使用 IMFMediaBuffer 接口锁定缓冲区。

有关处理视频数据的更多信息，请参阅未压缩视频缓冲区。

Flushing an MFT 刷新 MFT

刷新 MFT 会导致 MFT 丢弃其所有输入数据。这可能会导致输出流中断。当客户端不关心丢失数据时，客户端通常会在寻找输入流中的新点或切换到新输入流之前刷新 MFT。

要刷新 MFT，请使用 MFT_MESSAGE_COMMAND_FLUSH 消息调用 IMFTransform::ProcessMessage。

Draining an MFT 排空 MFT

耗尽 MFT 会导致 MFT 从已发送的任何输入数据中产生尽可能多的输出。如果 MFT 无法从可用输入生成完整的输出样本，它将丢弃输入数据。客户端通常会在到达源流末尾时或在源流中的格式更改之前立即耗尽 MFT。要排空 MFT，请执行以下操作：

1. 使用 MFT_MESSAGE_COMMAND_DRAIN 消息调用 ProcessMessage。该消息通知 MFT 它应该从已发送的输入数据中传递尽可能多的输出数据。
2. 调用ProcessOutput获取输出数据，直到该方法返回 MF_E_TRANSFORM_NEED_MORE_INPUT。

当 MFT 被耗尽时，它将不再接受任何输入。

Sample Attributes 采样属性

输入样本可能具有必须复制到相应输出样本的属性。

- 如果 MFT 针对 MFPKEY_EXATTRIBUTE_SUPPORTED 属性返回 VARIANT_TRUE，则 MFT 必须复制属性。
- 如果 MFPKEY_EXATTRIBUTE_SUPPORTED 属性为 VARIANT_FALSE 或未设置，则客户端必须复制属性。

对于具有 1 个输入和 1 个输出的 MFT，您可以使用以下一般规则：

- 如果每个输入样本恰好生成一个输出样本，则可以让客户端复制属性。保留 MFPKEY_EXATTRIBUTE_SUPPORTED 属性未设置。
- 如果输入样本和输出样本之间不存在一一对应的关系，则MFT必须确定输出样本的正确属性。将 MFPKEY_EXATTRIBUTE_SUPPORTED 属性设置为 VARIANT_TRUE。

Discontinuities不连续性

不连续性是音频或视频流中的中断。不连续性可能是由网络连接上的数据包丢失、文件数据损坏、从一个源流切换到另一个源流或多种其他原因引起的。通过在不连续性之后的第一个样本上设置 MFSampleExtension_Discontinuity 属性来发出不连续性信号。不可能在样本中间发出不连续的信号。因此，任何不连续的数据都应该在单独的样本中发送。

某些转换，尤其是处理未压缩数据的转换（例如音频和视频效果），在处理输入数据时应忽略不连续性。这些 MFT 通常设计用于处理连续数据，并且应将它们接收到的任何数据视为连续数据，即使在不连续之后也是如此。

如果 MFT 忽略输入数据上的不连续性，并且输出样本与输入样本具有相同的时间戳，则它仍应在输出样本上设置不连续性标志。然而，如果输出样本具有不同的时间戳，则 MFT 不应传播不连续性。（例如，在某些音频重采样器中就是这种情况。）流中错误位置的不连续性比没有不连续性更糟糕。

大多数解码器不能忽略不连续性，因为不连续性会影响下一个样本的解释。任何使用帧间压缩的编码技术（例如 MPEG-2）都属于这一类。某些编码方案仅使用帧内压缩，例如 DV 和 MJPEG。这些解码器可以安全地忽略不连续性。

响应不连续性的转换通常应在不连续性之前输出尽可能多的数据，并丢弃其余数据。具有不连续性标志的输入样本应像流中的第一个样本一样进行处理。（此行为与为 MFT_MESSAGE_COMMAND_DRAIN 消息指定的行为匹配。）但是，确切的详细信息将取决于媒体格式。

如果解码器不采取任何措施来减轻不连续性，则它应该将不连续性标志复制到输出数据。解复用器和其他完全处理压缩数据的 MFT 必须将任何不连续性复制到其输出流。否则，下游组件可能无法正确解码压缩数据。一般来说，将不连续性传递给下游几乎总是正确的，除非 MFT 包含显式代码来平滑不连续性。

Dynamic Format Changes动态格式变化

格式在流式传输期间可能会发生变化。例如，宽高比可以在视频流的中间发生变化。

有关MFT如何处理流更改的详细信息，请参阅处理流更改。

Stream Events直播事件

要将事件发送到 MFT，请调用 IMFTransform::ProcessEvent。如果该方法返回 MF_S_TRANSFORM_DO_NOT_PROPAGATE_EVENT，则 MFT 将在后续调用 ProcessOutput 时将该事件返回给调用者。如果该方法返回任何其他 HRESULT 值，则 MFT 将不会在 ProcessOutput 中将事件返回给客户端。在这种情况下，客户端负责将事件向下游传播到管道中的下一个组件（如果适用）。有关详细信息，请参阅 IMFTransform::ProcessOutput。

Asynchronous MFTs 异步MFT

当 MFT 在 Windows Vista 中引入时，API 是为同步数据处理而设计的。在该模型中，MFT 始终要么等待获取输入，要么等待产生输出。

考虑一个典型的视频解码器。为了获取解码的帧，客户端调用 IMFTransform::ProcessOutput。如果解码器有足够的数据来解码帧，则在 MFT 解码帧时 ProcessOutput 会阻塞。否则，ProcessOutput 返回 MF_E_TRANSFORM_NEED_MORE_INPUT，指示客户端应调用 IMFTransform::ProcessInput。

如果解码器在一个线程上执行所有解码操作，则该模型可以很好地工作。但假设解码器使用多个线程并行解码帧。为了获得最佳性能，只要解码线程空闲，解码器就应该接收新的输入。但是线程完成解码操作的速率不会与客户端对 ProcessInput 和 ProcessOutput 的调用完全一致，从而导致线程等待工作。

Windows 7 引入了事件驱动的 MFT 异步处理。在此模型中，每当 MFT 需要输入或有输出时，它都会向客户端发送一个事件。

异步MFT必须实现以下接口：

- IMFTransform
- IMFMediaEventGenerator
- IMFShutdown

Event 事件

异步 MFT 使用以下事件来指示其数据处理状态：

- METransformNeedInput 当 MFT 可以接受更多输入时发送。
- METransformHaveOutput 当MFT 有输出时发送。
- METransformDrainComplete 当排出操作完成时发送。参考Draining
- METransformMarker 当标记正在处理时发送。参考Marker

这些事件是带外发送的。了解 MFT 背景下带内事件和带外事件之间的区别非常重要。

最初的 MFT 设计支持带内事件。带内事件包含有关数据流的信息，例如有关格式更改的信息。客户端通过调用 IMFTransform::ProcessEvent 将带内事件发送到 MFT。MFT 可以在 ProcessOutput 方法中将带内事件发送回客户端。（具体来说，事件在 MFT_OUTPUT_DATA_BUFFER 结构的 pEvents 成员中传送。）

MFT 通过 IMFMediaEventGenerator 接口发送带外事件，如下所示：

1. MFT 实现 IMFMediaEventGenerator 接口，如媒体事件生成器中所述。

2. 客户端在 MFT 上调用 IUnknown::QueryInterface 来获取 IMFMediaEventGenerator 接口。异步 MFT 必须公开此接口。同步 MFT 不应公开此接口。
3. 客户端调用 IMFMediaEventGenerator::BeginGetEvent 和 IMFMediaEventGenerator::EndGetEvent 从 MFT 接收带外事件。

细节参考本节

Registering and Enumerating MFTs注册和枚举 MFT

要寻找在系统上注册的 MFT，应用程序可以调用 MFTEnumEx 函数。

当您注册媒体基础转换 (MFT) 时，会将两种类型的信息写入注册表：

- MFT的CLSID，以便客户端可以调用CoCreateInstance或CoGetClassObject来创建MFT的实例。此注册表项遵循 COM 类工厂的标准格式。有关详细信息，请参阅组件对象模型 (COM) 的 Windows SDK 文档。
- 使应用程序能够按功能类别枚举 MFT 的信息。

要在注册表中创建 MFT 枚举条目，请调用 MFTRegister 函数。您可以包含有关 MFT 的以下信息：

- MFT的类别，例如视频解码器或视频编码器。有关类别列表，请参阅 MFT_CATEGORY。
- MFT 支持的输入和输出格式的列表。每种格式都由主要类型和子类型定义。（要获取更详细的格式信息，客户端必须创建 MFT 并调用 IMFTransform 方法。）拓扑加载器在解析部分拓扑时使用此信息。

要从注册表中删除条目，请调用 MFTUnregister。

示例代码参考这里。

Field of Use Restrictions使用领域限制

Media Foundation 提供了一种机制，用于对 Media Foundation 转换 (MFT)（特别是编解码器）进行使用领域限制。此机制要求 MFT 阻止应用程序自己使用它，直到应用程序与 MFT 执行握手为止。媒体基金会没有定义握手——通常，它会涉及某种加密交换。

如果 MFT 有使用字段限制，请在注册 MFT 时设置 MFT_ENUM_FLAG_FIELDOFUSE 标志。该标志适用于以下 MFT 注册 API：

- MFTRegister
- MFTRegisterLocal
- MFTRegisterLocalByCLSID
- IMFLocalMFTRegistration::RegisterMFTs

默认情况下，使用此标志注册的 MFT 将从枚举结果中排除。要枚举具有使用字段限制的 MFT，请调用 MFTEnumEx 并在 Flags 参数中指定 MFT_ENUM_FLAG_FIELDOFUSE 标志。下图说明了此过程。

MFTEnum 函数始终不包含任何具有使用领域限制的 MFT。
要使用具有使用领域限制的 MFT，请执行以下步骤：

1. 该应用程序实现 IMFFieldOfUseMFTUnlock 接口。
2. IMFFieldOfUseMFTUnlock::Unlock 方法有一个指向 MFT 的 IUnknown 接口的指针。
3. 在 Unlock 方法中，应用程序使用 MFT 定义的任何机制执行所需的握手。Media Foundation API 未定义此步骤。
4. 如果 Unlock 方法成功，MFT 将自行解锁。

应用程序通过设置 MFT_FIELDOFUSE_UNLOCK_Attribute 属性来指定 IMFFieldOfUseMFTUnlock 指针。有多种不同的方法来设置此属性，具体取决于应用程序创建解码器或编码管道的方式：
详细参考这里。
下图显示了 MFT 激活对象和 IMFFieldOfUseMFTUnlock 接口之间的关系。

Comparison of MFTs and DMOsMFT 和 DMO 的比较

媒体基础转换 (MFT) 是首次随 DirectX 媒体对象 (DMO) 引入的转换模型的演变。
详细参考这里。

Writing a Custom MFT编写自定义 MFT

参考这里

Media Sinks 媒体接收器

媒体接收器是接收媒体数据的管道对象。媒体接收器是一个或多个媒体流的目的地。媒体接收器分为两大类：

- 渲染器是呈现数据以供播放的媒体接收器。增强视频渲染器（EVR）显示视频帧，音频渲染器通过声卡或其他音频设备播放音频流。
- 归档接收器是将数据写入文件或其他存储的媒体接收器。

它们之间的主要区别在于存档接收器不以固定的播放速率消耗数据。相反，它会尽快写入收到的数据。
媒体接收器公开 IMFMediaSink 接口。每个媒体接收器包含一个或多个流接收器。每个流接收器接收来自一个流的数据。流接收器暴漏一个 IMFStreamSink 接口。通常，应用程序不会直接创建媒体接收器。相反，应用程序创建一个或多个激活对象，媒体会话使用这些对象来创建接收器。接收器上的所

有其他操作均由媒体会话处理，并且应用程序不会调用媒体接收器或任何流接收器上的任何方法。有关激活对象的更多信息，请参阅激活对象。
如果您正在编写自定义媒体接收器，或者想要在不使用媒体会话的情况下直接使用媒体接收器的话，往下看：

Stream Sinks流接收器

媒体接收器可以具有固定数量的流接收器，也可以支持添加和删除流接收器。如果它具有固定数量的流接收器，则 IMFMediaSink::GetCharacteristics 方法将返回 MEDIASINK_FIXED_STREAMS 标志。否则，您可以添加和删除流接收器。要添加新的流接收器，请调用 IMFMediaSink::AddStreamSink。要删除流接收器，请调用 IMFMediaSink::RemoveStreamSink。MEDIASINK_FIXED_STREAMS 标志指示媒体接收器不支持这两种方法。（它可能支持其他一些方式来配置流的数量，例如通过在创建接收器时设置初始化参数。）流接收器的列表是有序的。要按索引值枚举它们，请调用 IMFMediaSink::GetStreamSinkByIndex 方法。
流接收器也有标识符。流标识符在媒体接收器内是唯一的，但不必是连续的。根据媒体接收器，流标识符可能具有与内容相关的某些含义。例如，存档接收器可能会将流标识符写入文件头中。否则，他们就是任意的。要通过标识符获取流接收器，请调用 IMFMediaSink::GetStreamSinkById。

Presentation Clock 渲染时钟

媒体接收器消耗样本的速率由演示时钟控制。媒体会话选择演示时钟并通过调用媒体接收器的 IMFMediaSink::SetPresentationClock 方法在媒体接收器上设置它。在开始流式传输之前，必须在媒体接收器上设置演示时钟。每个媒体接收器都需要一个演示时钟才能运行。媒体接收器使用演示时钟有两个目的：

- 当流媒体开始或停止时接收通知。媒体接收器通过 IMFClockStateSink 接口接收这些通知，所有媒体接收器都必须实现该接口。
- 确定何时应该渲染样本。当媒体接收器收到新样本时，它会从样本中获取时间戳并尝试在该呈现时间呈现样本。

演示时钟从另一个称为演示时间源的对象导出其时钟时间。演示时间源公开 IMFPresentationTimeSource 接口。一些媒体接收器想要访问准确的时钟，它们就要公开此接口。这意味着媒体接收器可以根据其自己的时钟提供的时间来安排样本。然而，媒体接收器不能假设情况确实如此。它必须始终使用来自演示时钟的时间，无论演示时钟是由媒体接收器本身还是由某个其他时钟驱动。
如果媒体接收器无法将速率与其自身以外的时钟相匹配，则 GetCharacteristics 方法将返回 MEDIASINK_CANNOT_MATCH_CLOCK 标志。如果存在此标志并且呈现时钟使用不同的呈现时间源，则媒体接收器的性能可能很差。例如，在播放过程中可能会出现故障。
无速率接收器（*rateless sink*）是一种媒体接收器，它忽略样本上的时间戳并在每个样本到达后立即使用数据。无速率媒体接收器从 GetCharacteristics 方法返回 MEDIASINK_RATELESS 标志。通常，此标志适用于存档接收器。如果管道中的每个媒体接收器都是无速率的，则媒体会话将使用特殊的无速率呈现时钟。该时钟的运行速度与接收器消耗样本的速度一样快。

Stream Formats流格式

在媒体接收器可以接收样本之前，客户端必须在流接收器上设置媒体类型。要设置媒体类型，请调用流接收器的 IMFStreamSink::GetMediaTypeHandler 方法。此方法返回指向 IMFMediaTypeHandler 接口的指针。使用此接口获取首选媒体类型列表、获取当前媒体类型以及设置媒体类型。

要获取首选媒体类型的列表，请调用 IMFMediaTypeHandler::GetMediaTypeByIndex。首选类型应作为对客户的提示。该列表可能不完整或包含部分媒体类型。部分媒体类型（*partial media type*）是一种不具有描述有效格式所需的所有属性的媒体类型。例如，部分视频类型可能指定颜色空间和位深度，但不指定图像宽度或高度。部分音频类型可能指定压缩格式和采样率，但不指定音频通道的数量。

要获取流接收器的当前媒体类型，请调用 IMFMediaTypeHandler::GetCurrentMediaType。当第一次创建流接收器时，它可能已经设置了默认媒体类型，或者在客户端设置媒体类型之前它可能没有媒体类型。

要设置媒体类型，请调用 IMFMediaTypeHandler::SetCurrentMediaType。某些流接收器可能不支持在设置后更改类型。因此，在设置媒体类型之前对其进行测试很有用。要测试媒体接收器是否接受媒体类型（无需设置类型），请调用 IMFMediaTypeHandler::IsMediaTypeSupported。

Data Flow 数据流

媒体接收器使用拉模型，这意味着流接收器根据需要请求数据。客户应及时回复，以免出现问题。

一些媒体接收器支持预加载prerolling。就是在演示时钟开始之前向媒体接收器提供数据的过程。如果媒体接收器支持预加载，则媒体接收器公开 IMFMediaSinkPreroll 接口，并且 GetCharacteristics 方法返回 MEDIASINK_CAN_PREROLL 标志。预卷可确保媒体接收器准备好在呈现时钟启动时呈现第一个样本。如果媒体接收器支持，建议客户端始终预加载，因为它可以防止播放期间出现故障或间隙。

流向媒体接收器的数据流的工作原理如下：

1. 客户端设置媒体类型和演示时钟。媒体接收器将自身注册到呈现时钟以接收有关时钟状态变化的通知。
2. （可选）客户端查询 IMFMediaSinkPreroll。如果媒体接收器公开此接口，则客户端调用 IMFMediaSinkPreroll::NotifyPreroll。否则，客户端跳至步骤 5。
3. 每个流接收器都会发送一个或多个 MEStreamSinkRequestSample 事件。为了响应每个事件，客户端获取该流的下一个数据样本并调用 IMFStreamSink::ProcessSample。
4. 当每个流接收器接收到足够的预加载数据时，它会发送 MEStreamSinkPrerolled 事件。
5. 客户端调用 IMFPresentationClock::Start 来启动演示时钟。
6. 演示时钟通过调用 IMFClockStateSink::OnClockStart 通知媒体接收器时钟正在启动。
7. 为了获取更多数据，每个流接收器都会发送 MEStreamSinkRequestSample 事件。为了响应每个事件，客户端获取下一个样本并调用 ProcessSample。重复此步骤直到演示结束。

大多数媒体接收器异步处理样本，因此流接收器一次可以发出多个样本请求。

在流式传输过程中，客户端可以随时调用 IMFStreamSink::PlaceMarker 和 IMFStreamSink::Flush。标记将在下一节中描述。刷新会导致流接收器删除已排队但尚未渲染的所有样本。

Markers标记

标记为客户端提供了一种指示流中特定点的方法。标记由以下信息组成：

- 标记类型，定义为 MFSTREAMSINK_MARKER_TYPE 枚举的成员。
- 与标记关联的数据。数据的含义取决于标记类型。某些标记类型没有数据。
- 供客户自己使用的可选数据。

要放置标记，客户端调用 IMFStreamSink::PlaceMarker。流接收器完成对 PlaceMarker 调用之前接收到的所有样本的处理，然后发送 MEStreamSinkMarker 事件。

大多数媒体接收器都会保留一个待处理样本队列，它们会异步处理这些样本。标记事件必须通过样本处理进行序列化，因此媒体接收器应将标记放置在同一队列中。例如，假设客户端进行以下方法调用：

1. **ProcessSample** (Sample #1)
2. **ProcessSample** (Sample #2)
3. **PlaceMarker** (Marker #1)
4. **ProcessSample** (Sample #3)
5. **PlaceMarker** (Marker #2)

在此示例中，流接收器必须在处理样本 #2 后发送标记 #1 的 MEStreamSinkMarker 事件，并在处理样本 #3 后发送标记 #2 的事件。

如果客户端刷新流接收器，则流接收器立即处理队列中的任何标记。它将这些事件的状态代码设置为 E_ABORT。

一些标记包含与媒体接收器相关的信息：

- MFSTREAMSINK_MARKER_TICK：表示流中存在间隙。下一个样本将是不连续的。
- MFSTREAMSINK_MARKER_ENDOFSEGMENT：表示段的结束或流的结束。下一个样本（如果有）可能是不连续的。
- MFSTREAMSINK_MARKER_EVENT：包含一个事件。根据事件类型和媒体接收器的实现，媒体接收器可能会处理该事件或忽略该事件。

State Changes状态变化

通过媒体接收器的IMFClockStateSink 接口向媒体接收器通知呈现时钟的状态变化。当客户端设置演示时钟时，媒体接收器调用 IMFPresentationClock::AddClockStateSink 来注册自身以接收来自时钟的通知。下表总结了媒体接收器响应时钟状态变化的行为方式。

Clock state change 时钟状态改变	Sample processing 来样加工	Marker processing 标记处理
OnClockStart时钟启动时	Process samples whose time stamp is equal to or later than the clock start time. 处理时间戳等于或晚于时钟开始时间的样本。	Send the MEStreamSinkMarker event when all of the samples received before the marker have been processed.

		当标记之前接收到的所有样本都已处理完毕时，发送 MEStreamSinkMarker 事件。
OnClockPause时钟暂停	The media sink can fail **ProcessSample** while paused. If the media sink accepts samples while paused, it must queue them until the clock restarts. Do not process any queued samples while paused. 媒体接收器在暂停时可能会导致 ProcessSample 失败。如果媒体接收器在暂停时接受样本，则必须将它们排队，直到时钟重新启动。暂停时不要处理任何排队的样本。	If there are queued samples, put markers onto the same queue. Send the marker event when the clock restarts. Otherwise, send the marker event immediately. 如果有排队样本，请将标记放入同一队列中。当时钟重新启动时发送标记事件。否则，立即发送标记事件。
OnClockRestart时钟重启	Process any samples that were queued while paused, and then treat the same as **OnClockStart** . 处理暂停时排队的任何样本，然后以与 OnClockStart 相同的方式处理。	Send MEStreamSinkMarker events for queued markers (serialized with sample processing), and then treat the same as **OnClockStart** . 发送排队标记的 MEStreamSinkMarker 事件（通过示例处理进行序列化），然后与 OnClockStart 进行相同的处理。
OnClockStop时钟停止	Drop all queued samples. Further calls to **ProcessSample** can fail. 删除所有排队的样本。对 ProcessSample 的进一步调用可能会失败。	Send queued marker events. On subsequent calls to **PlaceMarker** , send the marker event immediately. 发送排队的标记事件。在随后调用 PlaceMarker 时，立即发送标记事件。

更多信息参考这里。

Media Session媒体会话

媒体会话是管理管道中数据流的对象。它可用于播放和编码文件。

媒体会话公开 IMFMediaSession 接口。有两种创建媒体会话的方法，具体取决于您的应用程序是否支持受保护的内容：

- 如果您的应用程序不支持受保护的内容，您可以通过调用 MFCreateMediaSession 创建媒体会话。该函数在应用程序进程内创建媒体会话。
- 要支持受保护的内容，请通过调用 MFCreatePMPMediaSession 创建媒体会话。此函数在受保护的媒体路径 (PMP) 进程内创建媒体会话。应用程序接收一个指向代理对象的指针，该代理对象跨进程边界封送方法调用。请注意，PMP 媒体会话可用于播放清晰内容以及受保护的内容。

任何使用媒体会话的应用程序都将遵循以下一般步骤：

1. 创建拓扑
2. 通过调用 IMFMediaSession::SetTopology 对媒体会话上的拓扑进行排队。
3. 通过调用 IMFMediaSession::Start、IMFMediaSession::Pause 或 IMFMediaSession::Stop 控制数据流。
4. 在应用程序退出之前，调用 IMFMediaSession::Close 关闭媒体会话。
5. 通过调用 IMFMediaSource::Shutdown 关闭应用程序创建的所有媒体源。
6. 通过调用 IMFMediaSession::Shutdown 关闭媒体会话。

使用媒体会话时，应用程序不应直接启动、暂停或停止媒体源。所有状态更改都必须通过调用 IMFMediaSession 方法来启动。由媒体会话处理媒体源中的状态更改。

要播放受保护的内容，您必须通过调用 MFCreatePMPMediaSession 在受保护的媒体路径 (PMP) 内创建媒体会话。此函数在 PMP 内创建媒体会话的实例，并返回指向跨进程边界编组接口的代理对象的指针。

在大多数方面，使用 PMP 内的媒体会话对于应用程序来说是透明的。但是，应用程序可能需要调用某些操作来使用户能够播放内容。例如，用户可能需要获取 DRM 许可证。Media Foundation 使用 IMFContentEnabler 接口为这些操作定义通用机制。

媒体会话管理演示时钟的各个方面：

- 创建演示时钟。
- 选择时间源。
- 通知媒体接收器有关时钟的信息
- 根据需要启动、停止和暂停时钟。
- 关闭时钟。

要获取指向演示时钟的指针，请在媒体会话上调用 IMFMediaSession::GetClock。在媒体会话发送带有 MF_TOPOSTATUS_READY 标志的 MESessionTopologyStatus 事件之前，演示时钟不会返回有效时间。在此之前，GetClock 返回 MF_E_CLOCK_NO_TIME_SOURCE。

使用媒体会话的应用程序永远不应启动、停止或暂停演示时钟；改变时钟频率；或关闭时钟。

当应用程序调用 IMFMediaSession::Start 时，媒体会话将启动演示时钟，其起始时间等于 Start 方法中指定的起始位置。

How to Control Presentation States如何控制演示状态

媒体会话提供传输控制，例如更改呈现状态（播放列表样式播放场景中的播放、暂停和停止）。本主题描述应用程序应调用以更改播放状态的媒体会话方法。

State transition 状态转换	Description 描述
Play -> Pause	The presentation clock freezes. 演示时钟冻结。
Play -> Stop	The presentation clock is reset. 演示时钟已重置。
Pause -> Play	The presentation clock resumes from the time it froze during the Play to Pause transition. 演示时钟从播放到暂停转换期间冻结的时间恢复。
Pause -> Stop	The presentation clock is reset. 演示时钟已重置。
Stop -> Play	The presentation clock starts from the beginning of the presentation. 演示时钟从演示开始时开始计时。
Stop -> Pause	Not allowed. 不允许。

To change presentation states更改演示状态

- 调用 IMFMediaSession::Pause 方法暂停播放。在调用此方法之前，应用程序必须调用 IMFMediaSession::GetSessionCapability 方法来发现媒体源是否支持 Pause 状态。如果是，此方法在 pdwCaps 参数中返回 MFSESSIONCAP_PAUSE。暂停会暂时停止媒体会话、演示时钟和当前演示的流接收器。调用成功完成后，应用程序会收到 MESessionPaused 事件。
- 调用 IMFMediaSession::Stop 方法停止播放。此方法通过停止媒体源、相应的时钟和流接收器来停止媒体会话。如果媒体会话正在控制序列源，则底层本机源将被序列源停止。媒体会话停止后，应用程序会收到 MESessionStopped 事件。
- 调用 IMFMediaSession::Start 方法开始播放或寻找新位置。此方法从暂停和停止状态启动媒体会话。媒体会话负责设置管道中的数据流。此方法指示媒体会话启动演示时钟。在此调用之后，媒体会话将 MESessionStarted 事件发送到应用程序。

Media Session Events媒体会话事件

大多数媒体会话的操作都是异步执行的，因此应用程序必须使用媒体会话的 IMFMediaEventGenerator 接口来侦听事件。（IMFMediaSession 接口继承 IMFMediaEventGenerator）事件的确切顺序将取决于您的应用程序，但媒体会话几乎在任何情况下都会引发以下事件。

Event 事件	Description 描述
MEEndOfPresentation	Raised when the media source has completed the presentation. Data might still be moving through the pipeline at this time. 当媒体源完成演示时引发。此时数据可能仍在通过管道传输。
MEError	Raised if an error occurs during streaming. 如果流式传输期间发生错误，则会引发该错误。
MESessionClosed	Raised when the **Close** method completes. This event is the last event that the Media Session queues. After you receive this event, it is safe to shut down any media sources that you created. Close 方法完成时引发。该事件是媒体会话排队的最后一个事件。收到此事件后，可以安全地关闭您创建的任何媒体源。
MESessionEnded	Raised when the Media Session is done with the last presentation. 当媒体会话完成最后一次演示时引发。
MESessionNotifyPresentationTime	Notifies the application of the presentation time when the new presentation will start. 通知应用程序新演示文稿开始的演示时间。
MESessionStarted	Raised when the **Start** method completes. Unless an error occurred, data is moving through the pipeline at this point. 当 Start 方法完成时引发。除非发生错误，否则此时数据正在通过管道移动。
MESessionTopologySet	Raised when the **SetTopology** method completes. Unless an error occurs, the application does not need to take any action. 当 SetTopology 方法完成时引发。除非发生错误，否则应用程序不需要采取任何操作。
MESessionTopologyStatus	Raised at various times when the status of a topology changes. 当拓扑状态发生变化时，会在不同时间引发。

IMFMediaSession::Shutdown 方法不会引发事件。Shutdown 方法是同步的。此方法返回后，可以安全地释放事件回调指针。

除了来自媒体会话的事件之外，应用程序还可能从拓扑中的媒体接收器接收事件。这些可以是媒体接收器定义的自定义事件，可能包含任意数据。例如，接收器可能从源数据导出事件数据，源数据可能来自不受信任的外部源。应用程序应忽略任何它无法识别的事件，并在解析事件数据时小心谨慎。

PMP Media SessionPMP 媒体会话

应用程序可以在称为受保护媒体路径 (PMP) 进程的单独进程中创建媒体会话。PMP 过程的主要目的是使用数字版权管理 (DRM) 来播放受保护的内容。默认情况下，PMP 进程是在受保护环境 (PE) 内创建

的。只有可信的、签名的组件才能加载到 PE 中。PMP 流程的第二个好处是它将应用程序流程与媒体管道隔离。有关 PMP 进程的详细信息，请参阅受保护的媒体路径。关于PMP媒体会话参考这里。

Sequencer Source 序列源

序列源使应用程序能够按顺序播放媒体源集合，并在源之间实现无缝转换。您可以使用它来创建播放列表，或同时播放来自多个源的流。可用于以下场景：

- 创建一个可以从一个媒体源无缝切换到下一个媒体源的播放列表。
- 同时播放多个来源的流；例如，将一个文件中的音频与另一个文件中的视频一起播放。
- 在连续的播放列表条目中不同媒体源的流之间切换；例如，播放列表可以具有共享相同视频源的条目，而每个条目包含不同的音频源。

对于播放列表的每个元素，应用程序都会创建一个单独的拓扑。这些拓扑中的媒体源称为本机源（native soruces），以区别于序列源。在播放期间，整个拓扑序列称为演示（presentation），序列中的每个拓扑称为片段（segment）。

播放由媒体会话控制，它提供状态转换控制，例如播放、暂停和停止。媒体会话还管理演示时间并将事件发送到应用程序。（来自序列源的事件通过媒体会话转发到应用程序）

为了创建播放列表，应用程序创建一个或多个播放拓扑，并按照所需的播放顺序将它们在音序器源上排队。在内部，定序器源修改拓扑，以便源节点指向定序器源而不是本机源。应用程序将这些修改后的拓扑而不是原始拓扑发送到媒体会话。这使得定序器源能够聚合本机源并与媒体会话进行通信。

为了实现片段之间的无缝过渡，定序器源会预加载每个片段。当播放一个片段时，在播放下一个片段之前，序列源会触发包含演示描述符的 MENewPresentation 事件。应用程序使用此演示描述符来获取演示中下一个片段的拓扑，并将该拓扑在媒体会话上排队。

下图显示了播放列表条目通过序列源的数据流。应用程序使用源解析器创建本机源，为每个段构建拓扑，并将拓扑在定序器源上排队。

Using the Sequencer Source 使用序列源

要播放序列媒体源，请执行以下步骤：

1. 要创建定序器源，请调用 MFCreateSequencerSource 函数。
2. 对于每个片段，创建一个播放拓扑（如创建播放拓扑中所述）。要将拓扑添加到演示文稿中，请调用 IMFSequencerSource::AppendTopology。
3. 在开始播放之前，请在音序器源上调用 IMFMediaSource::CreatePresentationDescriptor。此方法返回指向第一个段的表示描述符的指针。要获取与此段关联的拓扑，请在 IMFMediaSourceTopologyProvider 接口的定序器源上调用 QueryInterface。将表示描述符传递给 IMFMediaSourceTopologyProvider::GetMediaSourceTopology 方法。该方法返回一个指向拓扑的指针。
4. 通过调用媒体会话的 IMFMediaSession::SetTopology 方法，将第一个分段的拓扑传递到媒体会话。

5. 通过调用 IMFMediaSession::Start 开始播放。
6. 当定序器源准备好预卷下一个片段时，它会发送一个 MENewPresentation 事件，其事件数据是 IMFPresentationDescriptor 接口指针。再次，通过在定序器源上调用 GetMediaSourceTopology 来获取片段的拓扑，并通过调用 SetTopology 在媒体会话上设置拓扑。无需重新启动媒体源；它会自动播放到下一个片段。
7. 在应用程序退出之前，通过调用 IMFMediaSource::Shutdown 关闭定序器源。

一个示例代码，参考 Sequencer Source Example Code

Sequencer Source Events序列源事件

当序列源播放文件序列时，媒体会话通常会发送在正常播放期间发送的所有相同事件，这些事件在媒体会话事件中列出。应用程序使用媒体会话的 IMFMediaEventGenerator 接口获取这些事件。
细节参考这里这里

How to Create a Playlist如何创建播放列表

参考这里

Sequencer Source Example Code定序器源示例代码

参考这里

Sequence Presentation Times 序列呈现时间

序列源支持两种不同的模式：播放列表序列和编辑序列。
在编辑序列中，应用程序在开始播放之前预先指定每个片段的持续时间。在播放列表序列中，应用程序不会提前指定持续时间。（事实上，持续时间可能不知道。）
在这两种情况下，您都可以指定片段的媒体开始和媒体停止时间。这些时间指定源文件中段开始和结束的位置。例如，假设源文件长 90 秒。如果您想修剪前 10 秒和最后 10 秒，您可以指定以下值：

- 媒体开始：10秒
- 媒体停止：80秒

要指定媒体开始时间，请在源节点上设置 MF_TOPONODE_MEDIASTART 属性。要指定媒体停止时间，请在源节点上设置 MF_TOPONODE_MEDIASTOP 属性。
要创建编辑序列，请在创建媒体会话时设置 MF_SESSION_GLOBAL_TIME 属性。否则，媒体会话需要播放列表序列。在编辑序列中，每个段拓扑都必须具有 MF_TOPOLOGY_PROJECTSTART 属性和 MF_TOPOLOGY_PROJECTSTOP 属性。

Playlist Sequences播放列表序列

在播放列表序列中，呈现时钟从零开始并继续跨越段边界。本机源提供的样本的时间戳等于媒体时间。管道将时间戳转换为正确的呈现时间，如下所示：

- New time stamp = media time + offset − media start。offset 的值是前一段结束的呈现时间。对于

第一段，偏移量为零。以下是如何计算这些时间戳转换的两个示例：

- 示例 1：假设第一个片段 (S1) 的长度为 10 秒，第二个片段 (S2) 的媒体开始时间为零。本机源使用媒体时间作为其时间戳，因此 S2 中的第一个样本的时间戳为零。偏移量为10秒（S1的持续时间），因此调整后的时间戳为：0 + 10 − 0 = 10秒。
- 示例 2：假设片段 S1 长 10 秒，S2 的媒体开始时间为 5 秒。S2 中的第一个样本的时间戳为 5 秒（媒体时间）。偏移量为10秒，因此调整后的时间戳为：5 + 10 − 5 = 10秒。

源节点下游的所有管道组件都会接收带有调整后的时间戳的样本。拓扑中的源节点可以具有不同的媒体开始时间，因此需要针对拓扑的每个分支单独计算调整。

当演示切换到下一个片段时，演示时钟不会停止或重置，并且演示时间单调增加。在新片段开始之前，媒体会话会向应用程序发送 MESessionNotifyPresentationTime 事件。该事件指定该段相对于呈现时钟的开始时间以及偏移值。当新段开始时，管道会使用值 VT_EMPTY 在定序器源上调用 Start。定序器源发送没有开始时间的 MESourceStarted 事件。

为了进行查找，应用程序指定段标识符加上段内的时间偏移量。寻道后，呈现时钟从段偏移处开始。以下是该过程如何工作的示例：

- 示例 3：应用程序寻求对 S3 进行分段，分段偏移量为 10 秒。演示时钟从 10 秒（段偏移）开始。该偏移量不包括段 S1 和 S2 的持续时间。定序器源发送 MESourceStarted 事件，其开始时间等于段偏移量（10 秒）。

定位后，如果继续播放到下一个片段，则过渡的工作方式与前面的示例类似，只是偏移量不包括跳过的片段。

以下是影响样本时间戳方式的一些进一步详细信息：

- 解码器可能需要超出媒体停止时间的数据。管道从源中提取解码器所需的尽可能多的数据，然后修剪解码器的输出样本。
- 转换可能会缓冲数据。例如，音频效果可能需要执行此操作。当段结束时，变换的最后一个样本的时间戳早于段的末尾，因为变换保留了一些数据。当下一段开始时，第一个样本的时间戳略早于该段的开始。时间戳之间没有间隙，因此到达媒体接收器的数据是连续的。当最后一段结束时，管道会耗尽转换，因此不会丢失数据。
- 源可能需要比媒体开始时间稍早开始，以拾取上一个关键帧。因此，调整后，第一个样本的呈现时间可能为负。

Editing Sequences编辑序列

在编辑序列中，应用程序通过设置 MF_TOPOLOGY_PROJECTSTART 和 MF_TOPOLOGY_PROJECTSTOP 属性提前指定段边界。管道计算时间戳调整的方式与播放列表序列几乎相同：

- 对于偏移量，它使用 MF_TOPOLOGY_PROJECTSTART 的值，而不是使用观察到的段末尾。
- 对于查找，偏移量使用等于段的 MF_TOPOLOGY_PROJECTSTART 值加上段偏移量的值。

因此，编辑序列中的演示时间始终相对于演示的开始，即使应用程序寻找另一个片段也是如此。

Rate Control速率控制

媒体会话支持更改播放速率，应用程序可以使用它来实现快进和快退等播放功能。本节介绍应用程序如何在使用媒体会话时更改播放速率。

在Media Foundation中，播放速率表示为当前播放速率与正常播放速率的比率。例如，速率 2.0 是正常速度的两倍，0.5 是正常速度的一半。负值表示反向播放。-2.0 的播放速率以正常速度的两倍向后播放流。速率为零会导致渲染一帧；此后，演示时钟不再前进。为了以零速率获得另一帧，应用程序必须寻找新的位置。应用程序使用以下接口来控制播放速率。

- IMFRateSupport。用于找出可能的最快和最慢播放速率。
- IMFRateControl。用于更改播放速率。

要获取这两个接口，请在媒体会话上调用 IMFGetService::GetService。服务标识符是 MF_RATE_CONTROL_SERVICE。

通过使用码率控制服务，应用程序可以实现快进和快退播放。

Thinning 抽帧

抽帧是减少流中样本数量以降低总体比特率的任何过程。对于视频，抽帧通常是通过丢弃增量帧并仅传送关键帧来完成的。通常，管道可以使用抽帧播放支持更快的播放速率，因为数据速率较低，因为增量帧未解码。

抽帧不会改变样本上的时间戳或持续时间。例如，如果视频流的标称速率为每秒 25 帧，则即使媒体源丢弃所有增量帧，每帧的持续时间仍标记为 40 毫秒。这意味着一帧的结束和下一帧的开始之间会有一个时间间隙。

Scrubbing 定位

定位是通过与滚动条、时间轴或其他时间的可视化表示进行交互，即刻地寻找流中特定点的过程。该术语来源于卷盘式磁带播放器时代，当来回摇晃卷盘以定位某个部分时，就像用磁带刮拭播放头。

Media Foundation 中的定位是通过将播放速率设置为零来实现的。有关详细信息，请参阅如何执行定位。

How to Determine Supported Rates如何确定支持的速率

在更改播放速率之前，应用程序应检查管道中的对象是否支持播放速率。IMFRateSupport 接口提供了获取最大正向和反向速率、最接近请求速率的支持速率以及最慢速率的方法。每个速率查询都可以指定播放方向以及是否使用稀疏化。使用 IMFRateControl 接口查询确切的播放速率。

有关更改播放速率的信息，请参阅如何在媒体会话上设置播放速率。

有关播放速率的一般信息，请参阅关于速率控制。

How to Set the Playback Rate on the Media Session 如何设置媒体会话的播放速率

为了实现快进和快退等播放功能，应用程序可能需要更改媒体流的播放速率。Media Foundation 提供应用程序必须使用的速率控制服务来动态设置播放速率。

在设置播放速率之前，应用程序应检查媒体源是否支持该速率。

- 调用 MFGetService 从媒体会话中获取速率控制对象。
  - 调用 MFGetService 的应用程序必须确保以下几点：
    - punkObject 参数包含已初始化的 IMFMediaSession 接口指针。
    - 释放 ppvObject 参数中接收的速率控制对象，以避免内存泄漏。
- 调用IMFRateControl::SetRate方法设置播放速率。SetRate 异步完成后，应用程序会收到 MESessionRateChanged 事件。

How to Perform Scrubbing如何进行定位

执行定位是为了通过与时间的视觉表示（例如滚动条）交互来即时查找文件中的特定点。在 Media Foundation 中，清理意味着搜索文件并获取一个更新的帧。

1. 调用 MFGetService 从媒体会话中获取 IMFRateControl 接口。
2. 调用 IMFRateControl::SetRate 将播放速率设置为零。有关调用此方法的更多信息，请参阅如何设置媒体会话的播放速率。
3. 通过在 MFTIME 类型中指定要查找的呈现时间，在 PROPVARIANT 中创建查找位置。
4. 以查找位置调用 IMFMediaSession::Start 开始播放。
5. 当清理操作完成时，媒体会话会发送 MESessionScrubSampleComplete 事件。等待此事件，然后再次调用 Start 进行另一个清理操作。

示例代码参考这里。

Implementing Rate Control 实现速率控制

如果您正在编写 Microsoft Media Foundation 管道对象（媒体源、转换或媒体接收器），则可能需要支持可变播放速率。为此，请实现以下接口：

1. 实现 IMFGetService 接口。
2. 支持 MF_RATE_CONTROL_SERVICE 服务。（请参阅服务接口）
3. 实现IMFRateSupport接口，获取对象支持的播放速率。
4. 实现IMFRateControl接口，获取或设置播放速率。

详细参考这里。

Topologies拓扑结构

拓扑是表示管道中数据流的对象。应用程序使用拓扑来准备媒体会话的管道。

拓扑是表示数据在管道中如何流动的对象。应用程序创建拓扑来描述每个流从媒体源到媒体接收器所采用的路径。应用程序将拓扑传递给媒体会话，媒体会话使用拓扑来控制数据流。

管道中的数据处理组件（媒体源、转换和媒体接收器）在拓扑中表示为节点。从一个组件到另一组件的数据流由节点之间的连接表示。定义了以下节点类型：

- 源节点：表示媒体源上的媒体流。
- 变换节点：表示媒体基础变换 (MFT)。

- 输出节点：表示媒体接收器上的流接收器。
- Tee节点：代表流中的一个分叉。Tee 节点是节点代表管道对象这一规则的一个例外。与其他节点类型不同，tee 节点只是引导数据流。

正常工作的拓扑必须包含至少一个连接到输出节点的源节点（可能通过一个或多个转换节点）。例如，下图显示了一种具有一个流的简单拓扑。

对于文件播放，变换节点可能代表解码器，输出节点代表音频或视频渲染器。对于文件编码，变换节点将表示编码器，输出节点将表示存档接收器，例如 ASF 文件接收器。
如果两个节点相连，产生数据的节点称为上游节点，接收数据的节点称为下游节点。例如，在上图中，源节点位于转换节点的上游。
在一对连接的节点中，上游节点上的连接点称为输出。下游节点上的连接点称为输入。下图显示了一对节点及其连接点以及它们之间的数据流。连接点在拓扑中不表示为单独的对象。它们由节点对象上的索引值指定。

源节点不能有任何输入。因此，源节点的上游不能有任何节点。同样，输出节点不能有输出，并且输出节点下游不能有任何节点。从源节点到输出节点的节点链称为拓扑的分支。本主题中的第一个图显示了具有单个分支的拓扑。通常，每个流有一个分支。例如，要播放具有一个音频流和一个视频流的文件，需要具有两个分支的拓扑。

Partial Topologies部分拓扑

完整的拓扑图包含了所需的每个流水线对象的节点。然而，应用程序并不总是需要创建完整的拓扑图。相反，它创建了一个省略了一个或多个转换节点的部分拓扑图。
媒体会话通过使用称为拓扑加载器的对象来完成拓扑。拓扑加载器通过插入所需的变换将部分拓扑转换为完整拓扑。转换的过程称为拓扑解析。
例如，要播放编码的音频流，拓扑必须在源节点和输出节点之间有一个解码器。该应用程序创建一个部分拓扑，将源节点直接连接到输出节点，无需解码器。拓扑加载器检查流格式，找到正确的解码器，并将转换节点插入拓扑中。
下图显示了应用程序创建的部分拓扑。

下图显示了拓扑加载器解析后的完整拓扑。在此示例中，拓扑加载器已为解码器插入了变换节点。

在当前版本的 Media Foundation 中，拓扑加载器支持播放拓扑。对于文件编码等场景，应用程序必须构建完整的拓扑。
应用程序还可以创建拓扑加载器并直接使用它。例如，您可以使用拓扑加载器解析部分拓扑，然后修改完整拓扑，然后再将其提供给媒体会话。要创建拓扑加载器，请调用 MFCreateTopoLoader。

Creating Topologies创建拓扑

创建拓扑的一般步骤如下：

1. 通过调用 MFCreateTopology 创建一个新的拓扑对象。此函数返回指向拓扑的 IMFTopology 接口的指针。
2. 最初，拓扑不包含任何节点。要为拓扑创建节点，请调用 MFCreateTopologyNode。此函数返回指向节点的 IMFTopologyNode 接口的指针。创建节点时必须指定节点类型：
   - Source node.源节点。
   - Transform node.变换节点。
   - Output node.输出节点。
   - Tee node.三通节点。
3. 初始化每个节点。初始化过程取决于节点类型，如后续主题中所述。
4. 通过调用 IMFTopology::AddNode 将每个节点添加到拓扑中。
5. 连接节点。要连接节点，请在上游节点上调用 IMFTopologyNode::ConnectOutput，并传入指向下游节点的指针。

Creating Source Nodes创建源节点

源节点代表来自媒体源的一个流。源节点必须包含指向媒体源、表示描述符和流描述符的指针。
要将源节点添加到拓扑，请执行以下操作：

1. 使用 MF_TOPOLOGY_SOURCESTREAM_NODE 标志调用 MFCreateTopologyNode 以创建源节点。
2. 在节点上设置 MF_TOPONODE_SOURCE 属性，并使用指向媒体源的指针。
3. 在节点上设置 MF_TOPONODE_PRESENTATION_DESCRIPTOR 属性，并使用指向媒体源的表示描述符的指针。
4. 在节点上设置 MF_TOPONODE_STREAM_DESCRIPTOR 属性，并使用指向流的流描述符的指针。
5. 调用 IMFTopology::AddNode 将节点添加到拓扑中。

Creating Transform Nodes创建变换节点

变换节点表示媒体基础变换 (MFT)，例如解码器或编码器。有几种不同的方法来初始化变换节点：

- 从指向 MFT 的指针。
- 来自 MFT 的 CLSID。
- 从指向 MFT 激活对象的指针。

如果要在受保护的媒体路径 (PMP) 内加载拓扑，则必须使用 CLSID 或激活对象，以便可以在受保护的进程内创建 MFT。第一种方法（使用指向 MFT 的指针）不适用于 PMP。

Creating a Transform Node from an MFT从 MFT 创建变换节点

要从 MFT 创建变换节点，请执行以下操作：

1. 创建 MFT 的实例并获取指向 MFT 的 IMFTransform 接口的指针。

2. 使用 MF_TOPOLOGY_TRANSFORM_NODE 标志调用 MFCreateTopologyNode 以创建变换节点。
3. 调用 IMFTopologyNode::SetObject 并传入 IMFTransform 指针。
4. 调用 IMFTopology::AddNode 将节点添加到拓扑中。

Creating a Transform Node from a CLSID从 CLSID 创建转换节点

要从 CLSID 创建转换节点，请执行以下操作：

1. 查找 MFT 的 CLSID。您可以使用 MFTEnum 函数按类别（例如解码器或编码器）查找 MFT 的 CLSID。您可能还知道要使用的特定 MFT 的 CLSID（例如，如果您实现了自己的自定义 MFT）。
2. 使用 MF_TOPOLOGY_TRANSFORM_NODE 标志调用 MFCreateTopologyNode 以创建变换节点。
3. 在节点上设置 MF_TOPONODE_TRANSFORM_OBJECTID 属性。该属性值是 CLSID。
4. 调用 IMFTopology::AddNode 将节点添加到拓扑中。

Creating a Transform Node from an Activation Object从激活对象创建变换节点

一些 MFT 提供激活对象。例如，MFCreateWMAEncoderActivate 函数返回 Windows Media Audio (WMA) 编码器的激活对象。确切的功能取决于 MFT。并非每个 MFT 都提供激活对象。有关更多信息，请参阅激活对象。

您还可以通过调用 MFTEnumEx 函数获取 MFT 激活对象。

要从激活对象创建变换节点，请执行以下操作：

- 创建激活对象并获取指向激活对象的 IMFActivate 接口的指针。
- 使用 MF_TOPOLOGY_TRANSFORM_NODE 标志调用 MFCreateTopologyNode 以创建变换节点。
- 调用 IMFTopologyNode::SetObject 并传入 IMFActivate 指针。
- 调用 IMFTopology::AddNode 将节点添加到拓扑中。

具体实例代码参考这里。

Creating Output Nodes创建输出节点

输出节点表示媒体接收器上的流接收器。初始化输出节点有两种方法：

- 从指针到流接收器
- 从指向媒体接收器的激活对象的指针。

如果要在受保护的媒体路径 (PMP) 内加载拓扑，则必须使用激活对象，以便可以在受保护的进程内创建媒体接收器。第一种方法（使用指向流接收器的指针）不适用于 PMP。

Creating an Output Node from a Stream Sink从 Stream Sink 创建输出节点

要从流接收器创建输出节点，请执行以下操作：

1. 创建媒体接收器的实例。

2. 使用媒体接收器的 IMFMediaSink 接口获取指向所需流接收器的指针。（IMFMediaSink 接口有多个返回指向流接收器的指针的方法。）
3. 使用 MF_TOPOLOGY_OUTPUT_NODE 标志调用 MFCreateTopologyNode 以创建输出节点。
4. 调用 IMFTopologyNode::SetObject 并将指针传递给流接收器的 IMFStreamSink 接口。
5. 将 MF_TOPONODE_NOSHUTDOWN_ON_REMOVE 属性设置为 FALSE（可选，但建议）。
6. 调用 IMFTopology::AddNode 将节点添加到拓扑中。

当应用程序关闭媒体会话时，媒体会话会自动关闭媒体接收器。因此，您不能将媒体接收器与媒体会话的另一个实例重复使用。

Creating an Output Node from an Activation Object从激活对象创建输出节点

任何受信任的媒体接收器都必须提供激活对象，以便可以在受保护的进程内创建媒体接收器。有关更多信息，请参阅激活对象。创建激活对象的特定函数将取决于媒体接收器。

要从激活对象创建输出节点，请执行以下操作：

1. 创建激活对象并获取指向激活对象的 IMFActivate 接口的指针。
2. 使用 MF_TOPOLOGY_OUTPUT_NODE 标志调用 MFCreateTopologyNode 以创建输出节点。
3. （可选）在节点上设置 MF_TOPONODE_STREAMID 属性以指定流接收器的流标识符。如果省略此属性，则节点默认使用流接收器 0。
4. 将 MF_TOPONODE_NOSHUTDOWN_ON_REMOVE 属性设置为 TRUE（可选，但建议）。
5. 调用 IMFTopologyNode::SetObject 并传入 IMFActivate 指针。
6. 调用 IMFTopology::AddNode 将节点添加到拓扑中。

示例代码参考这里。

Creating Playback Topologies 创建播放拓扑

对于基本播放，您可以创建部分拓扑，其中源节点直接连接到输出节点。您不需要为中间转换插入任何节点，例如解码器或颜色转换器。媒体会话将使用拓扑加载器来解析拓扑，拓扑加载器将插入所需的转换。

Creating the Topology创建拓扑

以下是从媒体源创建部分播放拓扑的总体步骤：

1. 创建媒体源。在大多数情况下，您将使用源解析器来创建媒体源。有关详细信息，请参阅源解析器。
2. 获取媒体源的演示描述符。
3. 创建一个空拓扑。
4. 使用表示描述符来枚举流描述符。对于每个流描述符：
    a. 获取流的主要媒体类型，例如音频或视频。
    b. 检查当前是否选择了流。（您也可以根据媒体类型选择或取消选择流。）
    c. 如果选择了流，则根据流的媒体类型为媒体接收器创建激活对象。

d. 添加流的源节点和媒体接收器的输出节点。

e. 将源节点连接到输出节点。

示例代码参考这里。

Connecting Streams to Media Sinks 将流连接到媒体接收器

对于每个选定的流，添加源节点和输出节点，然后连接这两个节点。源节点代表流。输出节点表示增强型视频渲染器 (EVR) 或流式音频渲染器 (SAR)。

Creating the Media Sink创建媒体接收器

示例代码参考这里。

Advanced Topology Building高级拓扑构建

如果您想要更好地控制发送到媒体会话的拓扑，您可以使用这些技术。

由于这些技术适用于超出标准拓扑加载器提供的功能的场景，因此许多细节将取决于应用程序的特定要求。因此，本节是围绕较小的子任务松散组织的，而不是完整的端到端场景。

典型的播放应用程序遵循以下步骤：

1. 应用程序构建部分拓扑并将其在媒体会话上排队。
2. 媒体会话调用拓扑加载器来解析拓扑。

如果您想超越拓扑加载器的功能，可以使用三种通用方法：

- 构建完整的拓扑。当您在媒体会话上对拓扑进行排队时，请使用 MFSESSION_SETTOPOLOGY_NORESOLUTION 标志调用 IMFMediaSession::SetTopology。该标志阻止媒体会话尝试解析拓扑。
- 直接调用拓扑加载器来解析拓扑。然后，您可以在将其在媒体会话上排队之前修改完整拓扑。
- 实现自定义拓扑加载器。通过这种方法，您可以对部分拓扑进行排队，但媒体会话会调用您的自定义加载程序而不是标准MF实现。这种方法的优点之一是您可以在受保护的环境中执行自定义拓扑构建。（但是，在这种情况下，拓扑加载器必须是受信任的组件。有关详细信息，请参阅受保护的媒体路径。）

Custom Topology Loaders自定义拓扑加载器

当应用程序在媒体会话上对部分拓扑进行排队时，媒体会话使用拓扑加载器来解析拓扑。默认情况下，媒体会话使用拓扑加载器的标准 Media Foundation 实现，但您也可以提供自定义实现。这使您可以更好地控制最终拓扑。自定义拓扑加载器必须实现 IMFTopoLoader 接口。

要在媒体会话上设置自定义拓扑加载器，请执行以下操作：

1. 通过调用 MFCreateAttributes 创建一个空属性存储。
2. 在属性存储上设置 MF_SESSION_TOPOLOADER 属性。该属性的值是自定义加载程序的 CLSID。
3. 调用 MFCreateMediaSession 为不受保护的内容创建媒体会话，或调用 MFCreatePMPMediaSessi

on 在受保护的媒体路径 (PMP) 内创建媒体会话。

没有必要在拓扑加载器中实现整个拓扑加载算法。作为替代方案，您可以包装标准 Media Foundation 拓扑加载器。在 IMFTopoLoader::Load 方法的实现中，根据需要修改拓扑并将修改后的拓扑传递到标准拓扑加载器。然后标准拓扑加载器完成拓扑。您还可以在将完成的拓扑传回媒体会话之前对其进行修改。

示例代码参考这里

Binding Output Nodes to Media Sinks将输出节点绑定到媒体接收器

如果您在媒体会话外部使用拓扑加载器，如何初始化拓扑中的输出节点。输出节点最初包含以下内容之一：

- IMFStreamSink 指针。
- IMFActivate 指针。

在后一种情况下，在拓扑加载器解析拓扑之前，IMFActivate 指针必须转换为 IMFStreamSink 指针。

在大多数情况下，该过程的工作原理如下：

1. 应用程序在媒体会话上对部分拓扑进行排队。
2. 对于所有输出节点，媒体会话将 IMFActivate 指针转换为 IMFStreamSink 指针。此过程称为将输出节点绑定到媒体接收器。
3. 媒体会话将修改后的拓扑发送到 IMFTopoLoader::Load 方法。

但是，如果您直接使用拓扑加载器（在媒体会话之外），您的应用程序必须在调用 IMFTopoLoader::Load 之前绑定输出节点。要绑定输出节点，请执行以下操作：

1. 调用 IMFTopologyNode::GetObject 获取节点的对象指针。
2. 查询 IMFStreamSink 接口的对象指针。如果此调用成功，则无需执行任何操作，因此请跳过其余步骤。
3. 如果上一步失败，则查询 IMFActivate 接口的对象指针。
4. 通过调用 IMFActivate::ActivateObject 创建媒体接收器。指定 IID_IMFMediaSink 以获取指向媒体接收器的 IMFMediaSink 接口的指针。
5. 查询节点的 MF_TOPONODE_STREAMID 属性。该属性的值是该节点的流接收器的标识符。如果未设置 MF_TOPONODE_STREAMID 属性，则默认流标识符为零。
6. 适当的流接收器可能已存在于媒体接收器上。要进行检查，请在媒体接收器上调用 IMFMediaSink::GetStreamSinkById。如果流接收器存在，则该方法返回指向其 IMFStreamSink 接口的指针。如果此调用失败，请尝试通过调用 IMFMediaSink::AddStreamSink 添加新的流接收器。如果两次调用都失败，则意味着媒体接收器不支持所请求的流标识符，并且该拓扑节点配置不正确。返回错误代码并跳过下一步。
7. 调用 IMFTopologyNode::SetObject，传入上一步中的 IMFStreamSink 指针。此调用替换了节点的对象指针，以便该节点包含指向流接收器的指针，而不是指向激活对象的指针。

示例代码参考这里

Adding a Decoder to a Topology将解码器添加到拓扑

对于大多数播放应用程序，您可以从发送到媒体会话的部分拓扑中省略解码器。媒体会话使用拓扑加载器来完成拓扑，并且拓扑加载器插入所需的任何解码器。但是，如果您想要选择特定的解码器，则可以手动将解码器添加到拓扑中。

以下是将解码器添加到拓扑的总体步骤。

1. 查找解码器的 CLSID。
2. 在拓扑中添加解码器节点。
3. 对于视频解码器，启用 DirectX 视频加速。音频解码器不需要此步骤。

Find the Decoder CLSID查找解码器 CLSID

如果您想使用特定的解码器，您可能已经知道该解码器的 CLSID。如果是这样，您可以跳过此步骤。否则，使用 MFTEnum 函数在注册表中查找 CLSID。该函数将多个搜索条件作为输入。要查找解码器，您只需指定输入格式（主要类型和子类型）。您可以从流描述符中获取这些，

MFTEnum 函数返回一个指向 CLSID 数组的指针。返回数组的顺序是任意的。在此示例中，该函数使用数组中的第一个 CLSID。您可以通过调用 MFTGetInfo 获取有关解码器的更多信息，包括解码器的友好名称。另请注意，MFTEnum 可以成功，但返回一个空数组，因此检查最后一个参数中返回的数组大小非常重要。

示例代码参考这里

Add the Decoder Node to the Topology将解码器节点添加到拓扑中

获得解码器的 CLSID 后，通过调用 MFCreateTopology 创建一个新的转换节点。通过在节点上设置 MF_TOPONODE_TRANSFORM_OBJECTID 属性来指定 CLSID。有关如何创建变换节点的示例，请参阅创建变换节点。然后，通过调用 IMFTopologyNode::ConnectOutput 将源节点连接到解码器节点，并将解码器节点连接到输出节点。

示例代码参考这里

Enable Video Acceleration启用视频加速

将音频或视频解码器添加到拓扑的下一步仅适用于视频解码器。为了获得最佳的视频播放性能，您应该启用 DirectX 视频加速 (DXVA)（如果视频解码器支持）。通常此步骤由拓扑加载器执行，但如果您手动将解码器添加到拓扑中，则必须自己执行此步骤。

作为此步骤的前提条件，拓扑中的所有输出节点必须绑定到媒体接收器。有关更多信息，请参阅将输出节点绑定到媒体接收器。

示例代码参考这里

Using Media Sources with the Media Session在媒体会话中使用媒体源

如果您使用媒体会话来控制播放，则应在媒体源上调用的方法集受到限制。本节介绍如何将媒体源与媒体会话结合使用。

以下是您的应用程序将执行的基本步骤：

1. 创建媒体源。要创建媒体源，请使用源解析器。有关详细信息，请参阅源解析器。源解析器返回指向源的 IMFMediaSource 接口的指针。（如果您编写了自定义媒体源，则可以提供自定义创建方法。）
2. 配置演示信息。要配置源的演示信息，请调用 IMFMediaSource::CreatePresentationDescriptor。您可以修改此副本，但更改在播放开始之前不会生效。播放期间请勿修改演示描述符。有关详细信息，请参阅呈现描述符。
3. 创建包含媒体源的拓扑。有关详细信息，请参阅拓扑。
4. 使用媒体会话来控制播放。媒体会话调用媒体源上的方法。应用程序此时不应调用媒体源上的任何方法。
5. 在释放媒体源之前，调用IMFMediaSource::Shutdown关闭媒体源。

如果您使用媒体会话，则应在媒体源上调用的唯一方法是 CreatePresentationDescriptor、GetCharacteristics 和 Shutdown。尤其：

- 请勿调用 **Start**, **Pause**, or **Stop**；这些方法只能由媒体会话调用。
- 不要调用任何 IMFMediaStream 方法。
- 不要直接从媒体源或任何流检索事件。媒体会话必须接收这些事件才能使管道正常运行。媒体会话转发应用程序所需的任何事件。

Source Reader 源阅读器

源读取器是使用媒体会话和 Microsoft Media Foundation 管道处理媒体数据的替代方法。

Why Use the Source Reader?为什么要使用源阅读器?

Media Foundation 提供了针对播放进行优化的管道。管道是端到端的，这意味着它处理从源（例如视频文件）一直到目的地（例如图形显示）的数据流。但是，如果您想在数据通过管道时读取或修改数据，则必须编写自定义插件。这需要对媒体基金会管道有相当深入的了解。对于某些任务，创建新插件的开销太大。源读取器是为这种类型的情况而设计的，当您想要从源获取原始数据而无需整个管道的开销时。

在内部，源读取器持有指向媒体源的指针。媒体源是一个媒体基础对象，它从外部源（例如媒体文件或视频捕获设备）生成媒体数据。源读取器管理对媒体源的所有方法调用。（有关媒体源的更多信息，请参阅媒体源。）

如果媒体源提供压缩数据，您可以使用源读取器来解码数据。在这种情况下，源读取器将加载正确的解码器并管理媒体源和解码器之间的数据流。源阅读器还可以执行一些有限的视频处理：从 YUV 到 RGB-32 的颜色转换以及软件去隔行，尽管不建议将这些操作用于实时视频渲染。下图说明了此过程。

源阅读器不会将数据发送到目的地；而是应该由应用程序来消费数据。例如，源阅读器可以读取视频文件，但不会将视频渲染到屏幕上。此外，源阅读器不管理演示时钟、处理计时问题或同步视频与音频。

在以下情况下考虑使用源阅读器：

- 您希望从媒体文件中获取数据，而不用担心底层文件结构。
- 您想要从音频或视频捕获设备获取数据。
- 您的数据处理任务对时间不敏感，或者您不需要演示时钟。
- 您已经有一个不基于 Media Foundation 的媒体管道，并且您希望将 Media Foundation 媒体源合并到您自己的管道中。

以下情况不建议使用源码阅读器：

- 对于受保护的内容。源阅读器不支持数字版权管理 (DRM)。
- 如果您关心底层文件结构的细节。源阅读器隐藏了此类细节。

Using the Source Reader to Process Media Data使用源读取器处理媒体数据

要使用源阅读器，请按照以下基本步骤操作：

1. 创建源阅读器的实例。
2. 枚举可能的输出格式。
3. 设置每个流的实际输出格式。
4. 处理来自源的数据。

Creating the Source Reader创建源阅读器

要创建源读取器的实例，请调用以下函数之一：

Function 功能	Description 描述
MFCreateSourceReaderFromURL	Takes a URL as input. This function uses the Source Resolver to create a media source from the URL. 将 URL 作为输入。此函数使用源解析器从 URL 创建媒体源。
MFCreateSourceReaderFromByteStream	Takes a pointer to a byte stream. This function also uses the Source Resolver to create the media source. 获取一个指向字节流的指针。此函数还使用源解析器来创建媒体源。
MFCreateSourceReaderFromMediaSource	Takes a pointer to a media source that has already been created. This function is useful for media sources that the Source Resolver cannot create, such capture devices or custom media sources.

	获取指向已创建的媒体源的指针。此功能对于源解析器无法创建的媒体源非常有用，例如捕获设备或自定义媒体源。

通常，对于媒体文件，请使用 MFCreateSourceReaderFromURL。对于网络摄像头等设备，请使用 MFCreateSourceReaderFromMediaSource。（有关 Microsoft Media Foundation 中的捕获设备的详细信息，请参阅音频/视频采集）

这些函数中的每一个都采用一个可选的 IMFAttributes 指针，该指针用于在源读取器上设置各种选项，如这些函数的参考主题中所述。要获得默认行为，请将此参数设置为 NULL。每个函数都返回一个 IMFSourceReader 指针作为输出参数。在调用任何这些函数之前，您必须先依次调用 CoInitialize(Ex) 和 MFStartup 函数。

示例代码参考这里。

Enumerating Output Formats枚举输出格式

每个媒体源至少有一个流。例如，视频文件可能包含视频流和音频流。每个流的格式使用媒体类型来描述，由 IMFMediaType 接口表示。有关媒体类型的更多信息，请参阅媒体类型。您必须检查媒体类型以了解从源读取器获取的数据的格式。

最初，每个流都有一个默认格式，您可以通过调用 IMFSourceReader::GetCurrentMediaType 方法找到该格式：

对于每个流，媒体源提供该流可能的媒体类型的列表。类型的数量取决于来源。如果源表示媒体文件，则每个流通常只有一种类型。另一方面，网络摄像头可能能够以多种不同格式传输视频。在这种情况下，应用程序可以从媒体类型列表中选择要使用的格式。

要获取流的媒体类型之一，请调用 IMFSourceReader::GetNativeMediaType 方法。此方法采用两个索引参数：流的索引和流的媒体类型列表的索引。要枚举流的所有类型，请增加列表索引，同时保持流索引不变。当列表索引超出范围时，GetNativeMediaType 返回 MF_E_NO_MORE_TYPES。

要枚举每个流的媒体类型，请增加流索引。当流索引超出范围时，GetNativeMediaType 返回 MF_E_INVALIDSTREAMNUMBER。

Setting Output Formats设置输出格式

要更改输出格式，请调用 IMFSourceReader::SetCurrentMediaType 方法。此方法采用流索引和媒体类型：

```
hr = pReader->SetCurrentMediaType(dwStreamIndex, pMediaType);
```

对于媒体类型，取决于是否要插入解码器。

- 要直接从源获取数据而不对其进行解码，请使用 GetNativeMediaType 返回的类型之一。
- 要解码流，请创建一个描述所需未压缩格式的新媒体类型。

对于解码器，创建媒体类型如下：

1. 调用 MFCreateMediaType 创建新的媒体类型。
2. 设置 MF_MT_MAJOR_TYPE 属性以指定音频或视频。
3. 设置MF_MT_SUBTYPE属性来指定解码格式的子类型。（请参阅音频子类型 GUID 和视频子类型 GUID。）
4. 调用 IMFSourceReader::SetCurrentMediaType。

源阅读器将自动加载解码器。要获取解码格式的完整详细信息，请在调用 SetCurrentMediaType 之后调用 IMFSourceReader::GetCurrentMediaType。

Processing Media Data处理媒体数据

要从源获取媒体数据，请调用 IMFSourceReader::ReadSample 方法，如以下代码所示。

```
DWORD streamIndex, flags;
LONGLONG llTimeStamp;

hr = pReader->ReadSample(
    MF_SOURCE_READER_ANY_STREAM,    // Stream index.
    0,                              // Flags.
    &streamIndex,                   // Receives the actual stream index.
    &flags,                         // Receives status flags.
    &llTimeStamp,                   // Receives the time stamp.
    &pSample                        // Receives the sample or NULL.
    );
```

第一个参数是要获取数据的流的索引。您还可以指定 MF_SOURCE_READER_ANY_STREAM 以从任何流中获取下一个可用数据。第二个参数包含可选标志；有关这些列表，请参阅 MF_SOURCE_READER_CONTROL_FLAG。第三个参数接收实际产生数据的流的索引。如果将第一个参数设置为 MF_SOURCE_READER_ANY_STREAM，则将需要此信息。第四个参数接收状态标志，指示读取数据时可能发生的各种事件，例如流中的格式更改。有关状态标志的列表，请参阅 MF_SOURCE_READER_FLAG。

如果媒体源能够为所请求的流生成数据，则 ReadSample 的最后一个参数接收指向媒体样本对象的 IMFSample 接口的指针。使用媒体样本可以：

- 获取指向媒体数据的指针。
- 获取演示时间和样本持续时间。
- 获取描述隔行扫描、场域和样本其他方面的属性。

媒体数据的内容取决于流的格式。对于未压缩的视频流，每个媒体样本包含一个视频帧。对于未压缩的音频流，每个媒体样本都包含一系列音频帧。

ReadSample 方法可以返回 S_OK，但不返回 pSample 参数中的媒体样本。例如，当到达文件末尾时，ReadSample 会在 dwFlags 中设置 MF_SOURCE_READERF_ENDOFSTREAM 标志，并将 pSample 设置为 NULL。在这种情况下，ReadSample 方法返回 S_OK，因为没有发生错误，即使 pSample 参数设置为 NULL。因此，在取消引用 pSample 之前，请务必检查它的值。

以下代码演示如何循环调用 ReadSample 并检查该方法返回的信息，直到到达媒体文件末尾。

```
HRESULT ProcessSamples(IMFSourceReader *pReader)
{
    HRESULT hr = S_OK;
    IMFSample *pSample = NULL;
    size_t  cSamples = 0;

    bool quit = false;
    while (!quit)
    {
        DWORD streamIndex, flags;
        LONGLONG llTimeStamp;

        hr = pReader->ReadSample(
            MF_SOURCE_READER_ANY_STREAM,    // Stream index.
            0,                              // Flags.
            &streamIndex,                   // Receives the actual stream index.
            &flags,                         // Receives status flags.
            &llTimeStamp,                   // Receives the time stamp.
            &pSample                        // Receives the sample or NULL.
            );

        if (FAILED(hr))
        {
            break;
        }

        wprintf(L"Stream %d (%I64d)\n", streamIndex, llTimeStamp);
        if (flags & MF_SOURCE_READERF_ENDOFSTREAM)
        {
            wprintf(L"\tEnd of stream\n");
```

```
            quit = true;
        }
        if (flags & MF_SOURCE_READERF_NEWSTREAM)
        {
            wprintf(L"\tNew stream\n");
        }
        if (flags & MF_SOURCE_READERF_NATIVEMEDIATYPECHANGED)
        {
            wprintf(L"\tNative type changed\n");
        }
        if (flags & MF_SOURCE_READERF_CURRENTMEDIATYPECHANGED)
        {
            wprintf(L"\tCurrent type changed\n");
        }
        if (flags & MF_SOURCE_READERF_STREAMTICK)
        {
            wprintf(L"\tStream tick\n");
        }

        if (flags & MF_SOURCE_READERF_NATIVEMEDIATYPECHANGED)
        {
            // The format changed. Reconfigure the decoder.
            hr = ConfigureDecoder(pReader, streamIndex);
            if (FAILED(hr))
            {
                break;
            }
        }

        if (pSample)
        {
            ++cSamples;
        }

        SafeRelease(&pSample);
    }

    if (FAILED(hr))
    {
```

```
        wprintf(L"ProcessSamples FAILED, hr = 0x%x\n", hr);
    }
    else
    {
        wprintf(L"Processed %d samples\n", cSamples);
    }
    SafeRelease(&pSample);
    return hr;
}
```

Draining the Data Pipeline排空数据管道

在数据处理期间，解码器或其他变换可能会缓冲输入样本。在下图中，应用程序调用 ReadSample 并接收演示时间等于 t1 的样本。解码器保存 t2 和 t3 的样本。

在下次调用 ReadSample 时，Source Reader可能会将 t4 提供给解码器并将 t2 返回给应用程序。如果要解码解码器中当前缓冲的所有样本，而不将任何新样本传递给解码器，请在 ReadSample 的 dwControlFlags 参数中设置 MF_SOURCE_READER_CONTROLF_DRAIN 标志。继续循环执行此操作，直到 ReadSample 返回 NULL 样本指针。根据解码器缓冲样本的方式，这可能会立即发生，也可能在多次调用 ReadSample 之后发生。

Getting the File Duration获取文件时长

要获取媒体文件的持续时间，请调用 IMFSourceReader::GetPresentationAttribute 方法并请求 MF_PD_DURATION 属性，如以下代码所示。

```
HRESULT GetDuration(IMFSourceReader *pReader, LONGLONG *phnsDuration)
{
    PROPVARIANT var;
    HRESULT hr = pReader->GetPresentationAttribute(MF_SOURCE_READER_MEDIASOURCE,
        MF_PD_DURATION, &var);
    if (SUCCEEDED(hr))
    {
        hr = PropVariantToInt64(var, phnsDuration);
        PropVariantClear(&var);
    }
    return hr;
}
```

此处显示的函数获取以 100 纳秒为单位的持续时间。除以 10,000,000 即可得到持续时间（以秒为单位）。

Seeking 定位

从本地文件获取数据的媒体源通常可以查找文件中的任意位置。网络摄像头等捕获设备通常无法搜索，因为数据是实时的。通过网络流式传输数据的源可能能够进行搜索，具体取决于网络流式传输协议。

要查明媒体源是否支持定位功能，请调用 IMFSourceReader::GetPresentationAttribute 并请求 MF_SOURCE_READER_MEDIASOURCE_CHARACTERISTICS 属性，如以下代码所示：

```
HRESULT GetSourceFlags(IMFSourceReader *pReader, ULONG *pulFlags)
{
    ULONG flags = 0;

    PROPVARIANT var;
    PropVariantInit(&var);

    HRESULT hr = pReader->GetPresentationAttribute(
        MF_SOURCE_READER_MEDIASOURCE,
        MF_SOURCE_READER_MEDIASOURCE_CHARACTERISTICS,
        &var);

    if (SUCCEEDED(hr))
    {
        hr = PropVariantToUInt32(var, &flags);
    }
    if (SUCCEEDED(hr))
    {
        *pulFlags = flags;
    }

    PropVariantClear(&var);
    return hr;
}
```

该函数从源获取一组功能标志。这些标志在 MFMEDIASOURCE_CHARACTERISTICS 枚举中定义。两个标志与定位相关：

Flag	Description
MFMEDIASOURCE_CAN_SEEK	The source can seek. 源码可求。
MFMEDIASOURCE_HAS_SLOW_SEEK	Seeking might take a long time to complete. For example, the source might need to download the entire file before it can seek. (There are no strict criteria for a source to return this flag.) 搜索可能需要很长时间才能完成。例如，源可能需要下载整个文件才能进行查找。（对于源返回此标志没有严格的标准。）

以下代码测试 MFMEDIASOURCE_CAN_SEEK 标志。

```
BOOL SourceCanSeek(IMFSourceReader *pReader)
{
    BOOL bCanSeek = FALSE;
    ULONG flags;
    if (SUCCEEDED(GetSourceFlags(pReader, &flags)))
    {
        bCanSeek = ((flags & MFMEDIASOURCE_CAN_SEEK) == MFMEDIASOURCE_CAN_SEEK);
    }
    return bCanSeek;
}
```

要定位，请调用 IMFSourceReader::SetCurrentPosition 方法，如以下代码所示。

```
HRESULT SetPosition(IMFSourceReader *pReader, const LONGLONG& hnsPosition)
{
    PROPVARIANT var;
    HRESULT hr = InitPropVariantFromInt64(hnsPosition, &var);
    if (SUCCEEDED(hr))
    {
        hr = pReader->SetCurrentPosition(GUID_NULL, var);
        PropVariantClear(&var);
    }
    return hr;
}
```

第一个参数给出用于指定搜索位置的时间格式。Media Foundation 中的所有媒体源都需要支持 100 纳秒单位，由值 GUID_NULL 指示。第二个参数是包含搜索位置的 PROPVARIANT。对于 100 纳秒时间单位，数据类型为 LONGLONG。

请注意，并非每个媒体源都提供帧精确搜索。查找的准确性取决于几个因素，例如关键帧间隔、媒体文件是否包含索引以及数据是否具有恒定或可变的比特率。因此，在您查找文件中的某个位置后，无法保证下一个样本上的时间戳与所请求的位置完全匹配。通常，实际位置不会晚于请求的位置，因此您可以丢弃样本，直到到达流中的所需点。

Playback Rate 播放速率

虽然您可以使用源阅读器设置播放速率，但这样做通常不是很有用，原因如下：

- 源阅读器不支持反向播放，即使媒体源支持。
- 应用程序控制演示时间，因此应用程序可以实现快速或慢速播放，而无需在源上设置速率。
- 某些媒体源支持抽帧（**thinning** ）模式，其中源提供较少的样本 - 通常只是关键帧。但是，如果要删除非关键帧，可以检查每个样本的 MFSampleExtension_CleanPoint 属性。

要使用 Source Reader 设置播放速率，请调用 IMFSourceReader::GetServiceForStream 方法以从媒体源获取 IMFRateSupport 和 IMFRateControl 接口。

Hardware Acceleration 硬件加速

Source Reader 与 Microsoft DirectX 视频加速 (DXVA) 2.0 兼容，用于硬件加速视频解码。要将 DXVA 与 Source Reader 结合使用，请执行以下步骤。

1. 创建 Microsoft Direct3D 设备。
2. 调用DXVA2CreateDirect3DDeviceManager9函数创建Direct3D设备管理器。该函数获取指向 IDirect3DDeviceManager9 接口的指针。
3. 使用指向 Direct3D 设备的指针调用 IDirect3DDeviceManager9::ResetDevice 方法。
4. 通过调用 MFCreateAttributes 函数创建属性存储。
5. 创建源阅读器。在创建函数的 pAttributes 参数中传递属性存储。

当您提供 Direct3D 设备时，源读取器会分配与 DXVA 视频处理器 API 兼容的视频样本。您可以使用 DXVA 视频处理来执行硬件去隔行或视频混合。有关详细信息，请参阅 DXVA 视频处理。另外，如果解码器支持DXVA 2.0，它将使用Direct3D设备来执行硬件加速解码。

Using the Source Reader in Asynchronous Mode在异步模式下使用源读取器

在异步模式下，应用程序提供回调接口，用于通知应用程序有数据可用。

Using Asynchronous Mode使用异步模式

源读取器以同步模式或异步模式运行。上一节中显示的代码示例假设源读取器使用同步模式，这是默认模式。在同步模式下，IMFSourceReader::ReadSample 方法会在媒体源生成下一个样本时阻塞。媒

体源通常从某些外部源（例如本地文件或网络连接）获取数据，因此该方法可能会在相当长的时间内阻塞调用线程。

在异步模式下，ReadSample 立即返回，工作在另一个线程上执行。操作完成后，Source Reader通过IMFSourceReaderCallback回调接口调用应用程序。要使用异步模式，必须在首次创建 Source Reader 时提供回调指针，如下所示：

1. 通过调用 MFCreateAttributes 函数创建属性存储。
2. 在属性存储上设置 MF_SOURCE_READER_ASYNC_CALLBACK 属性。该属性值是指向回调对象的指针。
3. 创建源读取器时，将属性存储传递到 pAttributes 参数中的创建函数。所有创建 Source Reader 的函数都有这个参数。

以下示例显示了这些步骤。

```
HRESULT CreateSourceReaderAsync(
    PCWSTR pszURL,
    IMFSourceReaderCallback *pCallback,
    IMFSourceReader **ppReader)
{
    HRESULT hr = S_OK;
    IMFAttributes *pAttributes = NULL;

    hr = MFCreateAttributes(&pAttributes, 1);
    if (FAILED(hr))
    {
        goto done;
    }

    hr = pAttributes->SetUnknown(MF_SOURCE_READER_ASYNC_CALLBACK, pCallback);
    if (FAILED(hr))
    {
        goto done;
    }

    hr = MFCreateSourceReaderFromURL(pszURL, pAttributes, ppReader);

done:
    SafeRelease(&pAttributes);
    return hr;
}
```

创建 Source Reader 后，您将无法在同步和异步模式之间切换。
若要以异步方式获取数据，请调用 ReadSample 方法，但将最后四个参数设置为 NULL，如下例所示。

```
// Request the first sample.
hr = pReader->ReadSample(MF_SOURCE_READER_FIRST_VIDEO_STREAM,
0, NULL, NULL, NULL, NULL);
```

当 ReadSample 方法异步完成时，Source Reader 调用您的 IMFSourceReaderCallback::OnReadSample 方法。该方法有五个参数：

- hrStatus：包含 HRESULT 值。这与 ReadSample 在同步模式下返回的值相同。如果 hrStatus 包含错误代码，您可以忽略其余参数。
- dwStreamIndex、dwStreamFlags、llTimestamp 和 pSample：这三个参数相当于 ReadSample 中的最后三个参数。它们分别包含流编号、状态标志和 IMFSample 指针。

```
STDMETHODIMP OnReadSample(HRESULT hrStatus, DWORD dwStreamIndex, DWORD
dwStreamFlags, LONGLONG llTimestamp, IMFSample *pSample);
```

此外，回调接口还定义了另外两个方法：

- OnEvent。当媒体源中发生某些事件（例如缓冲或网络连接事件）时通知应用程序。
- OnFlush。当 Flush 方法完成时调用。

Implementing the Callback Interface实现回调接口

回调接口必须是线程安全的，因为 OnReadSample 和其他回调方法是从工作线程调用的。
实现回调时可以采用多种不同的方法。例如，您可以在回调内完成所有工作，也可以使用回调通知应用程序（例如，通过发出事件句柄信号），然后从应用程序线程执行工作。
每次调用 IMFSourceReader::ReadSample 方法时，都会调用一次 OnReadSample 方法。要获取下一个示例，请再次调用 ReadSample。如果发生错误，将使用 hrStatus 参数的错误代码调用 OnReadSample。
以下示例显示了回调接口的最小实现。首先，这是实现该接口的类的声明。

```
#include <shlwapi.h>

class SourceReaderCB : public IMFSourceReaderCallback
```

```
{
public:
    SourceReaderCB(HANDLE hEvent) :
      m_nRefCount(1), m_hEvent(hEvent), m_bEOS(FALSE), m_hrStatus(S_OK)
    {
        InitializeCriticalSection(&m_critsec);
    }

    // IUnknown methods
    STDMETHODIMP QueryInterface(REFIID iid, void** ppv)
    {
        static const QITAB qit[] =
        {
            QITABENT(SourceReaderCB, IMFSourceReaderCallback),
            { 0 },
        };
        return QISearch(this, qit, iid, ppv);
    }
    STDMETHODIMP_(ULONG) AddRef()
    {
        return InterlockedIncrement(&m_nRefCount);
    }
    STDMETHODIMP_(ULONG) Release()
    {
        ULONG uCount = InterlockedDecrement(&m_nRefCount);
        if (uCount == 0)
        {
            delete this;
        }
        return uCount;
    }

    // IMFSourceReaderCallback methods
    STDMETHODIMP OnReadSample(HRESULT hrStatus, DWORD dwStreamIndex,
        DWORD dwStreamFlags, LONGLONG llTimestamp, IMFSample *pSample);

    STDMETHODIMP OnEvent(DWORD, IMFMediaEvent *)
    {
        return S_OK;
    }
```

```
44
45      STDMETHODIMP OnFlush(DWORD)
46      {
47          return S_OK;
48      }
49
50  public:
51      HRESULT Wait(DWORD dwMilliseconds, BOOL *pbEOS)
52      {
53          *pbEOS = FALSE;
54
55          DWORD dwResult = WaitForSingleObject(m_hEvent, dwMilliseconds);
56          if (dwResult == WAIT_TIMEOUT)
57          {
58              return E_PENDING;
59          }
60          else if (dwResult != WAIT_OBJECT_0)
61          {
62              return HRESULT_FROM_WIN32(GetLastError());
63          }
64
65          *pbEOS = m_bEOS;
66          return m_hrStatus;
67      }
68
69  private:
70
71      // Destructor is private. Caller should call Release.
72      virtual ~SourceReaderCB()
73      {
74      }
75
76      void NotifyError(HRESULT hr)
77      {
78          wprintf(L"Source Reader error: 0x%X\n", hr);
79      }
80
81  private:
82      long                m_nRefCount;        // Reference count.
```

```
    CRITICAL_SECTION    m_critsec;
    HANDLE              m_hEvent;
    BOOL                m_bEOS;
    HRESULT             m_hrStatus;

};
```

在此示例中，我们对 OnEvent 和 OnFlush 方法不感兴趣，因此它们只是返回 S_OK。该类使用事件句柄向应用程序发出信号；该句柄是通过构造函数提供的。

在这个最小的示例中，OnReadSample 方法仅将时间戳打印到控制台窗口。然后它存储状态代码和流结束标志，并向事件句柄发出信号：

```
HRESULT SourceReaderCB::OnReadSample(
    HRESULT hrStatus,
    DWORD /* dwStreamIndex */,
    DWORD dwStreamFlags,
    LONGLONG llTimestamp,
    IMFSample *pSample      // Can be NULL
    )
{
    EnterCriticalSection(&m_critsec);

    if (SUCCEEDED(hrStatus))
    {
        if (pSample)
        {
            // Do something with the sample.
            wprintf(L"Frame @ %I64d\n", llTimestamp);
        }
    }
    else
    {
        // Streaming error.
        NotifyError(hrStatus);
    }

    if (MF_SOURCE_READERF_ENDOFSTREAM & dwStreamFlags)
    {
```

```
        // Reached the end of the stream.
        m_bEOS = TRUE;
    }
    m_hrStatus = hrStatus;

    LeaveCriticalSection(&m_critsec);
    SetEvent(m_hEvent);
    return S_OK;
}
```

以下代码显示应用程序将使用此回调类从媒体文件中读取所有视频帧：

```
HRESULT ReadMediaFile(PCWSTR pszURL)
{
    HRESULT hr = S_OK;

    IMFSourceReader *pReader = NULL;
    SourceReaderCB *pCallback = NULL;

    HANDLE hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
    if (hEvent == NULL)
    {
        hr = HRESULT_FROM_WIN32(GetLastError());
        goto done;
    }

    // Create an instance of the callback object.
    pCallback = new (std::nothrow) SourceReaderCB(hEvent);
    if (pCallback == NULL)
    {
        hr = E_OUTOFMEMORY;
        goto done;
    }

    // Create the Source Reader.
    hr = CreateSourceReaderAsync(pszURL, pCallback, &pReader);
    if (FAILED(hr))
    {
```

```
        goto done;
    }

    hr = ConfigureDecoder(pReader, MF_SOURCE_READER_FIRST_VIDEO_STREAM);
    if (FAILED(hr))
    {
        goto done;
    }

    // Request the first sample.
    hr = pReader->ReadSample(MF_SOURCE_READER_FIRST_VIDEO_STREAM,
        0, NULL, NULL, NULL, NULL);
    if (FAILED(hr))
    {
        goto done;
    }

    while (SUCCEEDED(hr))
    {
        BOOL bEOS;
        hr = pCallback->Wait(INFINITE, &bEOS);
        if (FAILED(hr) || bEOS)
        {
            break;
        }
        hr = pReader->ReadSample(MF_SOURCE_READER_FIRST_VIDEO_STREAM,
            0, NULL, NULL, NULL, NULL);
    }

done:
    SafeRelease(&pReader);
    SafeRelease(&pCallback);
    return hr;
}
```

Tutorial: Decoding Audio教程：解码音频

在本教程中，您将创建一个带有两个命令行参数的控制台应用程序：包含音频流的输入文件的名称和输出文件名。应用程序从输入文件中读取五秒的音频数据，并将音频作为 WAVE 数据写入输出文件。

详情参考这里。

Protected Media Path受保护的媒体路径

本主题讨论三个相互关联的主题：受保护的环境、媒体互操作性网关以及撤销和续订。
详情参考这里。

Supported Media Formats in Media Foundation MF支持的媒体格式

详情参考这里

ASF Support in Media Foundation媒体基金会对 ASF 的支持

详情参考这里

Audio/Video Capture音频/视频捕捉

Audio/Video Capture in Media FoundationMedia Foundation 中的音频/视频采集

Microsoft Media Foundation 支持音频和视频捕获。视频捕获设备通过 UVC 类驱动程序支持，并且必须与 UVC 1.1 兼容。音频捕获设备通过 Windows 音频会话 API (WASAPI) 支持。
捕获设备在 Media Foundation 中由媒体源对象表示，该对象公开 IMFMediaSource 接口。大多数情况下，应用程序不会直接使用该接口，而是使用更高级别的 API（例如 Source Reader）来控制捕获设备。

Enumerate Capture Devices枚举捕获设备

要枚举系统上的捕获设备，请执行以下步骤：

1. 调用 MFCreateAttributes 函数创建属性存储。
2. 将 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE 属性设置为以下值之一：
   MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_AUDCAP_GUID，
   MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_GUID
3. 调用 MFEnumDeviceSources 函数。该函数分配 IMFActivate 指针数组。每个指针代表系统上一个设备的激活对象。
4. 调用 IMFActivate::ActivateObject 方法从激活对象之一创建媒体源的实例。

以下示例为枚举列表中的第一个视频捕获设备创建媒体源：

```
HRESULT CreateVideoCaptureDevice(IMFMediaSource **ppSource)
{
    *ppSource = NULL;

    UINT32 count = 0;

    IMFAttributes *pConfig = NULL;
    IMFActivate **ppDevices = NULL;

    // Create an attribute store to hold the search criteria.
    HRESULT hr = MFCreateAttributes(&pConfig, 1);

    // Request video capture devices.
    if (SUCCEEDED(hr))
    {
        hr = pConfig->SetGUID(
            MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE,
            MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_GUID
            );
    }

    // Enumerate the devices,
    if (SUCCEEDED(hr))
    {
        hr = MFEnumDeviceSources(pConfig, &ppDevices, &count);
    }

    // Create a media source for the first device in the list.
    if (SUCCEEDED(hr))
    {
        if (count > 0)
        {
            hr = ppDevices[0]->ActivateObject(IID_PPV_ARGS(ppSource));
        }
        else
        {
            hr = MF_E_NOT_FOUND;
        }
    }
```

```
    for (DWORD i = 0; i < count; i++)
    {
        ppDevices[i]->Release();
    }
    CoTaskMemFree(ppDevices);
    return hr;
}
```

您可以查询激活对象的各种属性，包括以下属性：

- MF_DEVSOURCE_ATTRIBUTE_FRIENDLY_NAME 属性包含设备的显示名称。显示名称适合向用户显示，但可能不是唯一的。
- 对于视频设备，MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_SYMBOLIC_LINK 属性包含设备的符号链接。符号链接唯一标识系统上的设备，但不是可读的字符串。
- 对于音频设备，MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_AUDCAP_ENDPOINT_ID 属性包含设备的音频端点 ID。音频端点ID类似于符号链接。它唯一标识系统上的设备，但不是可读的字符串。

以下示例采用 IMFActivate 指针数组并将每个设备的显示名称打印到调试窗口：

```
void DebugShowDeviceNames(IMFActivate **ppDevices, UINT count)
{
    for (DWORD i = 0; i < count; i++)
    {
        HRESULT hr = S_OK;
        WCHAR *szFriendlyName = NULL;

        // Try to get the display name.
        UINT32 cchName;
        hr = ppDevices[i]->GetAllocatedString(
            MF_DEVSOURCE_ATTRIBUTE_FRIENDLY_NAME,
            &szFriendlyName, &cchName);

        if (SUCCEEDED(hr))
        {
            OutputDebugString(szFriendlyName);
            OutputDebugString(L"\n");
        }
        CoTaskMemFree(szFriendlyName);
```

```
    }
}
```

如果您已经知道视频设备的符号链接，则还有另一种方法可以为设备创建媒体源：

1. 调用 MFCreateAttributes 创建属性存储。
2. 将 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE 属性设置为 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_GUID。
3. 将 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_SYMBOLIC_LINK 属性设置为符号链接。
4. 调用 MFCreateDeviceSource 或 MFCreateDeviceSourceActivate 函数。前者返回 IMFMediaSource 指针。后者返回一个指向激活对象的 IMFActivate 指针。您可以使用激活对象来创建源。（激活对象可以编组到另一个进程，因此如果您想在另一个进程中创建源，它会很有用。有关更多信息，请参阅激活对象。）

以下示例采用视频设备的符号链接并创建媒体源。

```
HRESULT CreateVideoCaptureDevice(PCWSTR *pszSymbolicLink, IMFMediaSource **ppSource)
{
    *ppSource = NULL;

    IMFAttributes *pAttributes = NULL;
    IMFMediaSource *pSource = NULL;

    HRESULT hr = MFCreateAttributes(&pAttributes, 2);

    // Set the device type to video.
    if (SUCCEEDED(hr))
    {
        hr = pAttributes->SetGUID(
            MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE,
            MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_GUID
            );
    }


    // Set the symbolic link.
    if (SUCCEEDED(hr))
    {
```

```
        hr = pAttributes->SetString(
            MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_SYMBOLIC_LINK,
            (LPCWSTR)pszSymbolicLink
            );
    }

    if (SUCCEEDED(hr))
    {
        hr = MFCreateDeviceSource(pAttributes, ppSource);
    }

    SafeRelease(&pAttributes);
    return hr;
}
```

有一种等效的方法可以根据音频端点 ID 创建音频设备：

1. 调用 MFCreateAttributes 创建属性存储。
2. 将 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE 属性设置为 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_AUDCAP_GUID。
3. 将 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_AUDCAP_ENDPOINT_ID 属性设置为端点 ID。
4. 调用 MFCreateDeviceSource 或 MFCreateDeviceSourceActivate 函数。

以下示例采用音频端点 ID 并创建媒体源。

```
HRESULT CreateAudioCaptureDevice(PCWSTR *pszEndPointID, IMFMediaSource
**ppSource)
{
    *ppSource = NULL;

    IMFAttributes *pAttributes = NULL;
    IMFMediaSource *pSource = NULL;

    HRESULT hr = MFCreateAttributes(&pAttributes, 2);

    // Set the device type to audio.
    if (SUCCEEDED(hr))
    {
        hr = pAttributes->SetGUID(
```

```
            MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE,
            MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_AUDCAP_GUID
            );
    }

    // Set the endpoint ID.
    if (SUCCEEDED(hr))
    {
        hr = pAttributes->SetString(
            MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_AUDCAP_ENDPOINT_ID,
            (LPCWSTR)pszEndPointID
            );
    }

    if (SUCCEEDED(hr))
    {
        hr = MFCreateDeviceSource(pAttributes, ppSource);
    }

    SafeRelease(&pAttributes);
    return hr;
}
```

Use a capture device使用捕获设备

为捕获设备创建媒体源后，使用源读取器从设备获取数据。源读取器提供包含捕获的音频数据或视频帧的媒体样本。下一步取决于您的应用场景：

- 视频预览：使用 Microsoft Direct3D 或 Direct2D 显示视频。
- 文件捕获：使用Sink Writer对文件进行编码。
- 音频预览：使用WASAPI。

如果您想将音频捕获与视频捕获结合起来，请使用聚合媒体源。聚合媒体源包含媒体源的集合，并将其所有流组合成单个媒体源对象。要创建聚合媒体源的实例，请调用 MFCreateAggregateSource 函数。

Shut down the capture device关闭捕获设备

当不再需要捕获设备时，必须通过调用 MFCreateDeviceSource 或 IMFActivate::ActivateObject 获得的 IMFMediaSource 对象上的 Shutdown 来关闭设备。未能调用 Shutdown 可能会导致内存链接，因为系统可能会保留对 IMFMediaSource 资源的引用，直到调用 Shutdown 为止。

```
if (g_pSource)
{
    g_pSource->Shutdown();
    g_pSource->Release();
    g_pSource = NULL;
}
```

如果您分配了一个包含到捕获设备的符号链接的字符串，您也应该释放该对象。

```
CoTaskMemFree(g_pwszSymbolicLink);
g_pwszSymbolicLink = NULL;

g_cchSymbolicLink = 0;
```

Video Capture (Microsoft Media Foundation) 视频采集（WMF）

Enumerating Video Capture Devices 枚举视频捕获设备

要枚举系统上的视频捕获设备，请执行以下操作：

1. 调用 MFCreateAttributes 创建属性存储。该函数接收 IMFAttributes 指针。
2. 调用 IMFAttributes::SetGUID 设置 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE 属性。将属性值设置为 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_GUID。
3. 调用 MFEnumDeviceSources。该函数接收 IMFActivate 指针数组和数组大小。每个指针代表一个不同的视频捕获设备。

要创建捕获设备的实例：

- 调用 IMFActivate::ActivateObject 以获取指向 IMFMediaSource 接口的指针。

以下代码显示了这些步骤：

```
HRESULT CreateVideoDeviceSource(IMFMediaSource **ppSource)
{
    *ppSource = NULL;

    IMFMediaSource *pSource = NULL;
    IMFAttributes *pAttributes = NULL;
    IMFActivate **ppDevices = NULL;
```

```
8
9       // Create an attribute store to specify the enumeration parameters.
10      HRESULT hr = MFCreateAttributes(&pAttributes, 1);
11      if (FAILED(hr))
12      {
13          goto done;
14      }
15
16      // Source type: video capture devices
17      hr = pAttributes->SetGUID(
18          MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE,
19          MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_GUID
20          );
21      if (FAILED(hr))
22      {
23          goto done;
24      }
25
26      // Enumerate devices.
27      UINT32 count;
28      hr = MFEnumDeviceSources(pAttributes, &ppDevices, &count);
29      if (FAILED(hr))
30      {
31          goto done;
32      }
33
34      if (count == 0)
35      {
36          hr = E_FAIL;
37          goto done;
38      }
39
40      // Create the media source object.
41      hr = ppDevices[0]->ActivateObject(IID_PPV_ARGS(&pSource));
42      if (FAILED(hr))
43      {
44          goto done;
45      }
46
47      *ppSource = pSource;
```

```
    (*ppSource)->AddRef();

done:
    SafeRelease(&pAttributes);

    for (DWORD i = 0; i < count; i++)
    {
        SafeRelease(&ppDevices[i]);
    }
    CoTaskMemFree(ppDevices);
    SafeRelease(&pSource);
    return hr;
}
```

创建媒体源后，释放接口指针并释放数组的内存：

```
SafeRelease(&pAttributes);

for (DWORD i = 0; i < count; i++)
{
    SafeRelease(&ppDevices[i]);
}
CoTaskMemFree(ppDevices);
```

Handling Video Device Loss处理视频设备丢失

Register For Device Notification注册设备通知

在开始从设备捕获之前，请调用 RegisterDeviceNotification 函数来注册设备通知。注册 KSCATEGORY_CAPTURE 设备类，如以下代码所示。

```
#include <Dbt.h>
#include <ks.h>
#include <ksmedia.h>

HDEVNOTIFY  g_hdevnotify = NULL;

```

```
BOOL RegisterForDeviceNotification(HWND hwnd)
{
    DEV_BROADCAST_DEVICEINTERFACE di = { 0 };
    di.dbcc_size = sizeof(di);
    di.dbcc_devicetype  = DBT_DEVTYP_DEVICEINTERFACE;
    di.dbcc_classguid  = KSCATEGORY_CAPTURE;

    g_hdevnotify = RegisterDeviceNotification(
        hwnd,
        &di,
        DEVICE_NOTIFY_WINDOW_HANDLE
        );

    if (g_hdevnotify == NULL)
    {
        return FALSE;
    }

    return TRUE;
}
```

Get the Symbolic Link of the Device获取设备的符号链接

枚举系统上的视频设备，如枚举视频捕获设备中所述。从列表中选择一个设备，然后查询激活对象的 MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_SYMBOLIC_LINK 属性，如以下代码所示。

```
WCHAR      *g_pwszSymbolicLink = NULL;
UINT32     g_cchSymbolicLink = 0;

HRESULT GetSymbolicLink(IMFActivate *pActivate)
{
    return pActivate->GetAllocatedString(
        MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_SYMBOLIC_LINK,
        &g_pwszSymbolicLink,
        &g_cchSymbolicLink
        );
}
```

Handle WM_DEVICECHANGE 处理 WM_DEVICECHANGE

在消息循环中，侦听 WM_DEVICECHANGE 消息。lParam消息参数是一个指向DEV_BROADCAST_HDR结构的指针。

```
case WM_DEVICECHANGE:
        if (lParam != 0)
        {
            HRESULT hr = S_OK;
            BOOL bDeviceLost = FALSE;

            hr = CheckDeviceLost((PDEV_BROADCAST_HDR)lParam, &bDeviceLost);

            if (FAILED(hr) || bDeviceLost)
            {
                CloseDevice();

                MessageBox(hwnd, L"Lost the capture device.", NULL, MB_OK);
            }
        }
        return TRUE;
```

接下来，将设备通知消息与您设备的符号链接进行比较，如下所示：

- 检查 DEV_BROADCAST_HDR 结构的 dbch_devicetype 成员。如果值为 DBT_DEVTYP_DEVICEINTERFACE，则将结构指针转换为 DEV_BROADCAST_DEVICEINTERFACE 结构。
- 将此结构的 dbcc_name 成员与设备的符号链接进行比较。

```
HRESULT CheckDeviceLost(DEV_BROADCAST_HDR *pHdr, BOOL *pbDeviceLost)
{
    DEV_BROADCAST_DEVICEINTERFACE *pDi = NULL;

    if (pbDeviceLost == NULL)
    {
        return E_POINTER;
    }

```

```
    *pbDeviceLost = FALSE;

    if (g_pSource == NULL)
    {
        return S_OK;
    }
    if (pHdr == NULL)
    {
        return S_OK;
    }
    if (pHdr->dbch_devicetype != DBT_DEVTYP_DEVICEINTERFACE)
    {
        return S_OK;
    }

    // Compare the device name with the symbolic link.

    pDi = (DEV_BROADCAST_DEVICEINTERFACE*)pHdr;

    if (g_pwszSymbolicLink)
    {
        if (_wcsicmp(g_pwszSymbolicLink, pDi->dbcc_name) == 0)
        {
            *pbDeviceLost = TRUE;
        }
    }

    return S_OK;
}
```

Unregister For Notification取消注册通知

在应用程序退出之前，调用 UnregisterDeviceNotification 取消注册设备通知/

```
void OnClose(HWND /*hwnd*/)
{
    if (g_hdevnotify)
    {
```

```
        UnregisterDeviceNotification(g_hdevnotify);
    }

    PostQuitMessage(0);
}
```

How to Set the Video Capture Format如何设置视频捕获格式

视频捕获设备可能支持多种捕获格式。格式通常因压缩类型、色彩空间（YUV 或 RGB）、帧大小或帧速率而异。

支持的格式列表包含在表示描述符中。有关详细信息，请参阅呈现描述符。

枚举支持的格式：

1. 为捕获设备创建媒体源。请参阅枚举视频捕获设备。
2. 在媒体源上调用 IMFMediaSource::CreatePresentationDescriptor 以获取演示描述符。
3. 调用 IMFPresentationDescriptor::GetStreamDescriptorByIndex 获取视频流的流描述符
4. 对流描述符调用 IMFStreamDescriptor::GetMediaTypeHandler。
5. 调用 IMFMediaTypeHandler::GetMediaTypeCount 获取支持的格式数量。
6. 在循环中，调用 IMFMediaTypeHandler::GetMediaTypeByIndex 来获取每种格式。格式由媒体类型表示。有关详细信息，请参阅视频媒体类型。

以下示例枚举设备的捕获格式：

```
HRESULT EnumerateCaptureFormats(IMFMediaSource *pSource)
{
    IMFPresentationDescriptor *pPD = NULL;
    IMFStreamDescriptor *pSD = NULL;
    IMFMediaTypeHandler *pHandler = NULL;
    IMFMediaType *pType = NULL;

    HRESULT hr = pSource->CreatePresentationDescriptor(&pPD);
    if (FAILED(hr))
    {
        goto done;
    }

    BOOL fSelected;
    hr = pPD->GetStreamDescriptorByIndex(0, &fSelected, &pSD);
    if (FAILED(hr))
```

```
    {
        goto done;
    }

    hr = pSD->GetMediaTypeHandler(&pHandler);
    if (FAILED(hr))
    {
        goto done;
    }

    DWORD cTypes = 0;
    hr = pHandler->GetMediaTypeCount(&cTypes);
    if (FAILED(hr))
    {
        goto done;
    }

    for (DWORD i = 0; i < cTypes; i++)
    {
        hr = pHandler->GetMediaTypeByIndex(i, &pType);
        if (FAILED(hr))
        {
            goto done;
        }

        LogMediaType(pType);
        OutputDebugString(L"\n");

        SafeRelease(&pType);
    }

done:
    SafeRelease(&pPD);
    SafeRelease(&pSD);
    SafeRelease(&pHandler);
    SafeRelease(&pType);
    return hr;
}
```

本示例中使用的 LogMediaType 函数在主题媒体类型调试代码中列出。

设置捕获格式：

1. 获取指向 IMFMediaTypeHandler 接口的指针，如上例所示。
2. 调用 IMFMediaTypeHandler::GetMediaTypeByIndex 以获取由索引指定的所需格式。
3. 调用 IMFMediaTypeHandler::SetCurrentMediaType 设置格式。

如果您没有设置捕获格式，设备将使用其默认格式。

以下示例设置捕获格式：

```
HRESULT SetDeviceFormat(IMFMediaSource *pSource, DWORD dwFormatIndex)
{
    IMFPresentationDescriptor *pPD = NULL;
    IMFStreamDescriptor *pSD = NULL;
    IMFMediaTypeHandler *pHandler = NULL;
    IMFMediaType *pType = NULL;

    HRESULT hr = pSource->CreatePresentationDescriptor(&pPD);
    if (FAILED(hr))
    {
        goto done;
    }

    BOOL fSelected;
    hr = pPD->GetStreamDescriptorByIndex(0, &fSelected, &pSD);
    if (FAILED(hr))
    {
        goto done;
    }

    hr = pSD->GetMediaTypeHandler(&pHandler);
    if (FAILED(hr))
    {
        goto done;
    }

    hr = pHandler->GetMediaTypeByIndex(dwFormatIndex, &pType);
    if (FAILED(hr))
    {
        goto done;
```

```
    }

    hr = pHandler->SetCurrentMediaType(pType);

done:
    SafeRelease(&pPD);
    SafeRelease(&pSD);
    SafeRelease(&pHandler);
    SafeRelease(&pType);
    return hr;
}
```

返回格式的顺序取决于设备。通常，它们首先按压缩类型或色彩空间进行分组；然后从每组内的最小帧尺寸到最大帧尺寸。

帧速率的处理方式与其他格式属性略有不同。有关详细信息，请参阅如何设置视频捕获帧速率。

注意：*在某些设备中，格式列表将包含每种格式的重复条目。例如，如果设备支持 15 种不同的捕获格式，则列表将包含 30 个条目。在每一对中，其中一种媒体类型的属性 MF_MT_AM_FORMAT_TYPE 等于 FORMAT_VideoInfo，另一种媒体类型的 MF_MT_AM_FORMAT_TYPE 等于 FORMAT_VideoInfo2。（这两个值在头文件 uuids.h 中定义。）第二种类型还可能包含附加颜色信息（扩展颜色信息）或显示不同的隔行扫描值 (MF_MT_INTERLACE_MODE)。这些重复类型的存在是为了支持较旧的 DirectShow 应用程序。在媒体基础应用程序中，每当列出重复的 FORMAT_VideoInfo2 类型时，您都应该忽略 FORMAT_VideoInfo 类型。*

How to Set the Video Capture Frame Rate如何设置视频捕获帧速率

视频捕获设备可能支持一定范围的帧速率。如果此信息可用，则最小和最大帧速率将存储为媒体类型属性：MF_MT_FRAME_RATE_RANGE_MAX，MF_MT_FRAME_RATE_RANGE_MIN

该范围可能因捕获格式而异。例如，在较大的帧尺寸下，最大帧速率可能会降低。

设置帧速率：

1. 为捕获设备创建媒体源。请参阅枚举视频捕获设备。
2. 在媒体源上调用 IMFMediaSource::CreatePresentationDescriptor 以获取演示描述符。
3. 调用 IMFPresentationDescriptor::GetStreamDescriptorByIndex 获取视频流的流描述符。
4. 对流描述符调用 IMFStreamDescriptor::GetMediaTypeHandler。
5. 枚举捕获格式，如如何设置视频捕获格式中所述。
6. 从列表中选择所需的输出格式。
7. 查询媒体类型的 MF_MT_FRAME_RATE_RANGE_MAX 和 MF_MT_FRAME_RATE_RANGE_MIN 属性。该值给出了支持的帧速率的范围。该设备可能支持此范围内的其他帧速率。
8. 设置媒体类型的 MF_MT_FRAME 属性以指定所需的帧速率。

9. 使用修改后的媒体类型调用 IMFMediaTypeHandler::SetCurrentMediaType。

以下示例将帧速率设置为设备支持的最大帧速率：

```
HRESULT SetMaxFrameRate(IMFMediaSource *pSource, DWORD dwTypeIndex)
{
    IMFPresentationDescriptor *pPD = NULL;
    IMFStreamDescriptor *pSD = NULL;
    IMFMediaTypeHandler *pHandler = NULL;
    IMFMediaType *pType = NULL;

    HRESULT hr = pSource->CreatePresentationDescriptor(&pPD);
    if (FAILED(hr))
    {
        goto done;
    }

    BOOL fSelected;
    hr = pPD->GetStreamDescriptorByIndex(dwTypeIndex, &fSelected, &pSD);
    if (FAILED(hr))
    {
        goto done;
    }

    hr = pSD->GetMediaTypeHandler(&pHandler);
    if (FAILED(hr))
    {
        goto done;
    }

    hr = pHandler->GetCurrentMediaType(&pType);
    if (FAILED(hr))
    {
        goto done;
    }

    // Get the maximum frame rate for the selected capture format.

    // Note: To get the minimum frame rate, use the
    // MF_MT_FRAME_RATE_RANGE_MIN attribute instead.
```

```
37
38      PROPVARIANT var;
39      if (SUCCEEDED(pType->GetItem(MF_MT_FRAME_RATE_RANGE_MAX, &var)))
40      {
41          hr = pType->SetItem(MF_MT_FRAME_RATE, var);
42
43          PropVariantClear(&var);
44
45          if (FAILED(hr))
46          {
47              goto done;
48          }
49
50          hr = pHandler->SetCurrentMediaType(pType);
51      }
52
53  done:
54      SafeRelease(&pPD);
55      SafeRelease(&pSD);
56      SafeRelease(&pHandler);
57      SafeRelease(&pType);
58      return hr;
59  }
```

Audio/Video Playback音频/视频播放

How to Play Media Files with Media Foundation如何使用 Media Foundation 播放媒体文件

详情参考这里。完整代码参考这里。

DirectX Video Acceleration 2.0DirectX 视频加速 2.0

DirectX 视频加速 (DXVA) 是一种 API 和相应的 DDI，用于使用硬件加速来加速视频编解码器处理。软件编解码器和软件视频处理器可以使用 DXVA 将某些 CPU 密集型操作卸载到 GPU。例如，软件解码器可以将离散余弦逆变换 (iDCT) 卸载到 GPU。

DirectX 视频加速 (DXVA) 是一个 API 和相应的 DDI，用于使用硬件加速来加速视频处理。软件编解码器和软件视频处理器可以使用 DXVA 将某些 CPU 密集型操作卸载到 GPU。例如，软件解码器可以将离散余弦逆变换 (iDCT) 卸载到 GPU。
在DXVA中，一些解码操作是由图形硬件驱动程序实现的。这组功能称为加速器。其他解码操作由用户模式应用软件实现，称为主机解码器或软件解码器。（主机解码器和软件解码器这两个术语是等效的。）加速器执行的处理称为脱离主机处理。通常，加速器使用 GPU 来加速某些操作。每当加速器执行解码操作时，主机解码器必须向加速器缓冲区传送包含执行操作所需的信息
DXVA 2 API 需要 Windows Vista 或更高版本。为了向后兼容，Windows Vista 中仍支持 DXVA 1 API。提供了一个模拟层，可以在任一版本的 API 和相反版本的 DDI 之间进行转换：

- 如果图形驱动程序符合 Windows 显示驱动程序模型 (WDDM)，则 DXVA 1 API 调用将转换为 DXVA 2 DDI 调用。
- 如果图形驱动程序使用旧版 Windows XP 显示驱动程序模型 (XPDM)，则 DXVA 2 API 调用将转换为 DXVA 1 DDI 调用。

详细信息参考这里。

Encoding and File Authoring编码和文件创作

详情参考这里。

Media Metadata媒体元数据

媒体文件包含描述文件内容的属性。在 Microsoft Media Foundation 中，这些属性可分类如下：

- 媒体类型属性指定编码参数，例如编码算法（媒体子类型）、视频帧大小、视频帧率、音频比特率和音频采样率。有关媒体类型属性的更多信息，请参阅媒体类型。
- 元数据包含媒体内容的描述性信息，例如标题、艺术家、作曲家和流派。元数据还可以描述编码参数。通过元数据访问此信息比通过媒体类型属性访问此信息更快。
- DRM 属性包含有关使用限制的信息。目前，Media Foundation 不通过元数据支持 DRM 属性（PKEY_DRM_IsProtected 属性除外）。

详情参考这里。

Networking in Media Foundation MF中的网络

详情参考这里。

Output Protection Manager输出保护管理器

输出保护管理器 (OPM) 使应用程序能够在视频内容通过物理连接器传输到显示设备时对其进行保护。
详情参考这里

Windows Media CodecsWindows 媒体编解码器

Windows Media 音频和视频编解码器是可用于压缩和解压缩数字媒体数据的对象的集合。每个编解码器由两个对象组成：编码器和解码器。文档的这一部分介绍如何使用 Windows Media 音频和视频编解码器的功能来生成和使用压缩数据流。

术语编解码器是术语压缩器和解压缩器的合并。编解码器通常实现为一对 COM 对象：一个用于编码内容，另一个用于解码内容。在某些情况下，COM 对象占用相同的动态链接库 (DLL)。

详情参考这里。