Contents

Core Audio APIs

1/18/2020 • 31 minutes to read • Edit Online

Overview of the Core Audio APIs technology.

To develop Core Audio APIs, you need these headers:

- audioclient.h
- audioendpoints.h
- audioenginebaseapo.h
- audioengineendpoint.h
- audiopolicy.h
- audiosessiontypes.h
- devicetopology.h
- endpointvolume.h
- mmdeviceapi.h
- spatialaudioclient.h
- spatialaudiohrtf.h
- spatialaudiometadata.h

For programming guidance for this technology, see:

- Core Audio APIs

Enumerations

TITLE	DESCRIPTION
_AUDCLNT_BUFFERFLAGS	The _AUDCLNT_BUFFERFLAGS enumeration defines flags that indicate the status of an audio endpoint buffer.
AUDCLNT_SHAREMODE	The AUDCLNT_SHAREMODE enumeration defines constants that indicate whether an audio stream will run in shared mode or in exclusive mode.
AUDCLNT_STREAMOPTIONS	Defines values that describe the characteristics of an audio stream.
AUDIO_STREAM_CATEGORY	Specifies the category of an audio stream.
AudioObjectType	Specifies the type of an ISpatialAudioObject.
AudioSessionState	The AudioSessionState enumeration defines constants that indicate the current state of an audio session.
ConnectorType	The ConnectorType enumeration indicates the type of connection that a connector is part of.
DataFlow	The DataFlow enumeration indicates the data-flow direction of an audio stream through a connector.

TITLE	DESCRIPTION
EDataFlow	The EDataFlow enumeration defines constants that indicate the direction in which audio data flows between an audio endpoint device and an application.
EndpointFormFactor	The EndpointFormFactor enumeration defines constants that indicate the general physical attributes of an audio endpoint device.
ERole	The ERole enumeration defines constants that indicate the role that the system has assigned to an audio endpoint device.
KSJACK_SINK_CONNECTIONTYPE	The KSJACK_SINK_CONNECTIONTYPE enumeration defines constants that specify the type of connection. These values are used in the KSJACK_SINK_INFORMATION structure that stores information about an audio jack sink.
PartType	The PartType enumeration defines constants that indicate whether a part in a device topology is a connector or subunit.
SpatialAudioHrtfDirectivityType	Specifies the shape in which sound is emitted by an ISpatialAudioObjectForHrtf.
SpatialAudioHrtfDistanceDecayType	Specifies the type of decay applied over distance from the position of an ISpatialAudioObjectForHrtf to the position of the listener.
SpatialAudioHrtfEnvironmentType	Specifies the type of acoustic environment that is simulated when audio is processed for an ISpatialAudioObjectForHrtf.
SpatialAudioMetadataCopyMode	Specifies the copy mode used when calling ISpatialAudioMetadataCopier::CopyMetadataForFrames.
SpatialAudioMetadataWriterOverflowMode	Specifies the desired behavior when an ISpatialAudioMetadataWriter attempts to write more items into the metadata buffer than was specified when the client was initialized.

Functions

TITLE	DESCRIPTION
Activate	The Activate method creates a COM object with the specified interface.
Activate	The Activate method activates a function-specific interface on a connector or subunit.
ActivateAudioInterfaceAsync	Enables Windows Store apps to access preexisting Component Object Model (COM) interfaces in the WASAPI family.
ActivateCompleted	Indicates that activation of a WASAPI interface is complete and results are available.

TITLE	DESCRIPTION
ActivateSpatialAudioMetadataCopier	Creates an ISpatialAudioMetadataWriter object for copying spatial audio metadata items from one ISpatialAudioMetadataItems object to another.
ActivateSpatialAudioMetadataItems	Creates an ISpatialAudioMetadataItems object for storing spatial audio metadata items.
ActivateSpatialAudioMetadataReader	Creates an ISpatialAudioMetadataWriter object for reading spatial audio metadata items from an ISpatialAudioMetadataItems object.
ActivateSpatialAudioMetadataWriter	Creates an ISpatialAudioMetadataWriter object for writing spatial audio metadata items to an ISpatialAudioMetadataItems object.
ActivateSpatialAudioObject	Activates an ISpatialAudioObject for audio rendering.
ActivateSpatialAudioObjectForHrtf	Activates an ISpatialAudioObjectForHrtf for audio rendering.
ActivateSpatialAudioObjectForMetadataCommands	Activate an ISpatialAudioObjectForMetadataCommands for rendering.
ActivateSpatialAudioObjectForMetadataItems	Activate an ISpatialAudioObjectForMetadataItems for rendering.
ActivateSpatialAudioStream	Activates and initializes spatial audio stream using one of the spatial audio stream activation structures.
AttachToBuffer	Attaches caller-provided memory for storage of ISpatialAudioMetadataItems objects.
AttachToPopulatedBuffer	Attaches a previously populated buffer for storage of ISpatialAudioMetadataItems objects. The metadata items already in the buffer are retained.
BeginUpdatingAudioObjects	Puts the system into the state where audio object data can be submitted for processing and the ISpatialAudioObject state can be modified.
Close	Completes any necessary operations on the SpatialAudioMetadataItems object and releases the object.
Close	Completes any necessary operations on the SpatialAudioMetadataItems object and releases the object.
Close	Completes any needed operations on the metadata buffer and releases the specified ISpatialAudioMetadataItems object.
ConnectTo	The ConnectTo method connects this connector to a connector in another device-topology object.

TITLE	DESCRIPTION
CopyMetadataForFrames	Copies metadata items from the source ISpatialAudioMetadataItems, provided to the Open method, object to the destination ISpatialAudioMetadataItems object, specified with the dstMetadataItems parameter.
DetachBuffer	Detaches the buffer. Memory can only be attached to a single metadata item at a time.
Disconnect	The Disconnect method disconnects this connector from another connector.
EndUpdatingAudioObjects	Notifies the system that the app has finished supplying audio data for the spatial audio objects activated with ActivateSpatialAudioObject.
EnumAudioEndpoints	The EnumAudioEndpoints method generates a collection of audio endpoint devices that meet the specified criteria.
EnumPartsIncoming	The EnumPartsIncoming method gets a list of all the incoming parts—that is, the parts that reside on data paths that are upstream from this part.
EnumPartsOutgoing	The EnumPartsOutgoing method retrieves a list of all the outgoing parts—that is, the parts that reside on data paths that are downstream from this part.
Get4BRange	The Get4BRange method gets the 4-byte range of the device-specific property value.
GetActivateResult	Gets the results of an asynchronous activation of a WASAPI interface initiated by an application calling the ActivateAudioInterfaceAsync function.
GetAllVolumes	The GetAllVolumes method retrieves the volume levels for all the channels in the audio stream.
GetAllVolumes	The GetAllVolumes method retrieves the volume levels for all the channels in the audio session.
GetAudioObjectType	Gets a value specifying the type of audio object that is represented by the ISpatialAudioObject.
GetAudioSessionControl	The GetAudioSessionControl method retrieves an audio session control.
GetAvailableDynamicObjectCount	Gets the number of dynamic spatial audio objects that are currently available.
GetAvailableOffloadConnectorCount	The GetAvailableOffloadConnectorCount method retrieves the number of avaialable endpoints that can handle offloaded streams on the hardware audio engine.
GetBuffer	Gets a buffer that is used to supply the audio data for the ISpatialAudioObject.

TITLE	DESCRIPTION
GetBuffer	Retrieves a pointer to the next available packet of data in the capture endpoint buffer.
GetBuffer	Retrieves a pointer to the next available space in the rendering endpoint buffer into which the caller can write a data packet.
GetBufferSize	The GetBufferSize method retrieves the size (maximum capacity) of the endpoint buffer.
GetBufferSizeLimits	The GetBufferSizeLimits method returns the buffer size limits of the hardware audio engine in 100-nanosecond units.
GetChannelConfig	The GetChannelConfig method gets the current channel-configuration mask from a channel-configuration control.
GetChannelCount	The GetChannelCount method gets the number of channels in the audio stream.
GetChannelCount	The GetChannelCount method gets the number of channels in the audio stream.
GetChannelCount	The GetChannelCount method retrieves the number of channels in the audio stream.
GetChannelCount	The GetChannelCount method retrieves the number of channels in the stream format for the audio session.
GetChannelCount	The GetChannelCount method gets a count of the channels in the audio stream that enters or leaves the audio endpoint device.
GetChannelsPeakValues	The GetChannelsPeakValues method gets the peak sample values for all the channels in the audio stream.
GetChannelVolume	The GetChannelVolume method retrieves the volume level for the specified channel in the audio stream.
GetChannelVolume	The GetChannelVolume method retrieves the volume level for the specified channel in the audio session.
GetChannelVolumeLevel	The GetChannelVolumeLevel method gets the volume level, in decibels, of the specified channel in the audio stream that enters or leaves the audio endpoint device.
GetChannelVolumeLevelScalar	The GetChannelVolumeLevelScalar method gets the normalized, audio-tapered volume level of the specified channel of the audio stream that enters or leaves the audio endpoint device.
GetChannelVolumes	The GetChannelVolumes method retrieves the volume levels for the various audio channels in the offloaded stream.
GetCharacteristics	The GetCharacteristics method is reserved for future use.

TITLE	DESCRIPTION
GetConnectedTo	The GetConnectedTo method gets the connector to which this connector is connected.
GetConnector	The GetConnector method gets the connector that is specified by a connector number.
GetConnectorCount	The GetConnectorCount method gets the number of connectors in the device-topology object.
GetConnectorIdConnectedTo	The GetConnectorIdConnectedTo method gets the global ID of the connector, if any, that this connector is connected to.
GetControlInterface	The GetControlInterface method gets a reference to the specified control interface, if this part supports it.
GetControlInterfaceCount	The GetControlInterfaceCount method gets the number of control interfaces that this part supports.
GetCount	Gets the number of supported audio formats in the list.
GetCount	The GetCount method retrieves a count of the devices in the device collection.
GetCount	The GetCount method gets the number of parts in the parts list.
GetCount	The GetCount method gets the total number of audio sessions that are open on the audio device.
GetCurrentPadding	The GetCurrentPadding method retrieves the number of frames of padding in the endpoint buffer.
GetCurrentSharedModeEnginePeriod	Returns the current format and periodicity of the audio engine.
GetDataFlow	The GetDataFlow method indicates whether the audio endpoint device is a rendering device or a capture device.
GetDataFlow	The GetDataFlow method gets the direction of data flow through this connector.
GetDefaultAudioEndpoint	The GetDefaultAudioEndpoint method retrieves the default audio endpoint for the specified data-flow direction and role.
GetDevice	The GetDevice method retrieves an audio endpoint device that is identified by an endpoint ID string.
GetDeviceId	The GetDeviceId method gets the device identifier of the device that is represented by the device-topology object.
GetDeviceIdConnectedTo	The GetDeviceIdConnectedTo method gets the device identifier of the audio device, if any, that this connector is connected to.

TITLE	DESCRIPTION
GetDevicePeriod	The GetDevicePeriod method retrieves the length of the periodic interval separating successive processing passes by the audio engine on the data in the endpoint buffer.
GetDevicePosition	The GetDevicePosition method gets the current device position, in frames, directly from the hardware.
GetDevicePreferredFormat	The GetDevicePreferredFormat method gets the preferred audio stream format for the connection.
GetDisplayName	The GetDisplayName method retrieves the display name for the audio session.
GetEnabled	The GetEnabled method gets the current state (enabled or disabled) of the AGC.
GetEnabled	The GetEnabled method gets the current state (enabled or disabled) of the loudness control.
GetEngineFormat	The GetEngineFormat method retrieves the current data format of the offloaded audio stream.
GetFormat	Gets the format with the specified index in the list. The formats are listed in order of importance. The most preferable format is first in the list.
GetFrameCount	Gets the total frame count of the ISpatialAudioMetadataItems, which defines valid item offsets.
GetFrequency	The GetFrequency method gets the device frequency.
GetGfxState	The GetGfxState method retrieves the GFX state of the offloaded audio stream.
GetGlobalId	The GetGlobalId method gets the global ID of this part.
GetGroupingParam	The GetGroupingParam method retrieves the grouping parameter of the audio session.
GetIconPath	The GetIconPath method retrieves the path for the display icon for the audio session.
GetId	The GetId method retrieves an endpoint ID string that identifies the audio endpoint device.
GetIID	The GetIID method gets the interface ID of the function-specific control interface of the part.
GetInfo	Gets the total frame count for the ISpatialAudioMetadataItems, which defines valid item offsets.
GetItemCount	The current number of items stored by the ISpatialAudioMetadataItems.

TITLE	DESCRIPTION
GetJackCount	The GetJackCount method gets the number of jacks required to connect to an audio endpoint device.
GetJackCount	The GetJackCount method gets the number of jacks on the connector, which are required to connect to an endpoint device.
GetJackDescription	The GetJackDescription method gets a description of an audio jack.
GetJackDescription2	The GetJackDescription2 method gets the description of a specified audio jack.
GetJackSinkInformation	The GetJackSinkInformation method retrieves the sink information for the specified jack.
GetLevel	The GetLevel method gets the peak level that the peak meter recorded for the specified channel since the peak level for that channel was previously read.
GetLevel	The GetLevel method gets the volume level, in decibels, of the specified channel.
GetLevelRange	The GetLevelRange method gets the range, in decibels, of the volume level of the specified channel.
GetLocalEffectsState	The GetLocalEffectsState method retrieves the local effects state that is currently applied to the offloaded audio stream.
GetLocalId	The GetLocalId method gets the local ID of this part.
GetMasterVolume	The GetMasterVolume method retrieves the client volume level for the audio session.
GetMasterVolumeLevel	The GetMasterVolumeLevel method gets the master volume level, in decibels, of the audio stream that enters or leaves the audio endpoint device.
GetMasterVolumeLevelScalar	The GetMasterVolumeLevelScalar method gets the master volume level of the audio stream that enters or leaves the audio endpoint device. The volume level is expressed as a normalized, audio-tapered value in the range from 0.0 to 1.0.
GetMaxDynamicObjectCount	Gets the maximum number of dynamic audio objects for the spatial audio client.
GetMaxFrameCount	Gets the maximum possible frame count per processing pass. This method can be used to determine the size of the source buffer that should be allocated to convey audio data for each processing pass.
GetMaxItemCount	The maximum number of items allowed by the ISpatialAudioMetadataItems, defined when the object is created.

TITLE	DESCRIPTION
GetMaxValueBufferLength	The size of the largest command value defined by the metadata format for the ISpatialAudioMetadataItems.
GetMeterChannelCount	Gets the number of available audio channels in the offloaded stream that can be metered.
GetMeteringChannelCount	The GetMeteringChannelCount method gets the number of channels in the audio stream that are monitored by peak meters.
GetMeteringData	The GetMeteringData method retrieves general information about the available audio channels in the offloaded stream.
GetMixFormat	The GetMixFormat method retrieves the stream format that the audio engine uses for its internal processing of shared-mode streams.
GetMute	The GetMute method retrieves the mute status of the offloaded audio stream.
GetMute	The GetMute method gets the current state (enabled or disabled) of the mute control.
GetMute	The GetMute method retrieves the current muting state for the audio session.
GetMute	The GetMute method gets the muting state of the audio stream that enters or leaves the audio endpoint device.
GetName	The GetName method gets the friendly name for the audio function that the control interface encapsulates.
GetName	The GetName method gets the friendly name of this part.
GetNativeStaticObjectTypeMask	Gets a channel mask which represents the subset of static speaker bed channels native to current rendering engine.
GetNextPacketSize	The GetNextPacketSize method retrieves the number of frames in the next data packet in the capture endpoint buffer.
GetPart	The GetPart method gets a part from the parts list.
GetPartById	The GetPartById method gets a part that is identified by its local ID.
GetPartType	The GetPartType method gets the part type of this part.
GetPeakValue	The GetPeakValue method gets the peak sample value for the channels in the audio stream.
GetPosition	The GetPosition method gets the current device position.

TITLE	DESCRIPTION
GetProcessId	The GetProcessId method retrieves the process identifier of the audio session.
GetSelection	The GetSelection method gets the local ID of the part that is connected to the selector input that is currently selected.
GetSelection	The GetSelection method gets the local ID of the part that is connected to the selector output that is currently selected.
GetService	Gets additional services from the ISpatialAudioObjectRenderStream.
GetService	The GetService method accesses additional services from the audio client object.
GetSession	The GetSession method gets the audio session specified by an audio session number.
GetSessionEnumerator	The GetSessionEnumerator method gets a pointer to the audio session enumerator object.
GetSessionIdentifier	The GetSessionIdentifier method retrieves the audio session identifier.
GetSessionInstanceIdentifier	The GetSessionInstanceIdentifier method retrieves the identifier of the audio session instance.
GetSharedModeEnginePeriod	Returns the range of periodicities supported by the engine for the specified stream format.
GetSignalPath	The GetSignalPath method gets a list of parts in the signal path that links two parts, if the path exists.
GetSimpleAudioVolume	The GetSimpleAudioVolume method retrieves a simple audio volume control.
GetSpatialAudioMetadataItems	Gets a pointer to the ISpatialAudioMetadataItems object which stores metadata items for the ISpatialAudioObjectForMetadataItems.
GetSpatialAudioMetadataItemsBufferLength	Gets the length of the buffer required to store the specified number of spatial audio metadata items.
GetState	The GetState method retrieves the current device state.
GetState	The GetState method retrieves the current state of the audio session.
GetStaticObjectPosition	Gets the position in 3D space of the specified static spatial audio channel.
GetStreamLatency	The GetStreamLatency method retrieves the maximum latency for the current stream and can be called any time after the stream has been initialized.

TITLE	DESCRIPTION
GetSubType	The GetSubType method gets the part subtype of this part.
GetSubunit	The GetSubunit method gets the subunit that is specified by a subunit number.
GetSubunitCount	The GetSubunitCount method gets the number of subunits in the device topology.
GetSupportedAudioObjectFormatEnumerator	Gets an IAudioFormatEnumerator that contains all supported audio formats for spatial audio objects, the first item in the list represents the most preferable format.
GetTopologyObject	The GetTopologyObject method gets a reference to the IDeviceTopology interface of the device-topology object that contains this part.
GetType	The GetType method gets the type of this connector.
GetType	The GetType method gets the data type of the device-specific property value.
GetValue	The GetValue method gets the current value of the device-specific property.
GetVolumeChannelCount	The GetVolumeChannelCount method retrieves the number of available audio channels in the offloaded stream.
GetVolumeRange	The GetVolumeRange method gets the volume range, in decibels, of the audio stream that enters or leaves the audio endpoint device.
GetVolumeRangeChannel	The GetVolumeRangeChannel method gets the volume range for a specified channel.
GetVolumeStepInfo	The GetVolumeStepInfo method gets information about the current step in the volume range.
Initialize	The Initialize method initializes the audio stream.
InitializeSharedAudioStream	Initializes a shared stream with the specified periodicity.
IsActive	Gets a boolean value indicating whether the ISpatialAudioObject is valid.
IsAudioObjectFormatSupported	Gets a value indicating whether ISpatialAudioObjectRenderStream supports a the specified format.
IsConnected	The IsConnected method indicates whether this connector is connected to another connector.
IsFormatSupported	The IsFormatSupported method indicates whether the audio endpoint device supports the specified audio stream format.

TITLE	DESCRIPTION
IsFormatSupported	The IsFormatSupported method indicates whether the audio endpoint device supports a particular stream format.
IsLastBufferControlSupported	Indicates if last buffer control is supported.
IsOffloadCapable	The IsOffloadCapable method retrieves information about whether or not the endpoint on which a stream is created is capable of supporting an offloaded audio stream.
IsSpatialAudioStreamAvailable	When successful, gets a value indicating whether the currently active spatial rendering engine supports the specified spatial audio render stream.
IsSystemSoundsSession	The IsSystemSoundsSession method indicates whether the session is a system sounds session.
Item	The Item method retrieves a pointer to the specified item in the device collection.
OnAvailableDynamicObjectCountChange	Notifies the spatial audio client when the rendering capacity for an ISpatialAudioObjectRenderStream is about to change, specifies the time after which the change will occur, and specifies the number of dynamic audio objects that will be available after the change.
OnChannelVolumeChanged	The OnChannelVolumeChanged method notifies the client that the volume level of an audio channel in the session submix has changed.
OnDefaultDeviceChanged	The OnDefaultDeviceChanged method notifies the client that the default audio endpoint device for a particular device role has changed.
OnDeviceAdded	The OnDeviceAdded method indicates that a new audio endpoint device has been added.
OnDeviceRemoved	The OnDeviceRemoved method indicates that an audio endpoint device has been removed.
OnDeviceStateChanged	The OnDeviceStateChanged method indicates that the state of an audio endpoint device has changed.
OnDisplayNameChanged	The OnDisplayNameChanged method notifies the client that the display name for the session has changed.
OnGroupingParamChanged	The OnGroupingParamChanged method notifies the client that the grouping parameter for the session has changed.
OnIconPathChanged	The OnIconPathChanged method notifies the client that the display icon for the session has changed.
OnNotify	The OnNotify method notifies the client when the status of a connector or subunit changes.

TITLE	DESCRIPTION
OnNotify	The OnNotify method notifies the client that the volume level or muting state of the audio endpoint device has changed.
OnPropertyValueChanged	The OnPropertyValueChanged method indicates that the value of a property belonging to an audio endpoint device has changed.
OnSessionCreated	The OnSessionCreated method notifies the registered processes that the audio session has been created.
OnSessionDisconnected	The OnSessionDisconnected method notifies the client that the audio session has been disconnected.
OnSimpleVolumeChanged	The OnSimpleVolumeChanged method notifies the client that the volume level or muting state of the audio session has changed.
OnStateChanged	The OnStateChanged method notifies the client that the stream-activity state of the session has changed.
OnVolumeDuckNotification	The OnVolumeDuckNotification method sends a notification about a pending system ducking event.
OnVolumeUnduckNotification	The OnVolumeUnduckNotification method sends a notification about a pending system unducking event.
Open	Opens an ISpatialAudioMetadataItems object for copying.
Open	Opens an ISpatialAudioMetadataItems object for reading.
Open	Opens an ISpatialAudioMetadataItems object for writing.
OpenPropertyStore	The OpenPropertyStore method retrieves an interface to the device's property store.
QueryHardwareSupport	The QueryHardwareSupport method queries the audio endpoint device for its hardware-supported functions.
QueryHardwareSupport	The QueryHardwareSupport method queries the audio endpoint device for its hardware-supported functions.
ReadNextItem	Gets the number of commands and the sample offset for the metadata item being read.
ReadNextItemCommand	Reads metadata commands and value data for the current item.
RegisterAudioSessionNotification	The RegisterAudioSessionNotification method registers the client to receive notifications of session events, including changes in the stream state.
RegisterControlChangeCallback	The RegisterControlChangeCallback method registers the IControlChangeNotify interface, which the client implements to receive notifications of status changes in this part.

TITLE	DESCRIPTION
RegisterControlChangeNotify	The RegisterControlChangeNotify method registers a client's notification callback interface.
RegisterDuckNotification	The RegisterDuckNotification method registers the application with the session manager to receive ducking notifications.
RegisterEndpointNotificationCallback	The RegisterEndpointNotificationCallback method registers a client's notification callback interface.
RegisterSessionNotification	The RegisterSessionNotification method registers the application to receive a notification when a session is created.
ReleaseBuffer	The ReleaseBuffer method releases the buffer.
ReleaseBuffer	The ReleaseBuffer method releases the buffer space acquired in the previous call to the IAudioRenderClient::GetBuffer method.
ReleaseOutputDataPointerForLastBuffer	Releases the output data pointer for the last buffer.
Reset	Reset a stopped audio stream.
Reset	The Reset method resets the audio stream.
ResetToDefault	Resets the format to the default setting provided by the device manufacturer.
SetAllVolumes	The SetAllVolumes method sets the individual volume levels for all the channels in the audio stream.
SetAllVolumes	The SetAllVolumes method sets the individual volume levels for all the channels in the audio session.
SetChannelConfig	The SetChannelConfig method sets the channel-configuration mask in a channel-configuration control.
SetChannelVolume	The SetChannelVolume method sets the volume level for the specified channel in the audio stream.
SetChannelVolume	The SetChannelVolume method sets the volume level for the specified channel in the audio session.
SetChannelVolumeLevel	The SetChannelVolumeLevel method sets the volume level, in decibels, of the specified channel of the audio stream that enters or leaves the audio endpoint device.
SetChannelVolumeLevelScalar	The SetChannelVolumeLevelScalar method sets the normalized, audio-tapered volume level of the specified channel in the audio stream that enters or leaves the audio endpoint device.
SetChannelVolumes	The SetChannelVolumes method sets the volume levels for the various audio channels in the offloaded stream.

TITLE	DESCRIPTION
SetClientProperties	Sets the properties of the audio stream by populating an AudioClientProperties structure.
SetDirectivity	Sets the spatial audio directivity model for the ISpatialAudioObjectForHrtf.
SetDisplayName	The SetDisplayName method assigns a display name to the current session.
SetDistanceDecay	Sets the decay model that is applied over distance from the position of an ISpatialAudioObjectForHrtf to the position of the listener.
SetDuckingPreference	The SetDuckingPreference method enables or disables the default stream attenuation experience (auto-ducking) provided by the system.
SetEnabled	The SetEnabled method enables or disables the AGC.
SetEnabled	The SetEnabled method enables or disables the loudness control.
SetEndOfStream	Instructs the system that the final block of audio data has been submitted for the ISpatialAudioObject so that the object can be deactivated and it's resources reused.
SetEngineDeviceFormat	The SetEngineDeviceFormat method sets the waveform audio format for the hardware audio engine.
SetEnvironment	Sets the type of acoustic environment that is simulated when audio is processed for the ISpatialAudioObjectForHrtf.
SetEventHandle	The SetEventHandle method sets the event handle that the system signals when an audio buffer is ready to be processed by the client.
SetGain	Sets the gain for the ISpatialAudioObjectForHrtf.
SetGfxState	The SetGfxState method sets the GFX state of the offloaded audio stream.
SetGroupingParam	The SetGroupingParam method assigns a session to a grouping of sessions.
SetIconPath	The SetIconPath method assigns a display icon to the current session.
SetLevel	The SetLevel method sets the volume level, in decibels, of the specified channel.
SetLevelAllChannels	The SetLevelAllChannels method sets the volume levels, in decibels, of all the channels in the audio stream.

TITLE	DESCRIPTION
SetLevelUniform	The SetLevelUniform method sets all channels in the audio stream to the same uniform volume level, in decibels.
SetLocalEffectsState	The SetLocalEffectsState method sets the local effects state that is to be applied to the offloaded audio stream.
SetMasterVolume	The SetMasterVolume method sets the master volume level for the audio session.
SetMasterVolumeLevel	The SetMasterVolumeLevel method sets the master volume level, in decibels, of the audio stream that enters or leaves the audio endpoint device.
SetMasterVolumeLevelScalar	The SetMasterVolumeLevelScalar method sets the master volume level of the audio stream that enters or leaves the audio endpoint device. The volume level is expressed as a normalized, audio-tapered value in the range from 0.0 to 1.0.
SetMute	The SetMute method sets the mute status of the offloaded audio stream.
SetMute	The SetMute method enables or disables the mute control.
SetMute	The SetMute method sets the muting state for the audio session.
SetMute	The SetMute method sets the muting state of the audio stream that enters or leaves the audio endpoint device.
SetOrientation	Sets the orientation in 3D space, relative to the listener's frame of reference, from which the ISpatialAudioObjectForHrtf audio data will be rendered.
SetPosition	Sets the position in 3D space, relative to the listener, from which the ISpatialAudioObjectForHrtf audio data will be rendered.
SetPosition	Sets the position in 3D space, relative to the listener, from which the ISpatialAudioObject audio data will be rendered.
SetSampleRate	The SetSampleRate method sets the sample rate of a stream.
SetSelection	The SetSelection method selects one of the inputs of the input selector.
SetSelection	The SetSelection method selects one of the outputs of the output selector.
SetValue	The SetValue method sets the value of the device-specific property.
SetVolume	Sets an audio amplitude multiplier that will be applied to the audio data provided by the ISpatialAudioObject before it is submitted to the audio rendering engine.

TITLE	DESCRIPTION
Start	Starts the spatial audio stream.
Start	The Start method starts the audio stream.
Stop	Stops a running audio stream.
Stop	The Stop method stops the audio stream.
UnregisterAudioSessionNotification	The UnregisterAudioSessionNotification method deletes a previous registration by the client to receive notifications.
UnregisterControlChangeCallback	The UnregisterControlChangeCallback method removes the registration of an IControlChangeNotify interface that the client previously registered by a call to the IPart::RegisterControlChangeCallback method.
UnregisterControlChangeNotify	The UnregisterControlChangeNotify method deletes the registration of a client's notification callback interface that the client registered in a previous call to the IAudioEndpointVolume::RegisterControlChangeNotify method.
UnregisterDuckNotification	The UnregisterDuckNotification method deletes a previous registration by the application to receive notifications.
UnregisterEndpointNotificationCallback	The UnregisterEndpointNotificationCallback method deletes the registration of a notification interface that the client registered in a previous call to the IMMDeviceEnumerator::RegisterEndpointNotificationCallback method.
UnregisterSessionNotification	The UnregisterSessionNotification method deletes the registration to receive a notification when a session is created.
VolumeStepDown	The VolumeStepDown method decrements, by one step, the volume level of the audio stream that enters or leaves the audio endpoint device.
VolumeStepUp	The VolumeStepUp method increments, by one step, the volume level of the audio stream that enters or leaves the audio endpoint device.
WriteNextItem	Starts a new metadata item at the specified offset.
WriteNextItemCommand	Writes metadata commands and value data to the current item.
WriteNextMetadataCommand	Writes a metadata command to the spatial audio object, each command may only be added once per object per processing cycle.

Interfaces

TITLE	DESCRIPTION
IActivateAudioInterfaceAsyncOperation	Represents an asynchronous operation activating a WASAPI interface and provides a method to retrieve the results of the activation.
IActivateAudioInterfaceCompletionHandler	Provides a callback to indicate that activation of a WASAPI interface is complete.
IAudioAutoGainControl	The IAudioAutoGainControl interface provides access to a hardware automatic gain control (AGC).
IAudioBass	The IAudioBass interface provides access to a hardware bass-level control.
IAudioCaptureClient	The IAudioCaptureClient interface enables a client to read input data from a capture endpoint buffer.
IAudioChannelConfig	The IAudioChannelConfig interface provides access to a hardware channel-configuration control.
IAudioClient	The IAudioClient interface enables a client to create and initialize an audio stream between an audio application and the audio engine (for a shared-mode stream) or the hardware buffer of an audio endpoint device (for an exclusive-mode stream).
IAudioClient2	The IAudioClient2 interface is derived from the IAudioClient interface, with a set of additional methods that enable a Windows Audio Session API (WASAPI) audio client to do the following:_opt in for offloading, query stream properties, and get information from the hardware that handles offloading.The audio client can be successful in creating an offloaded stream if the underlying endpoint supports the hardware audio engine, the endpoint has been enumerated and discovered by the audio system, and there are still offload pin instances available on the endpoint.
IAudioClient3	The IAudioClient3 interface is derived from the IAudioClient2 interface, with a set of additional methods that enable a Windows Audio Session API (WASAPI) audio client to query for the audio engine's supported periodicities and current periodicity as well as request initialization a shared audio stream with a specified periodicity.
IAudioClock	The IAudioClock interface enables a client to monitor a stream's data rate and the current position in the stream.
IAudioClock2	The IAudioClock2 interface is used to get the current device position.
IAudioClockAdjustment	The IAudioClockAdjustment interface is used to adjust the sample rate of a stream.
IAudioEndpointFormatControl	Used for resetting the current audio endpoint device format.

TITLE	DESCRIPTION
IAudioEndpointLastBufferControl	Provides functionality to allow an offload stream client to notify the endpoint that the last buffer has been sent only partially filled.
IAudioEndpointOffloadStreamMeter	The IAudioEndpointOffloadStreamMeter interface retrieves general information about the audio channels in the offloaded audio stream.
IAudioEndpointOffloadStreamMute	The IAudioEndpointOffloadStreamMute interface allows a client to manipulate the mute status of the offloaded audio stream.
IAudioEndpointOffloadStreamVolume	The IAudioEndpointOffloadStreamVolume interface allows the client application to manipulate the volume level of the offloaded audio stream.
IAudioEndpointVolume	The IAudioEndpointVolume interface represents the volume controls on the audio stream to or from an audio endpoint device.
IAudioEndpointVolumeCallback	The IAudioEndpointVolumeCallback interface provides notifications of changes in the volume level and muting state of an audio endpoint device.
IAudioEndpointVolumeEx	The IAudioEndpointVolumeEx interface provides volume controls on the audio stream to or from a device endpoint.
IAudioFormatEnumerator	Provides a list of supported audio formats. The most preferred format is first in the list. Get a reference to this interface by calling ISpatialAudioClient::GetSupportedAudioObjectFormatEnumerator.
IAudioInputSelector	The IAudioInputSelector interface provides access to a hardware multiplexer control (input selector).
IAudioLfxControl	The IAudioLfxControl interface allows the client to apply or remove local effects from the offloaded audio stream.
IAudioLoudness	The IAudioLoudness interface provides access to a "loudness" compensation control.
IAudioMeterInformation	The IAudioMeterInformation interface represents a peak meter on an audio stream to or from an audio endpoint device.
IAudioMidrange	The IAudioMidrange interface provides access to a hardware midrange-level control.
IAudioMute	The IAudioMute interface provides access to a hardware mute control.
IAudioOutputSelector	The IAudioOutputSelector interface provides access to a hardware demultiplexer control (output selector).

TITLE	DESCRIPTION
IAudioPeakMeter	The IAudioPeakMeter interface provides access to a hardware peak-meter control.
IAudioRenderClient	The IAudioRenderClient interface enables a client to write output data to a rendering endpoint buffer.
IAudioSessionControl	The IAudioSessionControl interface enables a client to configure the control parameters for an audio session and to monitor events in the session.
IAudioSessionControl2	The IAudioSessionControl2 interface can be used by a client to get information about the audio session.
IAudioSessionEnumerator	The IAudioSessionEnumerator interface enumerates audio sessions on an audio device.
IAudioSessionEvents	The IAudioSessionEvents interface provides notifications of session-related events such as changes in the volume level, display name, and session state.
IAudioSessionManager	The IAudioSessionManager interface enables a client to access the session controls and volume controls for both cross-process and process-specific audio sessions.
IAudioSessionManager2	The IAudioSessionManager2 interface enables an application to manage submixes for the audio device.
IAudioSessionNotification	The IAudioSessionNotification interface provides notification when an audio session is created.
IAudioStreamVolume	The IAudioStreamVolume interface enables a client to control and monitor the volume levels for all of the channels in an audio stream.
IAudioTreble	The IAudioTreble interface provides access to a hardware treble-level control.
IAudioVolumeDuckNotification	The IAudioVolumeDuckNotification interface is used to by the system to send notifications about stream attenuation changes.Stream Attenuation, or ducking, is a feature introduced in Windows 7, where the system adjusts the volume of a non-communication stream when a new communication stream is opened. For more information about this feature, see Default Ducking Experience.
IAudioVolumeLevel	The IAudioVolumeLevel interface provides access to a hardware volume control.
IChannelAudioVolume	The IChannelAudioVolume interface enables a client to control and monitor the volume levels for all of the channels in the audio session that the stream belongs to.
IConnector	The IConnector interface represents a point of connection between components.

TITLE	DESCRIPTION
IControlChangeNotify	The IControlChangeNotify interface provides notifications when the status of a part (connector or subunit) changes.
IControlInterface	The IControlInterface interface represents a control interface on a part (connector or subunit) in a device topology. The client obtains a reference to a part's IControlInterface interface by calling the IPart::GetControlInterface method.
IDeviceSpecificProperty	The IDeviceSpecificProperty interface provides access to the control value of a device-specific hardware control.
IDeviceTopology	The IDeviceTopology interface provides access to the topology of an audio device.
IHardwareAudioEngineBase	The IHardwareAudioEngineBase interface is implemented by audio endpoints for the audio stack to use to configure and retrieve information about the hardware audio engine.
IKsFormatSupport	The IKsFormatSupport interface provides information about the audio data formats that are supported by a software-configured I/O connection (typically a DMA channel) between an audio adapter device and system memory.
IKsJackDescription	The IKsJackDescription interface provides information about the jacks or internal connectors that provide a physical connection between a device on an audio adapter and an external or internal endpoint device (for example, a microphone or CD player).
IKsJackDescription2	The IKsJackDescription2 interface provides information about the jacks or internal connectors that provide a physical connection between a device on an audio adapter and an external or internal endpoint device (for example, a microphone or CD player).
IKsJackSinkInformation	The IKsJackSinkInformation interface provides access to jack sink information if the jack is supported by the hardware.
IMMDevice	The IMMDevice interface encapsulates the generic features of a multimedia device resource.
IMMDeviceCollection	The IMMDeviceCollection interface represents a collection of multimedia device resources.
IMMDeviceEnumerator	The IMMDeviceEnumerator interface provides methods for enumerating multimedia device resources.
IMMEndpoint	The IMMEndpoint interface represents an audio endpoint device.
IMMNotificationClient	The IMMNotificationClient interface provides notifications when an audio endpoint device is added or removed, when the state or properties of an endpoint device change, or when there is a change in the default role assigned to an endpoint device.

TITLE	DESCRIPTION
IPart	The IPart interface represents a part (connector or subunit) of a device topology.
IPartsList	The IPartsList interface represents a list of parts, each of which is an object with an IPart interface that represents a connector or subunit.
IPerChannelDbLevel	The IPerChannelDbLevel interface represents a generic subunit control interface that provides per-channel control over the volume level, in decibels, of an audio stream or of a frequency band in an audio stream.
ISimpleAudioVolume	The ISimpleAudioVolume interface enables a client to control the master volume level of an audio session.
ISpatialAudioClient	The ISpatialAudioClient interface enables a client to create audio streams that emit audio from a position in 3D space.
ISpatialAudioMetadataClient	Provides a class factory for creating ISpatialAudioMetadataItems, ISpatialAudioMetadataWriter, ISpatialAudioMetadataReader, and ISpatialAudioMetadataCopier objects.
ISpatialAudioMetadataCopier	Provides methods for copying all or subsets of metadata items from a source SpatialAudioMetadataItems into a destination SpatialAudioMetadataItems.
ISpatialAudioMetadataItems	Represents a buffer of spatial audio metadata items.
ISpatialAudioMetadataItemsBuffer	Provides methods for attaching buffers to SpatialAudioMetadataItems for in-place storage of data.
ISpatialAudioMetadataReader	Provides methods for extracting spatial audio metadata items and item command value pairs from an ISpatialAudioMetadataItems object.
ISpatialAudioMetadataWriter	Provides methods for storing spatial audio metadata items positioned within a range of corresponding audio frames.
ISpatialAudioObject	Represents an object that provides audio data to be rendered from a position in 3D space, relative to the user.
ISpatialAudioObjectBase	Base interface that represents an object that provides audio data to be rendered from a position in 3D space, relative to the user.
ISpatialAudioObjectForHrtf	Represents an object that provides audio data to be rendered from a position in 3D space, relative to the user, a head-relative transfer function (HRTF).
ISpatialAudioObjectForMetadataCommands	Used to write metadata commands for spatial audio.
ISpatialAudioObjectForMetadataItems	Used to write spatial audio metadata for applications that require multiple metadata items per buffer with frame-accurate placement.

TITLE	DESCRIPTION
ISpatialAudioObjectRenderStream	Provides methods for controlling a spatial audio object render stream, including starting, stopping, and resetting the stream.
ISpatialAudioObjectRenderStreamBase	Base interface that provides methods for controlling a spatial audio object render stream, including starting, stopping, and resetting the stream.
ISpatialAudioObjectRenderStreamForHrtf	Provides methods for controlling an Hrtf spatial audio object render stream, including starting, stopping, and resetting the stream.
ISpatialAudioObjectRenderStreamForMetadata	Provides methods for controlling a spatial audio object render stream for metadata, including starting, stopping, and resetting the stream.
ISpatialAudioObjectRenderStreamNotify	Provides notifications for spatial audio clients to respond to changes in the state of an ISpatialAudioObjectRenderStream.
ISubunit	The ISubunit interface represents a hardware subunit (for example, a volume control) that lies in the data path between a client and an audio endpoint device.

Structures

TITLE	DESCRIPTION
AUDIO_VOLUME_NOTIFICATION_DATA	The AUDIO_VOLUME_NOTIFICATION_DATA structure describes a change in the volume level or muting state of an audio endpoint device.
AudioExtensionParams	This structure is passed to the Control Panel Endpoint Extension property page through IShellPropSheetExt::AddPages and is used to create endpoint PropertyPages.
DIRECTX_AUDIO_ACTIVATION_PARAMS	The DIRECTX_AUDIO_ACTIVATION_PARAMS structure specifies the initialization parameters for a DirectSound stream.
KSJACK_DESCRIPTION	The KSJACK_DESCRIPTION structure describes an audio jack.
KSJACK_DESCRIPTION2	The KSJACK_DESCRIPTION2 structure describes an audio jack.To get the description of an audio jack of a connector, call IKsJackDescription2::GetJackDescription2.
KSJACK_SINK_INFORMATION	The KSJACK_SINK_INFORMATION structure stores information about an audio jack sink.

TITLE	DESCRIPTION
LUID	The LUID structure stores the video port identifier. This structure is stored in the PortId member of the KSJACK_SINK_INFORMATION structure.
SpatialAudioClientActivationParams	Represents optional activation parameters for a spatial audio render stream. Pass this structure to ActivateAudioInterfaceAsync when activating an ISpatialAudioClient interface.
SpatialAudioHrtfActivationParams	Specifies the activation parameters for an ISpatialAudioRenderStreamForHrtf.
SpatialAudioHrtfDirectivity	Represents an omnidirectional model for an ISpatialAudioObjectForHrtf. The omnidirectional emission is interpolated linearly with the directivity model specified in the Type field based on the value of the Scaling field.
SpatialAudioHrtfDirectivityCardioid	Represents a cardioid-shaped directivity model for an ISpatialAudioObjectForHrtf.
SpatialAudioHrtfDirectivityCone	Represents a cone-shaped directivity model for an ISpatialAudioObjectForHrtf.
SpatialAudioHrtfDirectivityUnion	Defines a spatial audio directivity model for an ISpatialAudioObjectForHrtf.
SpatialAudioHrtfDistanceDecay	Represents the decay model that is applied over distance from the position of an ISpatialAudioObjectForHrtf to the position of the listener.
SpatialAudioMetadataItemsInfo	Provides information about an ISpatialAudioMetadataItems object. Get a copy of this structure by calling GetInfo.
SpatialAudioObjectRenderStreamActivationParams	Represents activation parameters for a spatial audio render stream. Pass this structure to ISpatialAudioClient::ActivateSpatialAudioStream when activating a stream.
SpatialAudioObjectRenderStreamForMetadataActivationParams	Represents activation parameters for a spatial audio render stream for metadata. Pass this structure to ISpatialAudioClient::ActivateSpatialAudioStream when activating a stream.

audioclient.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs audioclient.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IAudioCaptureClient	The IAudioCaptureClient interface enables a client to read input data from a capture endpoint buffer.
IAudioClient	The IAudioClient interface enables a client to create and initialize an audio stream between an audio application and the audio engine (for a shared-mode stream) or the hardware buffer of an audio endpoint device (for an exclusive-mode stream).
IAudioClient2	The IAudioClient2 interface is derived from the IAudioClient interface, with a set of additional methods that enable a Windows Audio Session API (WASAPI) audio client to do the following:_opt in for offloading, query stream properties, and get information from the hardware that handles offloading.The audio client can be successful in creating an offloaded stream if the underlying endpoint supports the hardware audio engine, the endpoint has been enumerated and discovered by the audio system, and there are still offload pin instances available on the endpoint.
IAudioClient3	The IAudioClient3 interface is derived from the IAudioClient2 interface, with a set of additional methods that enable a Windows Audio Session API (WASAPI) audio client to query for the audio engine's supported periodicities and current periodicity as well as request initialization a shared audio stream with a specified periodicity.
IAudioClock	The IAudioClock interface enables a client to monitor a stream's data rate and the current position in the stream.
IAudioClock2	The IAudioClock2 interface is used to get the current device position.
IAudioClockAdjustment	The IAudioClockAdjustment interface is used to adjust the sample rate of a stream.
IAudioRenderClient	The IAudioRenderClient interface enables a client to write output data to a rendering endpoint buffer.
IAudioStreamVolume	The IAudioStreamVolume interface enables a client to control and monitor the volume levels for all of the channels in an audio stream.

TITLE	DESCRIPTION
IChannelAudioVolume	The IChannelAudioVolume interface enables a client to control and monitor the volume levels for all of the channels in the audio session that the stream belongs to.
ISimpleAudioVolume	The ISimpleAudioVolume interface enables a client to control the master volume level of an audio session.

Structures

TITLE	DESCRIPTION
AudioClientProperties	

Enumerations

TITLE	DESCRIPTION
_AUDCLNT_BUFFERFLAGS	The _AUDCLNT_BUFFERFLAGS enumeration defines flags that indicate the status of an audio endpoint buffer.
AUDCLNT_STREAMOPTIONS	Defines values that describe the characteristics of an audio stream.

_AUDCLNT_BUFFERFLAGS enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **_AUDCLNT_BUFFERFLAGS** enumeration defines flags that indicate the status of an audio endpoint buffer.

Syntax

```
typedef enum _AUDCLNT_BUFFERFLAGS {
  AUDCLNT_BUFFERFLAGS_DATA_DISCONTINUITY,
  AUDCLNT_BUFFERFLAGS_SILENT,
  AUDCLNT_BUFFERFLAGS_TIMESTAMP_ERROR
} ;
```

Constants

AUDCLNT_BUFFERFLAGS_DATA_DISCONTINUITY	The data in the packet is not correlated with the previous packet's device position; this is possibly due to a stream state transition or timing glitch.
AUDCLNT_BUFFERFLAGS_SILENT	Treat all of the data in the packet as silence and ignore the actual data values. For more information about the use of this flag, see Rendering a Stream and Capturing a Stream.
AUDCLNT_BUFFERFLAGS_TIMESTAMP_ERROR	The time at which the device's stream position was recorded is uncertain. Thus, the client might be unable to accurately set the time stamp for the current data packet.

Remarks

The IAudioCaptureClient::GetBuffer and IAudioRenderClient::ReleaseBuffer methods use the constants defined in the **_AUDCLNT_BUFFERFLAGS** enumeration.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Header	audioclient.h

See also

Core Audio Enumerations

IAudioCaptureClient::GetBuffer

AUDCLNT_STREAMOPTIONS enumeration

1/11/2020 • 2 minutes to read • Edit Online

Defines values that describe the characteristics of an audio stream.

Syntax

```
typedef enum AUDCLNT_STREAMOPTIONS {
  AUDCLNT_STREAMOPTIONS_NONE,
  AUDCLNT_STREAMOPTIONS_RAW,
  AUDCLNT_STREAMOPTIONS_MATCH_FORMAT,
  AUDCLNT_STREAMOPTIONS_AMBISONICS
} ;
```

Constants

AUDCLNT_STREAMOPTIONS_NONE	No stream options.
AUDCLNT_STREAMOPTIONS_RAW	The audio stream is a 'raw' stream that bypasses all signal processing except for endpoint specific, always-on processing in the Audio Processing Object (APO), driver, and hardware.
AUDCLNT_STREAMOPTIONS_MATCH_FORMAT	The audio client is requesting that the audio engine match the format proposed by the client. The audio engine will match this format only if the format is supported by the audio driver and associated APOs. Supported in Windows 10 and later.
AUDCLNT_STREAMOPTIONS_AMBISONICS	

Requirements

Minimum supported client	Windows 8.1 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps | UWP apps]
Header	audioclient.h

See also

Core Audio Enumerations

IAudioCaptureClient interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioCaptureClient** interface enables a client to read input data from a capture endpoint buffer. The client obtains a reference to the **IAudioCaptureClient** interface on a stream object by calling the IAudioClient::GetService method with parameter *riid* set to REFIID IID_IAudioCaptureClient.

The methods in this interface manage the movement of data packets that contain capture data. The length of a data packet is expressed as the number of audio frames in the packet. The size of an audio frame is specified by the **nBlockAlign** member of the W**AVEFORMATEX (or WAVEFORMATEXTENSIBLE)** structure that the client obtains by calling the IAudioClient::GetMixFormat method. The size in bytes of an audio frame equals the number of channels in the stream multiplied by the sample size per channel. For example, the frame size is four bytes for a stereo (2-channel) stream with 16-bit samples. A packet always contains an integral number of audio frames.

When releasing an **IAudioCaptureClient** interface instance, the client must call the **Release** method of the instance from the same thread as the call to **IAudioClient::GetService** that created the object.

For a code example that uses the **IAudioCaptureClient** interface, see Capturing a Stream.

Inheritance

The **IAudioCaptureClient** interface inherits from the IUnknown interface. **IAudioCaptureClient** also has these types of members:

- Methods

Methods

The **IAudioCaptureClient** interface has these methods.

METHOD	DESCRIPTION
IAudioCaptureClient::GetBuffer	Retrieves a pointer to the next available packet of data in the capture endpoint buffer.
IAudioCaptureClient::GetNextPacketSize	The GetNextPacketSize method retrieves the number of frames in the next data packet in the capture endpoint buffer.
IAudioCaptureClient::ReleaseBuffer	The ReleaseBuffer method releases the buffer.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows

Header	audioclient.h

See also

Core Audio Interfaces

IAudioClient::GetMixFormat

IAudioClient::GetService

WASAPI

IAudioCaptureClient::GetBuffer method

1/11/2020 • 6 minutes to read • Edit Online

Retrieves a pointer to the next available packet of data in the capture endpoint buffer.

Syntax

```
HRESULT GetBuffer(
  BYTE   **ppData,
  UINT32 *pNumFramesToRead,
  DWORD  *pdwFlags,
  UINT64 *pu64DevicePosition,
  UINT64 *pu64QPCPosition
);
```

Parameters

`ppData`

Pointer to a pointer variable into which the method writes the starting address of the next data packet that is available for the client to read.

`pNumFramesToRead`

Pointer to a **UINT32** variable into which the method writes the frame count (the number of audio frames available in the data packet). The client should either read the entire data packet or none of it.

`pdwFlags`

Pointer to a **DWORD** variable into which the method writes the buffer-status flags. The method writes either 0 or the bitwise-OR combination of one or more of the following _AUDCLNT_BUFFERFLAGS enumeration values:

AUDCLNT_BUFFERFLAGS_SILENT

AUDCLNT_BUFFERFLAGS_DATA_DISCONTINUITY

AUDCLNT_BUFFERFLAGS_TIMESTAMP_ERROR

> **Note** The AUDCLNT_BUFFERFLAGS_DATA_DISCONTINUITY flag is not supported in Windows Vista.
>
> In Windows 7 and later OS releases, this flag can be used for glitch detection. To start the capture stream, the client application must call IAudioClient::Start followed by calls to **GetBuffer** in a loop to read data packets until all of the available packets in the endpoint buffer have been read. **GetBuffer** sets the AUDCLNT_BUFFERFLAGS_DATA_DISCONTINUITY flag to indicate a glitch in the buffer pointed by *ppData*.

`pu64DevicePosition`

Pointer to a **UINT64** variable into which the method writes the device position of the first audio frame in the data packet. The device position is expressed as the number of audio frames from the start of the stream. This parameter can be **NULL** if the client does not require the device position. For more information, see Remarks.

`pu64QPCPosition`

Pointer to a **UINT64** variable into which the method writes the value of the performance counter at the time that the audio endpoint device recorded the device position of the first audio frame in the data packet. The method converts the counter value to 100-nanosecond units before writing it to **pu64QPCPosition*. This parameter can be **NULL** if the client does not require the performance counter value. For more information, see Remarks.

Return value

Possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
S_OK	The call succeeded and **pNumFramesToRead* is nonzero, indicating that a packet is ready to be read.
AUDCLNT_S_BUFFER_EMPTY	The call succeeded and **pNumFramesToRead* is 0, indicating that no capture data is available to be read.
AUDCLNT_E_BUFFER_ERROR	**Windows 7 and later**: GetBuffer failed to retrieve a data buffer and **ppData* points to **NULL**. For more information, see Remarks.
AUDCLNT_E_OUT_OF_ORDER	A previous **IAudioCaptureClient::GetBuffer** call is still in effect.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AU DCL NT_ E_B UFF ER_ OPE RAT ION _PE NDI NG	Buffer cannot be accessed because a stream reset is in progress.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.
E_P OIN TER	Parameter ppData, pNumFramesToRead, or pdwFlags is **NULL**.

Remarks

This method retrieves the next data packet from the capture endpoint buffer. At a particular time, the buffer might contain zero, one, or more packets that are ready to read. Typically, a buffer-processing thread that reads data from a capture endpoint buffer reads all of the available packets each time the thread executes.

During processing of an audio capture stream, the client application alternately calls **GetBuffer** and the IAudioCaptureClient::ReleaseBuffer method. The client can read no more than a single data packet with each **GetBuffer** call. Following each **GetBuffer** call, the client must call **ReleaseBuffer** to release the packet before the client can call **GetBuffer** again to get the next packet.

Two or more consecutive calls either to **GetBuffer** or to **ReleaseBuffer** are not permitted and will fail with error code AUDCLNT_E_OUT_OF_ORDER. To ensure the correct ordering of calls, a **GetBuffer** call and its corresponding **ReleaseBuffer** call must occur in the same thread.

During each **GetBuffer** call, the caller must either obtain the entire packet or none of it. Before reading the packet, the caller can check the packet size (available through the *pNumFramesToRead* parameter) to make sure that it has enough room to store the entire packet.

During each **ReleaseBuffer** call, the caller reports the number of audio frames that it read from the buffer. This number must be either the (nonzero) packet size or 0. If the number is 0, then the next **GetBuffer** call will present the caller with the same packet as in the previous **GetBuffer** call.

Following each **GetBuffer** call, the data in the packet remains valid until the next **ReleaseBuffer** call releases the buffer.

The client must call **ReleaseBuffer** after a **GetBuffer** call that successfully obtains a packet of any size other than 0. The client has the option of calling or not calling **ReleaseBuffer** to release a packet of size 0.

The method outputs the device position and performance counter through the *pu64DevicePosition* and *pu64QPCPosition* output parameters. These values provide a time stamp for the first audio frame in the data packet. Through the *pdwFlags* output parameter, the method indicates whether the reported device position is valid.

The device position that the method writes to **pu64DevicePosition* is the stream-relative position of the audio frame that is currently playing through the speakers (for a rendering stream) or being recorded through the microphone (for a capture stream). The position is expressed as the number of frames from the start of the stream. The size of a frame in an audio stream is specified by the **nBlockAlign** member of the **WAVEFORMATEX** (or **WAVEFORMATEXTENSIBLE**) structure that specifies the stream format. The size, in bytes, of an audio frame equals the number of channels in the stream multiplied by the sample size per channel. For example, for a stereo (2-channel) stream with 16-bit samples, the frame size is four bytes.

The performance counter value that **GetBuffer** writes to **pu64QPCPosition* is not the raw counter value obtained from the **QueryPerformanceCounter** function. If *t* is the raw counter value, and if *f* is the frequency obtained from the **QueryPerformanceFrequency** function, **GetBuffer** calculates the performance counter value as follows:

**pu64QPCPosition* = 10,000,000·*t*/*f*

The result is expressed in 100-nanosecond units. For more information about **QueryPerformanceCounter** and **QueryPerformanceFrequency**, see the Windows SDK documentation.

If no new packet is currently available, the method sets **pNumFramesToRead* = 0 and returns status code AUDCLNT_S_BUFFER_EMPTY. In this case, the method does not write to the variables that are pointed to by the *ppData*, *pu64DevicePosition*, and *pu64QPCPosition* parameters.

Clients should avoid excessive delays between the **GetBuffer** call that acquires a packet and the **ReleaseBuffer** call that releases the packet. The implementation of the audio engine assumes that the **GetBuffer** call and the corresponding **ReleaseBuffer** call occur within the same buffer-processing period. Clients that delay releasing a packet for more than one period risk losing sample data.

In Windows 7 and later, **GetBuffer** can return the **AUDCLNT_E_BUFFER_ERROR** error code for an audio client that uses the endpoint buffer in the exclusive mode. This error indicates that the data buffer was not retrieved because a data packet wasn't available (**ppData* received **NULL**).

If **GetBuffer** returns **AUDCLNT_E_BUFFER_ERROR**, the thread consuming the audio samples must wait for the next processing pass. The client might benefit from keeping a count of the failed **GetBuffer** calls. If **GetBuffer** returns this error repeatedly, the client can start a new processing loop after shutting down the current client by calling IAudioClient::Stop, IAudioClient::Reset, and releasing the audio client.

Examples

For a code example that calls the **GetBuffer** method, see Capturing a Stream.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows

Header	audioclient.h

See also

IAudioCaptureClient Interface

IAudioCaptureClient::ReleaseBuffer

IAudioClient::GetMixFormat

IAudioClock::GetPosition

IAudioCaptureClient::GetNextPacketSize method

1/11/2020 • 2 minutes to read • Edit Online

The **GetNextPacketSize** method retrieves the number of frames in the next data packet in the capture endpoint buffer.

Syntax

```
HRESULT GetNextPacketSize(
  UINT32 *pNumFramesInNextPacket
);
```

Parameters

`pNumFramesInNextPacket`

Pointer to a **UINT32** variable into which the method writes the frame count (the number of audio frames in the next capture packet).

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AUDCLNT_E_DEVICEINVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AUDCLNT_E_SERVICENOTRUNNING	The Windows audio service is not running.

E_POINTER	Parameter *pNumFramesInNextPacket* is **NULL**.

Remarks

Use this method only with shared-mode streams. It does not work with exclusive-mode streams.

Before calling the IAudioCaptureClient::GetBuffer method to retrieve the next data packet, the client can call **GetNextPacketSize** to retrieve the number of audio frames in the next packet. The count reported by **GetNextPacketSize** matches the count retrieved in the **GetBuffer** call (through the *pNumFramesToRead* output parameter) that follows the **GetNextPacketSize** call.

A packet always consists of an integral number of audio frames.

GetNextPacketSize must be called in the same thread as the GetBuffer and IAudioCaptureClient::ReleaseBuffer method calls that get and release the packets in the capture endpoint buffer.

For a code example that uses the **GetNextPacketSize** method, see Capturing a Stream.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioCaptureClient Interface

IAudioCaptureClient::GetBuffer

IAudioCaptureClient::ReleaseBuffer

IAudioClient::GetCurrentPadding

IAudioCaptureClient::ReleaseBuffer method

1/11/2020 • 2 minutes to read • Edit Online

The **ReleaseBuffer** method releases the buffer.

Syntax

```
HRESULT ReleaseBuffer(
  UINT32 NumFramesRead
);
```

Parameters

`NumFramesRead`

The number of audio frames that the client read from the capture buffer. This parameter must be either equal to the number of frames in the previously acquired data packet or 0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU DCL NT_ E_IN VAL ID_S IZE	The *NumFramesRead* parameter is set to a value other than the data packet size or 0.
AU DCL NT_ E_O UT_ OF_ OR DER	This call was not preceded by a corresponding IAudioCaptureClient::GetBuffer call.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

The client should call this method when it finishes reading a data packet that it obtained previously by calling the IAudioCaptureClient::GetBuffer method.

The data in the packet that the client obtained from a GetBuffer call is guaranteed to remain valid until the client calls **ReleaseBuffer** to release the packet.

Between each GetBuffer call and its corresponding **ReleaseBuffer** call, the client must either read the entire data packet or none of it. If the client reads the entire packet following the **GetBuffer** call, then it should call **ReleaseBuffer** with *NumFramesRead* set to the total number of frames in the data packet. In this case, the next call to **GetBuffer** will produce a new data packet. If the client reads none of the data from the packet following the call to **GetBuffer**, then it should call **ReleaseBuffer** with *NumFramesRead* set to 0. In this case, the next **GetBuffer** call will produce the same data packet as in the previous **GetBuffer** call.

If the client calls **ReleaseBuffer** with *NumFramesRead* set to any value other than the packet size or 0, then the call fails and returns error code AUDCLNT_E_INVALID_SIZE.

Clients should avoid excessive delays between the GetBuffer call that acquires a buffer and the **ReleaseBuffer** call that releases the buffer. The implementation of the audio engine assumes that the **GetBuffer** call and the corresponding **ReleaseBuffer** call occur within the same buffer-processing period. Clients that delay releasing a buffer for more than one period risk losing sample data.

For a code example that calls the **ReleaseBuffer** method, see Capturing a Stream.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]

Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioCaptureClient Interface

IAudioCaptureClient::GetBuffer

IAudioClient interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioClient** interface enables a client to create and initialize an audio stream between an audio application and the audio engine (for a shared-mode stream) or the hardware buffer of an audio endpoint device (for an exclusive-mode stream). A client obtains a reference to an **IAudioClient** interface for an audio endpoint device by following these steps:

1. By using one of the techniques described in IMMDevice Interface, obtain a reference to the **IMMDevice** interface for an audio endpoint device.
2. Call the IMMDevice::Activate method with parameter *iid* set to REFIID IID_IAudioClient.

The application thread that uses this interface must be initialized for COM. For more information about COM initialization, see the description of the **CoInitializeEx** function in the Windows SDK documentation.

For code examples that use the **IAudioClient** interface, see the following topics:

- Rendering a Stream
- Capturing a Stream
- Exclusive-Mode Streams

Inheritance

The **IAudioClient** interface inherits from the IUnknown interface. **IAudioClient** also has these types of members:

- Methods

Methods

The **IAudioClient** interface has these methods.

METHOD	DESCRIPTION
IAudioClient::GetBufferSize	The GetBufferSize method retrieves the size (maximum capacity) of the endpoint buffer.
IAudioClient::GetCurrentPadding	The GetCurrentPadding method retrieves the number of frames of padding in the endpoint buffer.
IAudioClient::GetDevicePeriod	The GetDevicePeriod method retrieves the length of the periodic interval separating successive processing passes by the audio engine on the data in the endpoint buffer.
IAudioClient::GetMixFormat	The GetMixFormat method retrieves the stream format that the audio engine uses for its internal processing of shared-mode streams.
IAudioClient::GetService	The GetService method accesses additional services from the audio client object.

METHOD	DESCRIPTION
IAudioClient::GetStreamLatency	The GetStreamLatency method retrieves the maximum latency for the current stream and can be called any time after the stream has been initialized.
IAudioClient::Initialize	The Initialize method initializes the audio stream.
IAudioClient::IsFormatSupported	The IsFormatSupported method indicates whether the audio endpoint device supports a particular stream format.
IAudioClient::Reset	The Reset method resets the audio stream.
IAudioClient::SetEventHandle	The SetEventHandle method sets the event handle that the system signals when an audio buffer is ready to be processed by the client.
IAudioClient::Start	The Start method starts the audio stream.
IAudioClient::Stop	The Stop method stops the audio stream.

Remarks

> **Note** In Windows 8, the first use of **IAudioClient** to access the audio device should be on the STA thread. Calls from an MTA thread may result in undefined behavior.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

Core Audio Interfaces

IMMDevice::Activate

WASAPI

IAudioClient::GetBufferSize method

1/11/2020 • 2 minutes to read • Edit Online

The **GetBufferSize** method retrieves the size (maximum capacity) of the endpoint buffer.

Syntax

```
HRESULT GetBufferSize(
  UINT32 *pNumBufferFrames
);
```

Parameters

`pNumBufferFrames`

Pointer to a **UINT32** variable into which the method writes the number of audio frames that the buffer can hold.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU DCL NT_ E_N OT_ INIT IALI ZED	The audio stream has not been successfully initialized.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.
E_POINTER	Parameter *pNumBufferFrames* is **NULL**.

Remarks

This method requires prior initialization of the IAudioClient interface. All calls to this method will fail with the error AUDCLNT_E_NOT_INITIALIZED until the client initializes the audio stream by successfully calling the IAudioClient::Initialize method.

This method retrieves the length of the endpoint buffer shared between the client application and the audio engine. The length is expressed as the number of audio frames the buffer can hold. The size in bytes of an audio frame is calculated as the number of channels in the stream multiplied by the sample size per channel. For example, the frame size is four bytes for a stereo (2-channel) stream with 16-bit samples.

The IAudioClient::Initialize method allocates the buffer. The client specifies the buffer length in the *hnsBufferDuration* parameter value that it passes to the **Initialize** method. For rendering clients, the buffer length determines the maximum amount of rendering data that the application can write to the endpoint buffer during a single processing pass. For capture clients, the buffer length determines the maximum amount of capture data that the audio engine can read from the endpoint buffer during a single processing pass. The client should always call **GetBufferSize** after calling **Initialize** to determine the actual size of the allocated buffer, which might differ from the requested size.

Rendering clients can use this value to calculate the largest rendering buffer size that can be requested from IAudioRenderClient::GetBuffer during each processing pass.

For code examples that call the **GetBufferSize** method, see the following topics:

- Rendering a Stream
- Capturing a Stream

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows

Header	audioclient.h

See also

IAudioClient Interface

IAudioClient::Initialize

IAudioRenderClient::GetBuffer

IAudioClient::GetCurrentPadding method

1/11/2020 • 4 minutes to read • Edit Online

The **GetCurrentPadding** method retrieves the number of frames of padding in the endpoint buffer.

Syntax

```
HRESULT GetCurrentPadding(
  UINT32 *pNumPaddingFrames
);
```

Parameters

`pNumPaddingFrames`

Pointer to a **UINT32** variable into which the method writes the frame count (the number of audio frames of padding in the buffer).

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU DCL NT_ E_N OT_ INIT IALI ZED	The audio stream has not been successfully initialized.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.
E_POINTER	Parameter *pNumPaddingFrames* is **NULL**.

Remarks

This method requires prior initialization of the IAudioClient interface. All calls to this method will fail with the error AUDCLNT_E_NOT_INITIALIZED until the client initializes the audio stream by successfully calling the IAudioClient::Initialize method.

This method retrieves a padding value that indicates the amount of valid, unread data that the endpoint buffer currently contains. A rendering application can use the padding value to determine how much new data it can safely write to the endpoint buffer without overwriting previously written data that the audio engine has not yet read from the buffer. A capture application can use the padding value to determine how much new data it can safely read from the endpoint buffer without reading invalid data from a region of the buffer to which the audio engine has not yet written valid data.

The padding value is expressed as a number of audio frames. The size of an audio frame is specified by the **nBlockAlign** member of the WAVEFORMATEX (or WAVEFORMATEXTENSIBLE) structure that the client passed to the IAudioClient::Initialize method. The size in bytes of an audio frame equals the number of channels in the stream multiplied by the sample size per channel. For example, the frame size is four bytes for a stereo (2-channel) stream with 16-bit samples.

For a shared-mode rendering stream, the padding value reported by **GetCurrentPadding** specifies the number of audio frames that are queued up to play in the endpoint buffer. Before writing to the endpoint buffer, the client can calculate the amount of available space in the buffer by subtracting the padding value from the buffer length. To ensure that a subsequent call to the IAudioRenderClient::GetBuffer method succeeds, the client should request a packet length that does not exceed the available space in the buffer. To obtain the buffer length, call the IAudioClient::GetBufferSize method.

For a shared-mode capture stream, the padding value reported by **GetCurrentPadding** specifies the number of frames of capture data that are available in the next packet in the endpoint buffer. At a particular moment, zero, one, or more packets of capture data might be ready for the client to read from the buffer. If no packets are currently available, the method reports a padding value of 0. Following the **GetCurrentPadding** call, an IAudioCaptureClient::GetBuffer method call will retrieve a packet whose length exactly equals the padding value reported by **GetCurrentPadding**. Each call to GetBuffer retrieves a whole packet. A packet always contains an integral number of audio frames.

For a shared-mode capture stream, calling **GetCurrentPadding** is equivalent to calling the IAudioCaptureClient::GetNextPacketSize method. That is, the padding value reported by **GetCurrentPadding** is equal to the packet length reported by **GetNextPacketSize**.

For an exclusive-mode rendering or capture stream that was initialized with the AUDCLNT_STREAMFLAGS_EVENTCALLBACK flag, the client typically has no use for the padding value reported by **GetCurrentPadding**. Instead, the client accesses an entire buffer during each processing pass. Each time a buffer becomes available for processing, the audio engine notifies the client by signaling the client's event handle. For more information about this flag, see IAudioClient::Initialize.

For an exclusive-mode rendering or capture stream that was not initialized with the AUDCLNT_STREAMFLAGS_EVENTCALLBACK flag, the client can use the padding value obtained from **GetCurrentPadding** in a way that is similar to that described previously for a shared-mode stream. The details are as follows.

First, for an exclusive-mode rendering stream, the padding value specifies the number of audio frames that are queued up to play in the endpoint buffer. As before, the client can calculate the amount of available space in the buffer by subtracting the padding value from the buffer length.

Second, for an exclusive-mode capture stream, the padding value reported by **GetCurrentPadding** specifies the current length of the next packet. However, this padding value is a snapshot of the packet length, which might increase before the client calls the IAudioCaptureClient::GetBuffer method. Thus, the length of the packet retrieved by **GetBuffer** is at least as large as, but might be larger than, the padding value reported by the **GetCurrentPadding** call that preceded the **GetBuffer** call. In contrast, for a shared-mode capture stream, the length of the packet obtained from **GetBuffer** always equals the padding value reported by the preceding **GetCurrentPadding** call.

For a code example that calls the **GetCurrentPadding** method, see Rendering a Stream.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioCaptureClient::GetBuffer

IAudioCaptureClient::GetNextPacketSize

IAudioClient Interface

IAudioClient::Initialize

IAudioRenderClient::GetBuffer

IAudioClient::GetDevicePeriod method

1/11/2020 • 3 minutes to read • Edit Online

The **GetDevicePeriod** method retrieves the length of the periodic interval separating successive processing passes by the audio engine on the data in the endpoint buffer.

Syntax

```
HRESULT GetDevicePeriod(
  REFERENCE_TIME *phnsDefaultDevicePeriod,
  REFERENCE_TIME *phnsMinimumDevicePeriod
);
```

Parameters

`phnsDefaultDevicePeriod`

Pointer to a REFERENCE_TIME variable into which the method writes a time value specifying the default interval between periodic processing passes by the audio engine. The time is expressed in 100-nanosecond units. For information about **REFERENCE_TIME**, see the Windows SDK documentation.

`phnsMinimumDevicePeriod`

Pointer to a REFERENCE_TIME variable into which the method writes a time value specifying the minimum interval between periodic processing passes by the audio endpoint device. The time is expressed in 100-nanosecond units.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU **DCL** **NT_** **E_D** **EVI** **CE_I** **NV** **ALI** **DAT** **ED**	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.
E_POINTER	Parameters *phnsDefaultDevicePeriod* and *phnsMinimumDevicePeriod* are both **NULL**.

Remarks

The client can call this method before calling the IAudioClient::Initialize method.

The *phnsDefaultDevicePeriod* parameter specifies the default scheduling period for a shared-mode stream. The *phnsMinimumDevicePeriod* parameter specifies the minimum scheduling period for an exclusive-mode stream.

At least one of the two parameters, *phnsDefaultDevicePeriod* and *phnsMinimumDevicePeriod*, must be non-**NULL** or the method returns immediately with error code E_POINTER. If both parameters are non-**NULL**, then the method outputs both the default and minimum periods.

For a shared-mode stream, the audio engine periodically processes the data in the endpoint buffer, which the engine shares with the client application. The engine schedules itself to perform these processing passes at regular intervals.

The period between processing passes by the audio engine is fixed for a particular audio endpoint device and represents the smallest processing quantum for the audio engine. This period plus the stream latency between the buffer and endpoint device represents the minimum possible latency that an audio application can achieve.

The client has the option of scheduling its periodic processing thread to run at the same time interval as the audio engine. In this way, the client can achieve the smallest possible latency for a shared-mode stream. However, in an application for which latency is less important, the client can reduce the process-switching overhead on the CPU by scheduling its processing passes to occur less frequently. In this case, the endpoint buffer must be proportionally larger to compensate for the longer period between processing passes.

The client determines the buffer size during its call to the IAudioClient::Initialize method. For a shared-mode stream, if the client passes this method an *hnsBufferDuration* parameter value of 0, the method assumes that the periods for the client and audio engine are guaranteed to be equal, and the method will allocate a buffer small enough to achieve the minimum possible latency. (In fact, any *hnsBufferDuration* value between 0 and the sum of the audio engine's period and device latency will have the same result.) Similarly, for an exclusive-mode stream, if the client sets *hnsBufferDuration* to 0, the method assumes that the period of the client is set to the minimum period of the audio endpoint device, and the method will allocate a buffer small enough to achieve the minimum possible latency.

If the client chooses to run its periodic processing thread less often, at the cost of increased latency, it can do so as long as it creates an endpoint buffer during the IAudioClient::Initialize call that is sufficiently large.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient Interface

IAudioClient::Initialize

IAudioClient::GetMixFormat method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMixFormat** method retrieves the stream format that the audio engine uses for its internal processing of shared-mode streams.

Syntax

```
HRESULT GetMixFormat(
  WAVEFORMATEX **ppDeviceFormat
);
```

Parameters

`ppDeviceFormat`

Pointer to a pointer variable into which the method writes the address of the mix format. This parameter must be a valid, non-**NULL** pointer to a pointer variable. The method writes the address of a **WAVEFORMATEX** (or **WAVEFORMATEXTENSIBLE**) structure to this variable. The method allocates the storage for the structure. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the **GetMixFormat** call fails, **ppDeviceFormat* is **NULL**. For information about **WAVEFORMATEX**, **WAVEFORMATEXTENSIBLE**, and **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU **DCL** **NT_** **E_D** **EVI** **CE_I** **NV** **ALI** **DAT** **ED**	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.
E_POINTER	Parameter *ppDeviceFormat* is **NULL**.
E_OUTOFMEMORY	Out of memory.

Remarks

The client can call this method before calling the IAudioClient::Initialize method. When creating a shared-mode stream for an audio endpoint device, the **Initialize** method always accepts the stream format obtained from a **GetMixFormat** call on the same device.

The mix format is the format that the audio engine uses internally for digital processing of shared-mode streams. This format is not necessarily a format that the audio endpoint device supports. Thus, the caller might not succeed in creating an exclusive-mode stream with a format obtained by calling **GetMixFormat**.

For example, to facilitate digital audio processing, the audio engine might use a mix format that represents samples as floating-point values. If the device supports only integer PCM samples, then the engine converts the samples to or from integer PCM values at the connection between the device and the engine. However, to avoid resampling, the engine might use a mix format with a sample rate that the device supports.

To determine whether the **Initialize** method can create a shared-mode or exclusive-mode stream with a particular format, call the IAudioClient::IsFormatSupported method.

By itself, a **WAVEFORMATEX** structure cannot specify the mapping of channels to speaker positions. In addition, although **WAVEFORMATEX** specifies the size of the container for each audio sample, it cannot specify the number of bits of precision in a sample (for example, 20 bits of precision in a 24-bit container). However, the **WAVEFORMATEXTENSIBLE** structure can specify both the mapping of channels to speakers and the number of bits of precision in each sample. For this reason, the **GetMixFormat** method retrieves a format descriptor that is in the form of a **WAVEFORMATEXTENSIBLE** structure instead of a standalone **WAVEFORMATEX** structure. Through the *ppDeviceFormat* parameter, the method outputs a pointer to the **WAVEFORMATEX** structure that is embedded at the start of this **WAVEFORMATEXTENSIBLE** structure. For more information about **WAVEFORMATEX** and **WAVEFORMATEXTENSIBLE**, see the Windows DDK documentation.

For more information about the **GetMixFormat** method, see Device Formats. For code examples that call **GetMixFormat**, see the following topics:

- Rendering a Stream
- Capturing a Stream

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient Interface

IAudioClient::Initialize

IAudioClient::IsFormatSupported

IAudioClient::GetService method

1/11/2020 • 3 minutes to read • Edit Online

The **GetService** method accesses additional services from the audio client object.

Syntax

```
HRESULT GetService(
  REFIID riid,
  void   **ppv
);
```

Parameters

`riid`

The interface ID for the requested service. The client should set this parameter to one of the following REFIID values:

IID_IAudioCaptureClient

IID_IAudioClock

IID_IAudioRenderClient

IID_IAudioSessionControl

IID_IAudioStreamVolume

IID_IChannelAudioVolume

IID_IMFTrustedOutput

IID_ISimpleAudioVolume

For more information, see Remarks.

`ppv`

Pointer to a pointer variable into which the method writes the address of an instance of the requested interface. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetService** call fails, **ppv* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION

E_POINTER	Parameter *ppv* is **NULL**.
E_NOINTERFACE	The requested interface is not available.
AUDCLNT_E_NOT_INITIALIZED	The audio stream has not been initialized.
AUDCLNT_E_WRONG_ENDPOINT_TYPE	The caller tried to access an IAudioCaptureClient interface on a rendering endpoint, or an IAudioRenderClient interface on a capture endpoint.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

This method requires prior initialization of the IAudioClient interface. All calls to this method will fail with the error AUDCLNT_E_NOT_INITIALIZED until the client initializes the audio stream by successfully calling the IAudioClient::Initialize method.

The **GetService** method supports the following service interfaces:

- IAudioCaptureClient
- IAudioClock
- IAudioRenderClient
- IAudioSessionControl
- IAudioStreamVolume
- IChannelAudioVolume
- IMFTrustedOutput
- ISimpleAudioVolume

In Windows 7, a new service identifier, **IID_IMFTrustedOutput**, has been added that facilitates the use of output trust authority (OTA) objects. These objects can operate inside or outside the Media Foundation's protected media path (PMP) and send content outside the Media Foundation pipeline. If the caller is outside PMP, then the OTA may not operate in the PMP, and the protection settings are less robust. OTAs must implement the IMFTrustedOutput interface. By passing **IID_IMFTrustedOutput** in **GetService**, an application can retrieve a pointer to the object's **IMFTrustedOutput** interface. For more information about protected objects and **IMFTrustedOutput**, see "Protected Media Path" in the Media Foundation SDK documentation.

For information about using trusted audio drivers in OTAs, see Protected User Mode Audio (PUMA).

Note that activating IMFTrustedOutput through this mechanism works regardless of whether the caller is running in PMP. However, if the caller is not running in a protected process (that is, the caller is not within Media Foundation's PMP) then the audio OTA might not operate in the PMP and the protection settings are less robust.

To obtain the interface ID for a service interface, use the **__uuidof** operator. For example, the interface ID of **IAudioCaptureClient** is defined as follows:

```
const IID IID_IAudioCaptureClient  __uuidof(IAudioCaptureClient)
```

For information about the **__uuidof** operator, see the Windows SDK documentation.

To release the **IAudioClient** object and free all its associated resources, the client must release all references to any service objects that were created by calling **GetService**, in addition to calling **Release** on the **IAudioClient** interface itself. The client must release a service from the same thread that releases the **IAudioClient** object.

The **IAudioSessionControl**, **IAudioStreamVolume**, **IChannelAudioVolume**, and **ISimpleAudioVolume** interfaces control and monitor aspects of audio sessions and shared-mode streams. These interfaces do not work with exclusive-mode streams.

For code examples that call the **GetService** method, see the following topics:

- Rendering a Stream
- Capturing a Stream

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioCaptureClient Interface

IAudioClient Interface

IAudioClient::Initialize

IAudioClock Interface

IAudioRenderClient Interface

IAudioSessionControl Interface

IAudioStreamVolume Interface

IChannelAudioVolume Interface

ISimpleAudioVolume Interface

IAudioClient::GetStreamLatency method

1/11/2020 • 2 minutes to read • Edit Online

The **GetStreamLatency** method retrieves the maximum latency for the current stream and can be called any time after the stream has been initialized.

Syntax

```
HRESULT GetStreamLatency(
  REFERENCE_TIME *phnsLatency
);
```

Parameters

`phnsLatency`

Pointer to a REFERENCE_TIME variable into which the method writes a time value representing the latency. The time is expressed in 100-nanosecond units. For more information about **REFERENCE_TIME**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU DCL NT_ E_N OT_ INIT IALI ZED	The audio stream has not been successfully initialized.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.
E_POINTER	Parameter *phnsLatency* is **NULL**.

Remarks

This method requires prior initialization of the IAudioClient interface. All calls to this method will fail with the error AUDCLNT_E_NOT_INITIALIZED until the client initializes the audio stream by successfully calling the IAudioClient::Initialize method.

This method retrieves the maximum latency for the current stream. The value will not change for the lifetime of the IAudioClient object.

Rendering clients can use this latency value to compute the minimum amount of data that they can write during any single processing pass. To write less than this minimum is to risk introducing glitches into the audio stream. For more information, see IAudioRenderClient::GetBuffer.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient Interface

IAudioClient::Initialize

IAudioRenderClient::GetBuffer

IAudioClient::Initialize method

1/11/2020 • 17 minutes to read • Edit Online

The **Initialize** method initializes the audio stream.

Syntax

```
HRESULT Initialize(
  AUDCLNT_SHAREMODE  ShareMode,
  DWORD              StreamFlags,
  REFERENCE_TIME     hnsBufferDuration,
  REFERENCE_TIME     hnsPeriodicity,
  const WAVEFORMATEX *pFormat,
  LPCGUID            AudioSessionGuid
);
```

Parameters

`ShareMode`

The sharing mode for the connection. Through this parameter, the client tells the audio engine whether it wants to share the audio endpoint device with other clients. The client should set this parameter to one of the following AUDCLNT_SHAREMODE enumeration values:

AUDCLNT_SHAREMODE_EXCLUSIVE

AUDCLNT_SHAREMODE_SHARED

`StreamFlags`

Flags to control creation of the stream. The client should set this parameter to 0 or to the bitwise OR of one or more of the AUDCLNT_STREAMFLAGS_XXX Constants or the AUDCLNT_SESSIONFLAGS_XXX Constants.

`hnsBufferDuration`

The buffer capacity as a time value. This parameter is of type **REFERENCE_TIME** and is expressed in 100-nanosecond units. This parameter contains the buffer size that the caller requests for the buffer that the audio application will share with the audio engine (in shared mode) or with the endpoint device (in exclusive mode). If the call succeeds, the method allocates a buffer that is a least this large. For more information about **REFERENCE_TIME**, see the Windows SDK documentation. For more information about buffering requirements, see Remarks.

`hnsPeriodicity`

The device period. This parameter can be nonzero only in exclusive mode. In shared mode, always set this parameter to 0. In exclusive mode, this parameter specifies the requested scheduling period for successive buffer accesses by the audio endpoint device. If the requested device period lies outside the range that is set by the device's minimum period and the system's maximum period, then the method clamps the period to that range. If this parameter is 0, the method sets the device period to its default value. To obtain the default device period, call the IAudioClient::GetDevicePeriod method. If the AUDCLNT_STREAMFLAGS_EVENTCALLBACK stream flag is set and AUDCLNT_SHAREMODE_EXCLUSIVE is set as the ShareMode, then *hnsPeriodicity* must be nonzero and equal to *hnsBufferDuration*.

`pFormat`

Pointer to a format descriptor. This parameter must point to a valid format descriptor of type **WAVEFORMATEX** (or **WAVEFORMATEXTENSIBLE**). For more information, see Remarks.

`AudioSessionGuid`

Pointer to a session GUID. This parameter points to a GUID value that identifies the audio session that the stream belongs to. If the GUID identifies a session that has been previously opened, the method adds the stream to that session. If the GUID does not identify an existing session, the method opens a new session and adds the stream to that session. The stream remains a member of the same session for its lifetime. Setting this parameter to **NULL** is equivalent to passing a pointer to a GUID_NULL value.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AUDCLNT_E_ALREADY_INITIALIZED	The **IAudioClient** object is already initialized.
AUDCLNT_E_WRONG_ENDPOINT_TYPE	The AUDCLNT_STREAMFLAGS_LOOPBACK flag is set but the endpoint device is a capture device, not a rendering device.
AUDCLNT_E_BUFFER_SIZE_NOT_ALIGNED	**Note** Applies to Windows 7 and later. The requested buffer size is not aligned. This code can be returned for a render or a capture device if the caller specified AUDCLNT_SHAREMODE_EXCLUSIVE and the AUDCLNT_STREAMFLAGS_EVENTCALLBACK flags. The caller must call Initialize again with the aligned buffer size. For more information, see Remarks.

AUDCLNT_E_BUFFER_SIZE_ERROR	**Note** Applies to Windows 7 and later. Indicates that the buffer duration value requested by an exclusive-mode client is out of range. The requested duration value for pull mode must not be greater than 500 milliseconds; for push mode the duration value must not be greater than 2 seconds.
AUDCLNT_E_CPUUSAGE_EXCEEDED	Indicates that the process-pass duration exceeded the maximum CPU usage. The audio engine keeps track of CPU usage by maintaining the number of times the process-pass duration exceeds the maximum CPU usage. The maximum CPU usage is calculated as a percent of the engine's periodicity. The percentage value is the system's CPU throttle value (within the range of 10% and 90%). If this value is not found, then the default value of 40% is used to calculate the maximum CPU usage.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AUDCLNT_E_DEVICE_IN_USE	The endpoint device is already in use. Either the device is being used in exclusive mode, or the device is being used in shared mode and the caller asked to use the device in exclusive mode.
AUDCLNT_E_ENDPOINT_CREATE_FAILED	The method failed to create the audio endpoint for the render or the capture device. This can occur if the audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_INVALID_DEVICE_PERIOD	**Note** Applies to Windows 7 and later. Indicates that the device period requested by an exclusive-mode client is greater than 500 milliseconds.
AUDCLNT_E_UNSUPPORTED_FORMAT	The audio engine (shared mode) or audio endpoint device (exclusive mode) does not support the specified format.
AUDCLNT_E_EXCLUSIVE_MODE_NOT_ALLOWED	The caller is requesting exclusive-mode use of the endpoint device, but the user has disabled exclusive-mode use of the device.
AUDCLNT_E_BUFDURATION_PERIOD_NOT_EQUAL	The AUDCLNT_STREAMFLAGS_EVENTCALLBACK flag is set but parameters *hnsBufferDuration* and *hnsPeriodicity* are not equal.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.
E_POINTER	Parameter *pFormat* is **NULL**.
E_INVALIDARG	Parameter *pFormat* points to an invalid format description; or the AUDCLNT_STREAMFLAGS_LOOPBACK flag is set but *ShareMode* is not equal to AUDCLNT_SHAREMODE_SHARED; or the AUDCLNT_STREAMFLAGS_CROSSPROCESS flag is set but *ShareMode* is equal to AUDCLNT_SHAREMODE_EXCLUSIVE. A prior call to SetClientProperties was made with an invalid category for audio/render streams.
E_OUTOFMEMORY	Out of memory.

Remarks

After activating an IAudioClient interface on an audio endpoint device, the client must successfully call **Initialize** once and only once to initialize the audio stream between the client and the device. The client can either connect directly to the audio hardware (exclusive mode) or indirectly through the audio engine (shared mode). In the **Initialize** call, the client specifies the audio data format, the buffer size, and audio session for the stream.

> **Note** In Windows 8, the first use of IAudioClient to access the audio device should be on the STA thread. Calls from an MTA thread may result in undefined behavior.

An attempt to create a shared-mode stream can succeed only if the audio device is already operating in shared mode or the device is currently unused. An attempt to create a shared-mode stream fails if the device is already operating in exclusive mode.

If a stream is initialized to be event driven and in shared mode, *ShareMode* is set to AUDCLNT_SHAREMODE_SHARED and one of the stream flags that are set includes AUDCLNT_STREAMFLAGS_EVENTCALLBACK. For such a stream, the associated application must also obtain a handle by making a call to IAudioClient::SetEventHandle. When it is time to retire the stream, the audio engine can then use the handle to release the stream objects. Failure to call **IAudioClient::SetEventHandle** before releasing the stream objects can cause a delay of several seconds (a time-out period) while the audio engine waits for an

available handle. After the time-out period expires, the audio engine then releases the stream objects.

Whether an attempt to create an exclusive-mode stream succeeds depends on several factors, including the availability of the device and the user-controlled settings that govern exclusive-mode operation of the device. For more information, see Exclusive-Mode Streams.

An **IAudioClient** object supports exactly one connection to the audio engine or audio hardware. This connection lasts for the lifetime of the **IAudioClient** object.

The client should call the following methods only *after* calling **Initialize**:

- IAudioClient::GetBufferSize
- IAudioClient::GetCurrentPadding
- IAudioClient::GetService
- IAudioClient::GetStreamLatency
- IAudioClient::Reset
- IAudioClient::SetEventHandle
- IAudioClient::Start
- IAudioClient::Stop

The following methods do not require that **Initialize** be called first:

- IAudioClient::GetDevicePeriod
- IAudioClient::GetMixFormat
- IAudioClient::IsFormatSupported

These methods can be called any time after activating the **IAudioClient** interface.

Before calling **Initialize** to set up a shared-mode or exclusive-mode connection, the client can call the IAudioClient::IsFormatSupported method to discover whether the audio engine or audio endpoint device supports a particular format in that mode. Before opening a shared-mode connection, the client can obtain the audio engine's mix format by calling the IAudioClient::GetMixFormat method.

The endpoint buffer that is shared between the client and audio engine must be large enough to prevent glitches from occurring in the audio stream between processing passes by the client and audio engine. For a rendering endpoint, the client thread periodically writes data to the buffer, and the audio engine thread periodically reads data from the buffer. For a capture endpoint, the engine thread periodically writes to the buffer, and the client thread periodically reads from the buffer. In either case, if the periods of the client thread and engine thread are not equal, the buffer must be large enough to accommodate the longer of the two periods without allowing glitches to occur.

The client specifies a buffer size through the *hnsBufferDuration* parameter. The client is responsible for requesting a buffer that is large enough to ensure that glitches cannot occur between the periodic processing passes that it performs on the buffer. Similarly, the **Initialize** method ensures that the buffer is never smaller than the minimum buffer size needed to ensure that glitches do not occur between the periodic processing passes that the engine thread performs on the buffer. If the client requests a buffer size that is smaller than the audio engine's minimum required buffer size, the method sets the buffer size to this minimum buffer size rather than to the buffer size requested by the client.

If the client requests a buffer size (through the *hnsBufferDuration* parameter) that is not an integral number of audio frames, the method rounds up the requested buffer size to the next integral number of frames.

Following the **Initialize** call, the client should call the IAudioClient::GetBufferSize method to get the precise size of the endpoint buffer. During each processing pass, the client will need the actual buffer size to calculate how much data to transfer to or from the buffer. The client calls the IAudioClient::GetCurrentPadding method to determine how much of the data in the buffer is currently available for processing.

To achieve the minimum stream latency between the client application and audio endpoint device, the client thread

should run at the same period as the audio engine thread. The period of the engine thread is fixed and cannot be controlled by the client. Making the client's period smaller than the engine's period unnecessarily increases the client thread's load on the processor without improving latency or decreasing the buffer size. To determine the period of the engine thread, the client can call the IAudioClient::GetDevicePeriod method. To set the buffer to the minimum size required by the engine thread, the client should call **Initialize** with the *hnsBufferDuration* parameter set to 0. Following the **Initialize** call, the client can get the size of the resulting buffer by calling **IAudioClient::GetBufferSize**.

A client has the option of requesting a buffer size that is larger than what is strictly necessary to make timing glitches rare or nonexistent. Increasing the buffer size does not necessarily increase the stream latency. For a rendering stream, the latency through the buffer is determined solely by the separation between the client's write pointer and the engine's read pointer. For a capture stream, the latency through the buffer is determined solely by the separation between the engine's write pointer and the client's read pointer.

The loopback flag (AUDCLNT_STREAMFLAGS_LOOPBACK) enables audio loopback. A client can enable audio loopback only on a rendering endpoint with a shared-mode stream. Audio loopback is provided primarily to support acoustic echo cancellation (AEC).

An AEC client requires both a rendering endpoint and the ability to capture the output stream from the audio engine. The engine's output stream is the global mix that the audio device plays through the speakers. If audio loopback is enabled, a client can open a capture buffer for the global audio mix by calling the IAudioClient::GetService method to obtain an IAudioCaptureClient interface on the rendering stream object. If audio loopback is not enabled, then an attempt to open a capture buffer on a rendering stream will fail. The loopback data in the capture buffer is in the device format, which the client can obtain by querying the device's PKEY_AudioEngine_DeviceFormat property.

On Windows versions prior to Windows 10, a pull-mode capture client will not receive any events when a stream is initialized with event-driven buffering (AUDCLNT_STREAMFLAGS_EVENTCALLBACK) and is loopback-enabled (AUDCLNT_STREAMFLAGS_LOOPBACK). If the stream is opened with this configuration, the **Initialize** call succeeds, but relevant events are not raised to notify the capture client each time a buffer becomes ready for processing. To work around this, initialize a render stream in event-driven mode. Each time the client receives an event for the render stream, it must signal the capture client to run the capture thread that reads the next set of samples from the capture endpoint buffer. As of Windows 10 the relevant event handles are now set for loopback-enabled streams that are active.

Note that all streams must be opened in share mode because exclusive-mode streams cannot operate in loopback mode. For more information about audio loopback, see Loopback Recording.

The AUDCLNT_STREAMFLAGS_EVENTCALLBACK flag indicates that processing of the audio buffer by the client will be event driven. WASAPI supports event-driven buffering to enable low-latency processing of both shared-mode and exclusive-mode streams.

The initial release of Windows Vista supports event-driven buffering (that is, the use of the AUDCLNT_STREAMFLAGS_EVENTCALLBACK flag) for rendering streams only.

In the initial release of Windows Vista, for capture streams, the AUDCLNT_STREAMFLAGS_EVENTCALLBACK flag is supported only in shared mode. Setting this flag has no effect for exclusive-mode capture streams. That is, although the application specifies this flag in exclusive mode through the **Initialize** call, the application will not receive any events that are usually required to capture the audio stream. In the Windows Vista Service Pack 1 release, this flag is functional in shared-mode and exclusive mode; an application can set this flag to enable event-buffering for capture streams. For more information about capturing an audio stream, see Capturing a Stream.

To enable event-driven buffering, the client must provide an event handle to the system. Following the **Initialize** call and before calling the IAudioClient::Start method to start the stream, the client must call the IAudioClient::SetEventHandle method to set the event handle. While the stream is running, the system periodically signals the event to indicate to the client that audio data is available for processing. Between processing passes, the

client thread waits on the event handle by calling a synchronization function such as **WaitForSingleObject**. For more information about synchronization functions, see the Windows SDK documentation.

For a shared-mode stream that uses event-driven buffering, the caller must set both *hnsPeriodicity* and *hnsBufferDuration* to 0. The **Initialize** method determines how large a buffer to allocate based on the scheduling period of the audio engine. Although the client's buffer processing thread is event driven, the basic buffer management process, as described previously, is unaltered. Each time the thread awakens, it should call IAudioClient::GetCurrentPadding to determine how much data to write to a rendering buffer or read from a capture buffer. In contrast to the two buffers that the **Initialize** method allocates for an exclusive-mode stream that uses event-driven buffering, a shared-mode stream requires a single buffer.

For an exclusive-mode stream that uses event-driven buffering, the caller must specify nonzero values for *hnsPeriodicity* and *hnsBufferDuration*, and the values of these two parameters must be equal. The **Initialize** method allocates two buffers for the stream. Each buffer is equal in duration to the value of the *hnsBufferDuration* parameter. Following the **Initialize** call for a rendering stream, the caller should fill the first of the two buffers before starting the stream. For a capture stream, the buffers are initially empty, and the caller should assume that each buffer remains empty until the event for that buffer is signaled. While the stream is running, the system alternately sends one buffer or the other to the client—this form of double buffering is referred to as "ping-ponging". Each time the client receives a buffer from the system (which the system indicates by signaling the event), the client must process the entire buffer. For example, if the client requests a packet size from the IAudioRenderClient::GetBuffer method that does not match the buffer size, the method fails. Calls to the **IAudioClient::GetCurrentPadding** method are unnecessary because the packet size must always equal the buffer size. In contrast to the buffering modes discussed previously, the latency for an event-driven, exclusive-mode stream depends directly on the buffer size.

As explained in Audio Sessions, the default behavior for a session that contains rendering streams is that its volume and mute settings persist across system restarts. The AUDCLNT_STREAMFLAGS_NOPERSIST flag overrides the default behavior and makes the settings nonpersistent. This flag has no effect on sessions that contain capture streams—the settings for those sessions are never persistent. In addition, the settings for a session that contains a loopback stream (a stream that is initialized with the AUDCLNT_STREAMFLAGS_LOOPBACK flag) are not persistent.

Only a session that connects to a rendering endpoint device can have persistent volume and mute settings. The first stream to be added to the session determines whether the session's settings are persistent. Thus, if the AUDCLNT_STREAMFLAGS_NOPERSIST or AUDCLNT_STREAMFLAGS_LOOPBACK flag is set during initialization of the first stream, the session's settings are not persistent. Otherwise, they are persistent. Their persistence is unaffected by additional streams that might be subsequently added or removed during the lifetime of the session object.

After a call to **Initialize** has successfully initialized an **IAudioClient** interface instance, a subsequent **Initialize** call to initialize the same interface instance will fail and return error code E_ALREADY_INITIALIZED.

If the initial call to **Initialize** fails, subsequent **Initialize** calls might fail and return error code E_ALREADY_INITIALIZED, even though the interface has not been initialized. If this occurs, release the **IAudioClient** interface and obtain a new **IAudioClient** interface from the MMDevice API before calling **Initialize** again.

For code examples that call the **Initialize** method, see the following topics:

- Rendering a Stream
- Capturing a Stream
- Exclusive-Mode Streams

Starting with Windows 7, **Initialize** can return AUDCLNT_E_BUFFER_SIZE_NOT_ALIGNED for a render or a capture device. This indicates that the buffer size, specified by the caller in the *hnsBufferDuration* parameter, is not aligned. This error code is returned only if the caller requested an exclusive-mode stream (AUDCLNT_SHAREMODE_EXCLUSIVE)

and event-driven buffering (AUDCLNT_STREAMFLAGS_EVENTCALLBACK).

If **Initialize** returns AUDCLNT_E_BUFFER_SIZE_NOT_ALIGNED, the caller must call **Initialize** again and specify the aligned buffer size. Use the following steps:

1. Call IAudioClient::GetBufferSize and receive the next-highest-aligned buffer size (in frames).
2. Call **IAudioClient::Release** to release the audio client used in the previous call that returned AUDCLNT_E_BUFFER_SIZE_NOT_ALIGNED.
3. Calculate the aligned buffer size in 100-nansecond units (hns). The buffer size is `(REFERENCE_TIME)((10000.0 * 1000 / WAVEFORMATEX.nSamplesPerSecond * nFrames) + 0.5)`. In this formula, `nFrames` is the buffer size retrieved by GetBufferSize.
4. Call the IMMDevice::Activate method with parameter *iid* set to REFIID IID_IAudioClient to create a new audio client.
5. Call **Initialize** again on the created audio client and specify the new buffer size and periodicity.

Starting with Windows 10, hardware-offloaded audio streams must be event driven. This means that if you call IAudioClient2::SetClientProperties and set the *bIsOffload* parameter of the AudioClientProperties to TRUE, you must specify the **AUDCLNT_STREAMFLAGS_EVENTCALLBACK** flag in the *StreamFlags* parameter to **IAudioClient::Initialize**.

Examples

The following example code shows how to respond to the **AUDCLNT_E_BUFFER_SIZE_NOT_ALIGNED** return code.

```
#define REFTIMES_PER_SEC  10000000

HRESULT CreateAudioClient(IMMDevice* pDevice, IAudioClient** ppAudioClient)
{
    if (!pDevice)
    {
        return E_INVALIDARG;
    }

    if (!ppAudioClient)
    {
        return E_POINTER;
    }

    HRESULT hr = S_OK;

    WAVEFORMATEX *pwfx = NULL;

    REFERENCE_TIME hnsRequestedDuration = REFTIMES_PER_SEC;

    UINT32 nFrames = 0;

    IAudioClient *pAudioClient = NULL;

    // Get the audio client.
    CHECK_HR( hr = pDevice->Activate(
        __uuidof(IAudioClient),
        CLSCTX_ALL,
        NULL,
        (void**)&pAudioClient));

    // Get the device format.
    CHECK_HR( hr = pAudioClient->GetMixFormat(&pwfx));

    // Open the stream and associate it with an audio session.
    hr = pAudioClient->Initialize(
        AUDCLNT_SHAREMODE_EXCLUSIVE,
        AUDCLNT_STREAMFLAGS_EVENTCALLBACK,
        hnsRequestedDuration,
```

```
        hnsRequestedDuration,
        pwfx,
        NULL);

    // If the requested buffer size is not aligned...
    if (hr == AUDCLNT_E_BUFFER_SIZE_NOT_ALIGNED)
    {
        // Get the next aligned frame.
        CHECK_HR( hr = pAudioClient->GetBufferSize(&nFrames));

        hnsRequestedDuration = (REFERENCE_TIME)
        ((10000.0 * 1000 / pwfx->nSamplesPerSec * nFrames) + 0.5);

        // Release the previous allocations.
        SAFE_RELEASE(pAudioClient);
        CoTaskMemFree(pwfx);

        // Create a new audio client.
        CHECK_HR( hr = pDevice->Activate(
            __uuidof(IAudioClient),
            CLSCTX_ALL,
            NULL,
            (void**)&pAudioClient));

        // Get the device format.
        CHECK_HR( hr = pAudioClient->GetMixFormat(&pwfx));

        // Open the stream and associate it with an audio session.
        CHECK_HR( hr = pAudioClient->Initialize(
            AUDCLNT_SHAREMODE_EXCLUSIVE,
            AUDCLNT_STREAMFLAGS_EVENTCALLBACK,
            hnsRequestedDuration,
            hnsRequestedDuration,
            pwfx,
            NULL));
    }
    else
    {
        CHECK_HR (hr);
    }

    // Return to the caller.
    *(ppAudioClient) = pAudioClient;
    (*ppAudioClient)->AddRef();

done:

    // Clean up.
    CoTaskMemFree(pwfx);
    SAFE_RELEASE(pAudioClient);
    return hr;
}
```

Requirements

Target Platform	Windows
Header	audioclient.h

See also

IAudioCaptureClient Interface

IAudioClient Interface

IAudioClient::GetBufferSize

IAudioClient::GetCurrentPadding

IAudioClient::GetDevicePeriod

IAudioClient::GetMixFormat

IAudioClient::GetService

IAudioClient::SetEventHandle

IAudioClient::Start

IAudioRenderClient::GetBuffer

IAudioClient::IsFormatSupported method

1/11/2020 • 4 minutes to read • Edit Online

The **IsFormatSupported** method indicates whether the audio endpoint device supports a particular stream format.

Syntax

```
HRESULT IsFormatSupported(
  AUDCLNT_SHAREMODE  ShareMode,
  const WAVEFORMATEX *pFormat,
  WAVEFORMATEX       **ppClosestMatch
);
```

Parameters

`ShareMode`

The sharing mode for the stream format. Through this parameter, the client indicates whether it wants to use the specified format in exclusive mode or shared mode. The client should set this parameter to one of the following AUDCLNT_SHAREMODE enumeration values:

AUDCLNT_SHAREMODE_EXCLUSIVE

AUDCLNT_SHAREMODE_SHARED

`pFormat`

Pointer to the specified stream format. This parameter points to a caller-allocated format descriptor of type **WAVEFORMATEX** or **WAVEFORMATEXTENSIBLE**. The client writes a format description to this structure before calling this method. For information about **WAVEFORMATEX** and **WAVEFORMATEXTENSIBLE**, see the Windows DDK documentation.

`ppClosestMatch`

Pointer to a pointer variable into which the method writes the address of a **WAVEFORMATEX** or **WAVEFORMATEXTENSIBLE** structure. This structure specifies the supported format that is closest to the format that the client specified through the *pFormat* parameter. For shared mode (that is, if the *ShareMode* parameter is AUDCLNT_SHAREMODE_SHARED), set *ppClosestMatch* to point to a valid, non-**NULL** pointer variable. For exclusive mode, set *ppClosestMatch* to **NULL**. The method allocates the storage for the structure. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the **IsFormatSupported** call fails and *ppClosestMatch* is non-**NULL**, the method sets **ppClosestMatch* to **NULL**. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

RETURN CODE	DESCRIPTION
S_OK	Succeeded and the audio endpoint device supports the specified stream format.

S_FALSE	Succeeded with a closest match to the specified format.
AUDCLNT_E_UNSUPPORTED_FORMAT	Succeeded but the specified format is not supported in exclusive mode.

If the operation fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pFormat* is **NULL**, or *ppClosestMatch* is **NULL** and *ShareMode* is AUDCLNT_SHAREMODE_SHARED.
E_INVALIDARG	Parameter *ShareMode* is a value other than AUDCLNT_SHAREMODE_SHARED or AUDCLNT_SHAREMODE_EXCLUSIVE.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

This method provides a way for a client to determine, before calling IAudioClient::Initialize, whether the audio engine supports a particular stream format.

For exclusive mode, **IsFormatSupported** returns S_OK if the audio endpoint device supports the caller-specified format, or it returns AUDCLNT_E_UNSUPPORTED_FORMAT if the device does not support the format. The *ppClosestMatch* parameter can be **NULL**. If it is not **NULL**, the method writes **NULL** to **ppClosestMatch*.

For shared mode, if the audio engine supports the caller-specified format, **IsFormatSupported** sets ***ppClosestMatch** to **NULL** and returns S_OK. If the audio engine does not support the caller-specified format but does support a similar format, the method retrieves the similar format through the *ppClosestMatch* parameter and returns S_FALSE. If the audio engine does not support the caller-specified format or any similar format, the method sets **ppClosestMatch* to **NULL** and returns AUDCLNT_E_UNSUPPORTED_FORMAT.

In shared mode, the audio engine always supports the mix format, which the client can obtain by calling the IAudioClient::GetMixFormat method. In addition, the audio engine might support similar formats that have the same sample rate and number of channels as the mix format but differ in the representation of audio sample values. The audio engine represents sample values internally as floating-point numbers, but if the caller-specified format represents sample values as integers, the audio engine typically can convert between the integer sample values and its internal floating-point representation.

The audio engine might be able to support an even wider range of shared-mode formats if the installation package for the audio device includes a local effects (LFX) audio processing object (APO) that can handle format conversions. An LFX APO is a software module that performs device-specific processing of an audio stream. The audio graph builder in the Windows audio service inserts the LFX APO into the stream between each client and the audio engine. When a client calls the **IsFormatSupported** method and the method determines that an LFX APO is installed for use with the device, the method directs the query to the LFX APO, which indicates whether it supports the caller-specified format.

For example, a particular LFX APO might accept a 6-channel surround sound stream from a client and convert the stream to a stereo format that can be played through headphones. Typically, an LFX APO supports only client formats with sample rates that match the sample rate of the mix format.

For more information about APOs, see the white papers titled "Custom Audio Effects in Windows Vista" and "Reusing the Windows Vista Audio System Effects" at the Audio Device Technologies for Windows website. For more information about the **IsFormatSupported** method, see Device Formats.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient Interface

IAudioClient::GetMixFormat

IAudioClient::Reset method

1/11/2020 • 2 minutes to read • Edit Online

The **Reset** method resets the audio stream.

Syntax

```
HRESULT Reset();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If the method succeeds and the stream was already reset, the method returns S_FALSE. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AUDCLNT_E_NOT_INITIALIZED	The audio stream has not been successfully initialized.
AUDCLNT_E_NOT_STOPPED	The audio stream was not stopped at the time the call was made.

AU DCL NT_ E_B UFF ER_ OPE RAT ION _PE NDI NG	The client is currently writing to or reading from the buffer.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

This method requires prior initialization of the IAudioClient interface. All calls to this method will fail with the error AUDCLNT_E_NOT_INITIALIZED until the client initializes the audio stream by successfully calling the IAudioClient::Initialize method.

Reset is a control method that the client calls to reset a stopped audio stream. Resetting the stream flushes all pending data and resets the audio clock stream position to 0. This method fails if it is called on a stream that is not stopped.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient Interface

IAudioClient::Initialize

IAudioClient::SetEventHandle method

1/11/2020 • 2 minutes to read • Edit Online

The **SetEventHandle** method sets the event handle that the system signals when an audio buffer is ready to be processed by the client.

Syntax

```
HRESULT SetEventHandle(
  HANDLE eventHandle
);
```

Parameters

`eventHandle`

The event handle.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *eventHandle* is **NULL** or an invalid handle.
AU DCL NT_ E_E VEN TH AN DLE _NO T_E XPE CTE D	The audio stream was not initialized for event-driven buffering.

AUDCLNT_E_NOT_INITIALIZED	The audio stream has not been successfully initialized.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

This method requires prior initialization of the IAudioClient interface. All calls to this method will fail with the error AUDCLNT_E_NOT_INITIALIZED until the client initializes the audio stream by successfully calling the IAudioClient::Initialize method.

During stream initialization, the client can, as an option, enable event-driven buffering. To do so, the client calls the IAudioClient::Initialize method with the AUDCLNT_STREAMFLAGS_EVENTCALLBACK flag set. After enabling event-driven buffering, and before calling the IAudioClient::Start method to start the stream, the client must call **SetEventHandle** to register the event handle that the system will signal each time a buffer becomes ready to be processed by the client.

The event handle should be in the nonsignaled state at the time that the client calls the Start method.

If the client has enabled event-driven buffering of a stream, but the client calls the Start method for that stream without first calling **SetEventHandle**, the **Start** call will fail and return an error code.

If the client does not enable event-driven buffering of a stream but attempts to set an event handle for the stream by calling **SetEventHandle**, the call will fail and return an error code.

For a code example that calls the **SetEventHandle** method, see Exclusive-Mode Streams.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient Interface

IAudioClient::Initialize

IAudioClient::Start

IAudioClient::Start method

1/11/2020 • 2 minutes to read • Edit Online

The **Start** method starts the audio stream.

Syntax

```
HRESULT Start();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU DCL NT_ E_N OT_ INIT IALI ZED	The audio stream has not been successfully initialized.
AU DCL NT_ E_N OT_ STO PPE D	The audio stream was not stopped at the time of the Start call.
AU DCL NT_ E_E VEN TH AN DLE _NO T_S ET	The audio stream is configured to use event-driven buffering, but the caller has not called IAudioClient::SetEventHandle to set the event handle on the stream.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

This method requires prior initialization of the IAudioClient interface. All calls to this method will fail with the error AUDCLNT_E_NOT_INITIALIZED until the client initializes the audio stream by successfully calling the IAudioClient::Initialize method.

Start is a control method that the client calls to start the audio stream. Starting the stream causes the IAudioClient object to begin streaming data between the endpoint buffer and the audio engine. It also causes the stream's audio clock to resume counting from its current position.

The first time this method is called following initialization of the stream, the IAudioClient object's stream position counter begins at 0. Otherwise, the clock resumes from its position at the time that the stream was last stopped. Resetting the stream forces the stream position back to 0.

To avoid start-up glitches with rendering streams, clients should not call **Start** until the audio engine has been initially loaded with data by calling the IAudioRenderClient::GetBuffer and IAudioRenderClient::ReleaseBuffer methods on the rendering interface.

For code examples that call the **Start** method, see the following topics:

- Rendering a Stream
- Capturing a Stream

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]

Target Platform	Windows
Header	audioclient.h

See also

IAudioClient Interface

IAudioClient::Initialize

IAudioRenderClient::GetBuffer

IAudioRenderClient::ReleaseBuffer

IAudioClient::Stop method

1/11/2020 • 2 minutes to read • Edit Online

The **Stop** method stops the audio stream.

Syntax

```
HRESULT Stop();
```

Parameters

This method has no parameters.

Return value

If the method succeeds and stops the stream, it returns S_OK. If the method succeeds and the stream was already stopped, the method returns S_FALSE. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AUDCLNT_E_NOT_INITIALIZED	The client has not been successfully initialized.
AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

This method requires prior initialization of the IAudioClient interface. All calls to this method will fail with the error AUDCLNT_E_NOT_INITIALIZED until the client initializes the audio stream by successfully calling the IAudioClient::Initialize method.

Stop is a control method that stops a running audio stream. This method stops data from streaming through the

client's connection with the audio engine. Stopping the stream freezes the stream's audio clock at its current stream position. A subsequent call to IAudioClient::Start causes the stream to resume running from that position. If necessary, the client can call the IAudioClient::Reset method to reset the position while the stream is stopped.

For code examples that call the **Stop** method, see the following topics:

- Rendering a Stream
- Capturing a Stream

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient Interface

IAudioClient::Initialize

IAudioClient::Reset

IAudioClient::Start

IAudioClient2 interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioClient2** interface is derived from the IAudioClient interface, with a set of additional methods that enable a Windows Audio Session API (WASAPI) audio client to do the following: opt in for offloading, query stream properties, and get information from the hardware that handles offloading.The audio client can be successful in creating an offloaded stream if the underlying endpoint supports the hardware audio engine, the endpoint has been enumerated and discovered by the audio system, and there are still offload pin instances available on the endpoint.

Inheritance

The **IAudioClient2** interface inherits from the IUnknown interface. **IAudioClient2** also has these types of members:

- Methods

Methods

The **IAudioClient2** interface has these methods.

METHOD	DESCRIPTION
IAudioClient2::GetBufferSizeLimits	The GetBufferSizeLimits method returns the buffer size limits of the hardware audio engine in 100-nanosecond units.
IAudioClient2::IsOffloadCapable	The IsOffloadCapable method retrieves information about whether or not the endpoint on which a stream is created is capable of supporting an offloaded audio stream.
IAudioClient2::SetClientProperties	Sets the properties of the audio stream by populating an AudioClientProperties structure.

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

AudioClientProperties

Core Audio Interfaces

IAudioClient2::GetBufferSizeLimits method

1/11/2020 • 2 minutes to read • Edit Online

The **GetBufferSizeLimits** method returns the buffer size limits of the hardware audio engine in 100-nanosecond units.

Syntax

```
HRESULT GetBufferSizeLimits(
  const WAVEFORMATEX *pFormat,
  BOOL               bEventDriven,
  REFERENCE_TIME     *phnsMinBufferDuration,
  REFERENCE_TIME     *phnsMaxBufferDuration
);
```

Parameters

`pFormat`

A pointer to the target format that is being queried for the buffer size limit.

`bEventDriven`

Boolean value to indicate whether or not the stream can be event-driven.

`phnsMinBufferDuration`

Returns a pointer to the minimum buffer size (in 100-nanosecond units) that is required for the underlying hardware audio engine to operate at the format specified in the *pFormat* parameter, without frequent audio glitching.

`phnsMaxBufferDuration`

Returns a pointer to the maximum buffer size (in 100-nanosecond units) that the underlying hardware audio engine can support for the format specified in the *pFormat* parameter.

Return value

The **GetBufferSizeLimits** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code. For example, it can return **AUDCLNT_E_DEVICEINVALIDATED**, if the device was removed and the method is called.

Remarks

The **GetBufferSizeLimits** method is a device-facing method and does not require prior audio stream initialization.

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient2

IAudioClient2::IsOffloadCapable method

1/11/2020 • 2 minutes to read • Edit Online

The **IsOffloadCapable** method retrieves information about whether or not the endpoint on which a stream is created is capable of supporting an offloaded audio stream.

Syntax

```
HRESULT IsOffloadCapable(
  AUDIO_STREAM_CATEGORY Category,
  BOOL                  *pbOffloadCapable
);
```

Parameters

`Category`

An enumeration that specifies the category of an audio stream.

`pbOffloadCapable`

A pointer to a Boolean value. **TRUE** indicates that the endpoint is offload-capable. **FALSE** indicates that the endpoint is not offload-capable.

Return value

The **IsOffloadCapable** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

AUDIO_STREAM_CATEGORY

IAudioClient2

IAudioClient2::SetClientProperties method

1/11/2020 • 2 minutes to read • Edit Online

Sets the properties of the audio stream by populating an AudioClientProperties structure.

Syntax

```
HRESULT SetClientProperties(
  const AudioClientProperties *pProperties
);
```

Parameters

`pProperties`

Pointer to an AudioClientProperties structure.

Return value

The **SetClientProperties** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Remarks

Starting with Windows 10, hardware-offloaded audio streams must be event driven. This means that if you call **IAudioClient2::SetClientProperties** and set the *bIsOffload* parameter of the AudioClientProperties to TRUE, you must specify the **AUDCLNT_STREAMFLAGS_EVENTCALLBACK** flag in the *StreamFlags* parameter to IAudioClient::Initialize.

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

AudioClientProperties

IAudioClient2

IAudioClient::Initialize

IAudioClient3 interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioClient3** interface is derived from the IAudioClient2 interface, with a set of additional methods that enable a Windows Audio Session API (WASAPI) audio client to query for the audio engine's supported periodicities and current periodicity as well as request initialization a shared audio stream with a specified periodicity.

Inheritance

The **IAudioClient3** interface inherits from IAudioClient2. **IAudioClient3** also has these types of members:

- Methods

Methods

The **IAudioClient3** interface has these methods.

METHOD	DESCRIPTION
IAudioClient3::GetCurrentSharedModeEnginePeriod	Returns the current format and periodicity of the audio engine.
IAudioClient3::GetSharedModeEnginePeriod	Returns the range of periodicities supported by the engine for the specified stream format.
IAudioClient3::InitializeSharedAudioStream	Initializes a shared stream with the specified periodicity.

Requirements

Minimum supported client	Windows 10 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

Core Audio Interfaces

IAudioClient2

IAudioClient3::GetCurrentSharedModeEnginePeriod method

1/11/2020 • 2 minutes to read • Edit Online

Returns the current format and periodicity of the audio engine. This method enables audio clients to match the current period of the audio engine.

Syntax

```
HRESULT GetCurrentSharedModeEnginePeriod(
  WAVEFORMATEX **ppFormat,
  UINT32       *pCurrentPeriodInFrames
);
```

Parameters

`ppFormat`

Type: **WAVEFORMATEX****

The current device format that is being used by the audio engine.

`pCurrentPeriodInFrames`

Type: **UINT32***

The current period of the audio engine, in audio frames.

Return value

Type: **HRESULT**

This method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Remarks

Note The values returned by this method are instantaneous values and may be invalid immediately after the call returns if, for example, another audio client sets the periodicity or format to a different value.

Note The caller is responsible for calling CoTaskMemFree to deallocate the memory of the **WAVEFORMATEX** structure populated by this method.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient3

IAudioClient3::GetSharedModeEnginePeriod method

1/11/2020 • 2 minutes to read • Edit Online

Returns the range of periodicities supported by the engine for the specified stream format. The periodicity of the engine is the rate at which the engine wakes an event-driven audio client to transfer audio data to or from the engine. The values returned depend on the characteristics of the audio client as specified through a previous call to IAudioClient2::SetClientProperties.

Syntax

```
HRESULT GetSharedModeEnginePeriod(
  const WAVEFORMATEX *pFormat,
  UINT32             *pDefaultPeriodInFrames,
  UINT32             *pFundamentalPeriodInFrames,
  UINT32             *pMinPeriodInFrames,
  UINT32             *pMaxPeriodInFrames
);
```

Parameters

`pFormat`

Type: **const WAVEFORMATEX***

The stream format for which the supported periodicities are queried.

`pDefaultPeriodInFrames`

Type: **UINT32***

The default period with which the engine will wake the client for transferring audio samples

`pFundamentalPeriodInFrames`

Type: **UINT32***

The fundamental period with which the engine will wake the client for transferring audio samples. When setting the audio engine periodicity, you must use an integral multiple of this value.

`pMinPeriodInFrames`

Type: **UINT32***

The shortest period, in audio frames, with which the audio engine will wake the client for transferring audio samples.

`pMaxPeriodInFrames`

Type: **UINT32***

The longest period, in audio frames, with which the audio engine will wake the client for transferring audio samples.

Return value

Type: **HRESULT**

This method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Remarks

Audio clients request a specific periodicity from the audio engine with the *PeriodInFrames* parameter to IAudioClient3::InitializeSharedAudioStream. The value of *PeriodInFrames* must be an integral multiple of the value returned in the *pFundamentalPeriodInFrames* parameter. *PeriodInFrames* must also be greater than or equal to the value returned in *pMinPeriodInFrames* and less than or equal to the value of *pMaxPeriodInFrames*.

For example, for a 44100 kHz format, **GetSharedModeEnginePeriod** might return:

- *pDefaultPeriodInFrames* = 448 frames (about 10.16 milliseconds)
- *pFundamentalPeriodInFrames* = 4 frames (about 0.09 milliseconds)
- *pMinPeriodInFrames* = 48 frames (about 1.09 milliseconds)
- *pMaxPeriodInFrames* = 448 frames (same as the default)

Allowed values for the *PeriodInFrames* parameter to **InitializeSharedAudioStream** would include 48 and 448. They would also include things like 96 and 128.

They would NOT include 4 (which is smaller than the minimum allowed value) or 98 (which is not a multiple of the fundamental) or 1000 (which is larger than the maximum allowed value).

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient3

IAudioClient3::InitializeSharedAudioStream method

1/11/2020 • 4 minutes to read • Edit Online

Initializes a shared stream with the specified periodicity.

Syntax

```
HRESULT InitializeSharedAudioStream(
  DWORD              StreamFlags,
  UINT32             PeriodInFrames,
  const WAVEFORMATEX *pFormat,
  LPCGUID            AudioSessionGuid
);
```

Parameters

`StreamFlags`

Type: **DWORD**

Flags to control creation of the stream. The client should set this parameter to 0 or to the bitwise OR of one or more of the supported AUDCLNT_STREAMFLAGS_XXX Constants or AUDCLNT_SESSIONFLAGS_XXX Constants. The supported AUDCLNT_STREAMFLAGS_XXX Constants for this parameter when using this method are:

- AUDCLNT_STREAMFLAGS_EVENTCALLBACK
- AUDCLNT_STREAMFLAGS_AUTOCONVERTPCM
- AUDCLNT_STREAMFLAGS_SRC_DEFAULT_QUALITY

`PeriodInFrames`

Type: **UINT32**

Periodicity requested by the client. This value must be an integral multiple of the value returned in the *pFundamentalPeriodInFrames* parameter to IAudioClient3::GetSharedModeEnginePeriod. *PeriodInFrames* must also be greater than or equal to the value returned in *pMinPeriodInFrames* and less than or equal to the value returned in *pMaxPeriodInFrames*.

`pFormat`

Type: **const WAVEFORMATEX***

Pointer to a format descriptor. This parameter must point to a valid format descriptor of type WAVEFORMATEX or WAVEFORMATEXTENSIBLE. For more information, see the Remarks section for IAudioClient::Initialize.

`AudioSessionGuid`

Type: **LPCGUID**

Pointer to a session GUID. This parameter points to a GUID value that identifies the audio session that the stream belongs to. If the GUID identifies a session that has been previously opened, the method adds the stream to that session. If the GUID does not identify an existing session, the method opens a new session and adds the stream to that session. The stream remains a member of the same session for its lifetime. Setting this parameter to **NULL** is

equivalent to passing a pointer to a GUID_NULL value.

Return value

Type: **HRESULT**

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AUDCLNT_E_ALREADY_INITIALIZED	The **IAudioClient** object is already initialized.
AUDCLNT_E_WRONG_ENDPOINT_TYPE	The AUDCLNT_STREAMFLAGS_LOOPBACK flag is set but the endpoint device is a capture device, not a rendering device.
AUDCLNT_E_CPUUSAGE_EXCEEDED	Indicates that the process-pass duration exceeded the maximum CPU usage. The audio engine keeps track of CPU usage by maintaining the number of times the process-pass duration exceeds the maximum CPU usage. The maximum CPU usage is calculated as a percent of the engine's periodicity. The percentage value is the system's CPU throttle value (within the range of 10% and 90%). If this value is not found, then the default value of 40% is used to calculate the maximum CPU usage.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_DEVICE_IN_USE	The endpoint device is already in use. Either the device is being used in exclusive mode, or the device is being used in shared mode and the caller asked to use the device in exclusive mode.
AUDCLNT_E_ENGINE_FORMAT_LOCKED	The client specified AUDCLNT_STREAMOPTIONS_MATCH_FORMAT when calling IAudioClient2::SetClientProperties, but the format of the audio engine has been locked by another client. In this case, you can call **IAudioClient2::SetClientProperties** without specifying the match format option and then use audio engine's current format.
AUDCLNT_E_ENGINE_PERIODICITY_LOCKED	The client specified AUDCLNT_STREAMOPTIONS_MATCH_FORMAT when calling IAudioClient2::SetClientProperties, but the periodicity of the audio engine has been locked by another client. In this case, you can call **IAudioClient2::SetClientProperties** without specifying the match format option and then use audio engine's current periodicity.
AUDCLNT_E_ENDPOINT_CREATE_FAILED	The method failed to create the audio endpoint for the render or the capture device. This can occur if the audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_INVALID_DEVICE_PERIOD	Indicates that the requested device period specified with the *PeriodInFrames* is not an integral multiple of the fundamental periodicity of the audio engine, is shorter than the engine's minimum period, or is longer than the engine's maximum period. Get the supported periodicity values of the engine by calling IAudioClient3::GetSharedModeEnginePeriod.
AUDCLNT_E_UNSUPPORTED_FORMAT	The audio engine (shared mode) or audio endpoint device (exclusive mode) does not support the specified format.
AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.
E_POINTER	Parameter *pFormat* is **NULL**.
E_INVALIDARG	Parameter *pFormat* points to an invalid format description; or the AUDCLNT_STREAMFLAGS_LOOPBACK flag is set but *ShareMode* is not equal to AUDCLNT_SHAREMODE_SHARED; or the AUDCLNT_STREAMFLAGS_CROSSPROCESS flag is set but *ShareMode* is equal to AUDCLNT_SHAREMODE_EXCLUSIVE. A prior call to SetClientProperties was made with an invalid category for audio/render streams.

E_OUTOFMEMORY	Out of memory.

Remarks

Unlike IAudioClient::Initialize, this method does not allow you to specify a buffer size. The buffer size is computed based on the periodicity requested with the *PeriodInFrames* parameter. It is the client app's responsibility to ensure that audio samples are transferred in and out of the buffer in a timely manner.

Audio clients should check for allowed values for the *PeriodInFrames* parameter by calling IAudioClient3::GetSharedModeEnginePeriod. The value of *PeriodInFrames* must be an integral multiple of the value returned in the *pFundamentalPeriodInFrames* parameter. *PeriodInFrames* must also be greater than or equal to the value returned in *pMinPeriodInFrames* and less than or equal to the value of *pMaxPeriodInFrames*.

For example, for a 44100 kHz format, **GetSharedModeEnginePeriod** might return:

- *pDefaultPeriodInFrames* = 448 frames (about 10.16 milliseconds)
- *pFundamentalPeriodInFrames* = 4 frames (about 0.09 milliseconds)
- *pMinPeriodInFrames* = 48 frames (about 1.09 milliseconds)
- *pMaxPeriodInFrames* = 448 frames (same as the default)

Allowed values for the *PeriodInFrames* parameter to **InitializeSharedAudioStream** would include 48 and 448. They would also include things like 96 and 128.

They would NOT include 4 (which is smaller than the minimum allowed value) or 98 (which is not a multiple of the fundamental) or 1000 (which is larger than the maximum allowed value).

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient

IAudioClient2

IAudioClient3

IAudioClock interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioClock** interface enables a client to monitor a stream's data rate and the current position in the stream. The client obtains a reference to the **IAudioClock** interface of a stream object by calling the IAudioClient::GetService method with parameter *riid* set to REFIID IID_IAudioClock.

When releasing an **IAudioClock** interface instance, the client must call the interface's Release method from the same thread as the call to **IAudioClient::GetService** that created the object.

Inheritance

The **IAudioClock** interface inherits from the IUnknown interface. **IAudioClock** also has these types of members:

- Methods

Methods

The **IAudioClock** interface has these methods.

METHOD	DESCRIPTION
IAudioClock::GetCharacteristics	The GetCharacteristics method is reserved for future use.
IAudioClock::GetFrequency	The GetFrequency method gets the device frequency.
IAudioClock::GetPosition	The GetPosition method gets the current device position.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

Core Audio Interfaces

IAudioClient::GetService

WASAPI

IAudioClock::GetCharacteristics method

1/11/2020 • 2 minutes to read • Edit Online

The **GetCharacteristics** method is reserved for future use.

Syntax

```
HRESULT GetCharacteristics(
  DWORD *pdwCharacteristics
);
```

Parameters

`pdwCharacteristics`

Pointer to a **DWORD** variable into which the method writes a value that indicates the characteristics of the audio clock.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pdwCharacteristics* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClock Interface

IAudioClock::GetFrequency method

1/11/2020 • 2 minutes to read • Edit Online

The **GetFrequency** method gets the device frequency.

Syntax

```
HRESULT GetFrequency(
  UINT64 *pu64Frequency
);
```

Parameters

`pu64Frequency`

Pointer to a **UINT64** variable into which the method writes the device frequency. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P OIN TER	Parameter *pu64Frequency* is **NULL**.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

The device frequency is the frequency generated by the hardware clock in the audio device. This method reports the device frequency in units that are compatible with those of the device position that the IAudioClock::GetPosition method reports. For example, if, for a particular stream, the **GetPosition** method expresses the position *p* as a byte offset, the **GetFrequency** method expresses the frequency *f* in bytes per second. For any stream, the offset in seconds from the start of the stream can always be reliably calculated as *p/f* regardless of the units in which *p* and *f* are expressed.

In Windows Vista, the device frequency reported by successive calls to **GetFrequency** never changes during the lifetime of a stream.

If the clock generated by an audio device runs at a nominally constant frequency, the frequency might still vary slightly over time due to drift or jitter with respect to a reference clock. The reference clock might be a wall clock or the system clock used by the **QueryPerformanceCounter** function. The **GetFrequency** method ignores such variations and simply reports a constant frequency. However, the position reported by the **IAudioClient::GetPosition** method takes all such variations into account to report an accurate position value each time it is called. For more information about **QueryPerformanceCounter**, see the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClock Interface

IAudioClock::GetPosition

IAudioClock::GetPosition method

1/11/2020 • 4 minutes to read • Edit Online

The **GetPosition** method gets the current device position.

Syntax

```
HRESULT GetPosition(
  UINT64 *pu64Position,
  UINT64 *pu64QPCPosition
);
```

Parameters

`pu64Position`

Pointer to a **UINT64** variable into which the method writes the device position. The device position is the offset from the start of the stream to the current position in the stream. However, the units in which this offset is expressed are undefined—the device position value has meaning only in relation to the frequency reported by the IAudioClock::GetFrequency method. For more information, see Remarks.

`pu64QPCPosition`

Pointer to a **UINT64** variable into which the method writes the value of the performance counter at the time that the audio endpoint device read the device position (**pu64Position*) in response to the **GetPosition** call. The method converts the counter value to 100-nanosecond time units before writing it to **pu64QPCPosition*. This parameter can be **NULL** if the client does not require the performance counter value.

Return value

If the method succeeds and obtains an accurate reading of the position, it returns S_OK. If the method succeeds but the duration of the call is long enough to detract from the accuracy of the position reading, the method returns S_FALSE. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pu64Position* is **NULL**.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

Rendering or capture clients that need to expose a clock based on the stream's current playback or record position can use this method to derive that clock.

This method retrieves two correlated stream-position values:

- Device position. The client obtains the device position through output parameter *pu64Position*. This is the stream position of the sample that is currently playing through the speakers (for a rendering stream) or being recorded through the microphone (for a capture stream).
- Performance counter. The client obtains the performance counter through output parameter *pu64QPCPosition*. This is the counter value that the method obtained by calling the **QueryPerformanceCounter** function at the time that the audio endpoint device recorded the stream position (**pu64Position*). Note that **GetPosition** converts the counter value to 100-nanosecond time units.

The device position is meaningless unless it is combined with the device frequency reported by the **IAudioClock::GetFrequency** method. The reason is that the units in which the device positions for different streams are expressed might vary according to factors such as whether the stream was opened in shared mode or exclusive mode. However, the frequency *f* obtained from **GetFrequency** is always expressed in units that are compatible with those of the device position *p*. Thus, the stream-relative offset in seconds can always be calculated as *p*/*f*.

The device position is a stream-relative offset. That is, it is specified as an offset from the start of the stream. The device position can be thought of as an offset into an idealized buffer that contains the entire stream and is contiguous from beginning to end.

Given the device position and the performance counter at the time of the **GetPosition** call, the client can provide a more timely estimate of the device position at a slightly later time by calling **QueryPerformanceCounter** to obtain the current performance counter, and extrapolating the device position based on how far the counter has advanced since the original device position was recorded. The client can call the **QueryPerformanceFrequency** function to determine the frequency of the clock that increments the counter. Before comparing the raw counter value obtained from **QueryPerformanceCounter** to the value written to **pu64QPCPosition* by **GetPosition**, convert the raw counter value to 100-nanosecond time units as follows:

1. Multiply the raw counter value by 10,000,000.
2. Divide the result by the counter frequency obtained from **QueryPerformanceFrequency**.

For more information about **QueryPerformanceCounter** and **QueryPerformanceFrequency**, see the Windows SDK documentation.

Immediately following creation of a new stream, the device position is 0. Following a call to the IAudioClient::Start method, the device position increments at a uniform rate. The IAudioClient::Stop method freezes the device position, and a subsequent **Start** call causes the device position to resume incrementing from its value at the time of the **Stop** call. A call to IAudioClient::Reset, which should only occur while the stream is stopped, resets the device position to 0.

When a new or reset rendering stream initially begins running, its device position might remain 0 for a few milliseconds until the audio data has had time to propagate from the endpoint buffer to the rendering endpoint device. The device position changes from 0 to a nonzero value when the data begins playing through the device.

Successive device readings are monotonically increasing. Although the device position might not change between two successive readings, the device position never decreases from one reading to the next.

The *pu64Position* parameter must be a valid, non-**NULL** pointer or the method will fail and return error code E_POINTER.

Position measurements might occasionally be delayed by intermittent, high-priority events. These events might be unrelated to audio. In the case of an exclusive-mode stream, the method can return S_FALSE instead of S_OK if the method succeeds but the duration of the call is long enough to detract from the accuracy of the reported position. When this occurs, the caller has the option of calling the method again to attempt to retrieve a more accurate position (as indicated by return value S_OK). However, the caller should avoid performing this test in an infinite loop in the event that the method consistently returns S_FALSE.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient::Reset

IAudioClient::Start

IAudioClient::Stop

IAudioClock Interface

IAudioClock::GetFrequency

IAudioClock2 interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioClock2** interface is used to get the current device position.

To get a reference to the **IAudioClock2** interface, the application must call **IAudioClock::QueryInterface** to request the interface pointer from the stream object's IAudioClock interface.

The client obtains a reference to the **IAudioClock** interface of a stream object by calling the IAudioClient::GetService method with parameter *riid* set to REFIID IID_IAudioClock.

When releasing an **IAudioClock2** interface instance, the client must call the interface's **Release** method from the same thread as the call to IAudioClient::GetService that created the object.

Inheritance

The **IAudioClock2** interface inherits from the IUnknown interface. **IAudioClock2** also has these types of members:

- Methods

Methods

The **IAudioClock2** interface has these methods.

METHOD	DESCRIPTION
IAudioClock2::GetDevicePosition	The GetDevicePosition method gets the current device position, in frames, directly from the hardware.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

Core Audio Interfaces

IAudioClient::GetService

IAudioClock

IAudioClock2::GetDevicePosition method

1/11/2020 • 2 minutes to read • Edit Online

The **GetDevicePosition** method gets the current device position, in frames, directly from the hardware.

Syntax

```
HRESULT GetDevicePosition(
  UINT64 *DevicePosition,
  UINT64 *QPCPosition
);
```

Parameters

`DevicePosition`

Receives the device position, in frames. The received position is an unprocessed value that the method obtains directly from the hardware. For more information, see Remarks.

`QPCPosition`

Receives the value of the performance counter at the time that the audio endpoint device read the device position retrieved in the *DevicePosition* parameter in response to the **GetDevicePosition** call. **GetDevicePosition** converts the counter value to 100-nanosecond time units before writing it to *QPCPosition*. *QPCPosition* can be **NULL** if the client does not require the performance counter value. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *DevicePosition* is **NULL**.
AUDCLNT_E_DEVICEINVALIDATED **AUDCLNT_E_SERVICENOTRUNNING**	The audio endpoint has been disconnected.

AU **DCL** **NT_** **S_P** **OSI** **TIO** **N_S** **TAL** **LED**	The IAudioClient::Start method has not been called for this stream.

Remarks

This method only applies to shared-mode streams.

This method retrieves two correlated stream-position values:

- Device position. The client retrieves the unprocessed device position in *DevicePosition*. This is the stream position of the sample that is currently playing through the speakers (for a rendering stream) or being recorded through the microphone (for a capture stream). The sampling rate of the device endpoint may be different from the sampling rate of the mix format used by the client. To retrieve the device position from the client, call IAudioClock::GetPosition.
- Performance counter. The client retrieves the performance counter in *QPCPosition*. **GetDevicePosition** obtains the counter value by calling the **QueryPerformanceCounter** function at the time that the audio endpoint device stores the stream position in the *DevicePosition* parameter of the **GetDevicePosition** method. **GetDevicePosition** converts the counter value to 100-nanosecond time units. For more information about **QueryPerformanceCounter** and **QueryPerformanceFrequency**, see the Windows SDK documentation.

Given the device position and the performance counter at the time of the **GetDevicePosition** call, the client can get a more timely estimate of the device position at a later time by calling **QueryPerformanceCounter** to obtain the current performance counter, and extrapolating the device position based on how far the counter has advanced since the original device position was recorded. The client can call the **QueryPerformanceCounter** function to get the frequency of the clock that increments the counter. Before comparing the raw counter value obtained from **QueryPerformanceCounter** to the value retrieved by **GetDevicePosition**, convert the raw counter value to 100-nanosecond time units as follows:

1. Multiply the raw counter value by 10,000,000.
2. Divide the result by the counter frequency obtained from **QueryPerformanceCounter**.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClock2

IAudioClockAdjustment interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioClockAdjustment** interface is used to adjust the sample rate of a stream.

The client obtains a reference to the **IAudioClockAdjustment** interface of a stream object by calling the IAudioClient::GetService method with parameter *riid* set to REFIID IID_IAudioClockAdjustment. Adjusting the sample rate is not supported for exclusive mode streams.

The **IAudioClockAdjustment** interface must be obtained from an audio client that is initialized with both the AUDCLNT_STREAMFLAGS_RATEADJUST flag and the share mode set to AUDCLNT_SHAREMODE_SHARED. If Initialize is called in an exclusive mode with the AUDCLNT_STREAMFLAGS_RATEADJUST flag, **Initialize** fails with the AUDCLNT_E_UNSUPPORTED_FORMAT error code.

When releasing an **IAudioClockAdjustment** interface instance, the client must call the interface's **Release** method from the same thread as the call to IAudioClient::GetService that created the object.

Inheritance

The **IAudioClockAdjustment** interface inherits from the IUnknown interface. **IAudioClockAdjustment** also has these types of members:

- Methods

Methods

The **IAudioClockAdjustment** interface has these methods.

METHOD	DESCRIPTION
IAudioClockAdjustment::SetSampleRate	The SetSampleRate method sets the sample rate of a stream.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

AUDCLNT_STREAMFLAGS_XXX Constants

Core Audio Interfaces

IAudioClockAdjustment::SetSampleRate method

1/11/2020 • 2 minutes to read • Edit Online

The **SetSampleRate** method sets the sample rate of a stream.

Syntax

```
HRESULT SetSampleRate(
  float flSampleRate
);
```

Parameters

`flSampleRate`

The new sample rate in frames per second.

Return value

If the method succeeds, it returns S_OK.

RETURN CODE	DESCRIPTION
AU DCL NT_ E_N OT_ INIT IALI ZED	The audio stream has not been successfully initialized.
E_IN VAL IDA RG	The sample rate is out of the range for the Audio Processing Object.

Remarks

This method must not be called from a real-time processing thread.

The new sample rate will take effect after the current frame is done processing and will remain in effect until **SetSampleRate** is called again. The audio client must be initialized in shared-mode (AUDCLNT_SHAREMODE_SHARED), otherwise **SetSampleRate** fails.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

AUDCLNT_STREAMFLAGS_XXX Constants

IAudioClockAdjustment

IAudioRenderClient interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioRenderClient** interface enables a client to write output data to a rendering endpoint buffer. The client obtains a reference to the **IAudioRenderClient** interface of a stream object by calling the IAudioClient::GetService method with parameter *riid* set to **REFIID** IID_IAudioRenderClient.

The methods in this interface manage the movement of data packets that contain audio-rendering data. The length of a data packet is expressed as the number of audio frames in the packet. The size of an audio frame is specified by the **nBlockAlign** member of the **WAVEFORMATEX** structure that the client obtains by calling the IAudioClient::GetMixFormat method. The size in bytes of an audio frame equals the number of channels in the stream multiplied by the sample size per channel. For example, the frame size is four bytes for a stereo (2-channel) stream with 16-bit samples. A packet always contains an integral number of audio frames.

When releasing an **IAudioRenderClient** interface instance, the client must call the interface's **Release** method from the same thread as the call to **IAudioClient::GetService** that created the object.

For code examples that use the **IAudioRenderClient** interface, see the following topics:

- Rendering a Stream
- Exclusive-Mode Streams

Inheritance

The **IAudioRenderClient** interface inherits from the IUnknown interface. **IAudioRenderClient** also has these types of members:

- Methods

Methods

The **IAudioRenderClient** interface has these methods.

METHOD	DESCRIPTION
IAudioRenderClient::GetBuffer	Retrieves a pointer to the next available space in the rendering endpoint buffer into which the caller can write a data packet.
IAudioRenderClient::ReleaseBuffer	The ReleaseBuffer method releases the buffer space acquired in the previous call to the IAudioRenderClient::GetBuffer method.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows

Header	audioclient.h

See also

Core Audio Interfaces

IAudioClient::GetMixFormat

IAudioClient::GetService

WASAPI

IAudioRenderClient::GetBuffer method

1/11/2020 • 3 minutes to read • Edit Online

Retrieves a pointer to the next available space in the rendering endpoint buffer into which the caller can write a data packet.

Syntax

```
HRESULT GetBuffer(
  UINT32 NumFramesRequested,
  BYTE   **ppData
);
```

Parameters

`NumFramesRequested`

The number of audio frames in the data packet that the caller plans to write to the requested space in the buffer. If the call succeeds, the size of the buffer area pointed to by **ppData* matches the size specified in *NumFramesRequested*.

`ppData`

Pointer to a pointer variable into which the method writes the starting address of the buffer area into which the caller will write the data packet.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU DCL NT_ E_B UFF ER_ ERR OR	GetBuffer failed to retrieve a data buffer and **ppData* points to **NULL**. For more information, see Remarks.

AUDCLNT_E_BUFFER_TOO_LARGE	The *NumFramesRequested* value exceeds the available buffer space (buffer size minus padding size).
AUDCLNT_E_BUFFER_SIZE_ERROR	The stream is exclusive mode and uses event-driven buffering, but the client attempted to get a packet that was not the size of the buffer.
AUDCLNT_E_OUT_OF_ORDER	A previous **IAudioRenderClient::GetBuffer** call is still in effect.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AUDCLNT_E_BUFFER_OPERATION_PENDING	Buffer cannot be accessed because a stream reset is in progress.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.
E_POINTER	Parameter *ppData* is **NULL**.

Remarks

The caller can request a packet size that is less than or equal to the amount of available space in the buffer (except in the case of an exclusive-mode stream that uses event-driven buffering; for more information, see IAudioClient::Initialize). The available space is simply the buffer size minus the amount of data in the buffer that is already queued up to be played. If the caller specifies a *NumFramesRequested* value that exceeds the available space in the buffer, the call fails and returns error code AUDCLNT_E_BUFFER_TOO_LARGE.

The client is responsible for writing a sufficient amount of data to the buffer to prevent glitches from occurring in the audio stream. For more information about buffering requirements, see IAudioClient::Initialize.

After obtaining a data packet by calling **GetBuffer**, the client fills the packet with rendering data and issues the packet to the audio engine by calling the IAudioRenderClient::ReleaseBuffer method.

The client must call **ReleaseBuffer** after a **GetBuffer** call that successfully obtains a packet of any size other than 0. The client has the option of calling or not calling **ReleaseBuffer** to release a packet of size 0.

For nonzero packet sizes, the client must alternate calls to **GetBuffer** and **ReleaseBuffer**. Each **GetBuffer** call must be followed by a corresponding **ReleaseBuffer** call. After the client has called **GetBuffer** to acquire a data packet, the client cannot acquire the next data packet until it has called **ReleaseBuffer** to release the previous packet. Two or more consecutive calls either to **GetBuffer** or to **ReleaseBuffer** are not permitted and will fail with error code AUDCLNT_E_OUT_OF_ORDER.

To ensure the correct ordering of calls, a **GetBuffer** call and its corresponding **ReleaseBuffer** call must occur in the same thread.

The size of an audio frame is specified by the **nBlockAlign** member of the **WAVEFORMATEX** structure that the client obtains by calling the IAudioClient::GetMixFormat method.

If the caller sets *NumFramesRequested* = 0, the method returns status code S_OK but does not write to the variable that the *ppData* parameter points to.

Clients should avoid excessive delays between the **GetBuffer** call that acquires a buffer and the **ReleaseBuffer** call that releases the buffer. The implementation of the audio engine assumes that the **GetBuffer** call and the corresponding **ReleaseBuffer** call occur within the same buffer-processing period. Clients that delay releasing a buffer for more than one period risk losing sample data.

In Windows 7, **GetBuffer** can return the **AUDCLNT_E_BUFFER_ERROR** error code for an audio client that uses the endpoint buffer in the exclusive mode. This error indicates that the data buffer was not retrieved because a data

packet was not available (**ppData* received **NULL**).

If **GetBuffer** returns **AUDCLNT_E_BUFFER_ERROR**, the thread consuming the audio samples must wait for the next processing pass. The client might benefit from keeping a count of the failed **GetBuffer** calls. If **GetBuffer** returns this error repeatedly, the client can start a new processing loop after shutting down the current client by calling IAudioClient::Stop, IAudioClient::Reset, and releasing the audio client.

Examples

For code examples that call the **GetBuffer** method, see the following topics:

- Rendering a Stream
- Exclusive-Mode Streams

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient::GetBufferSize

IAudioClient::GetCurrentPadding

IAudioClient::Initialize

IAudioRenderClient Interface

IAudioRenderClient::ReleaseBuffer

IAudioRenderClient::ReleaseBuffer method

1/11/2020 • 2 minutes to read • Edit Online

The **ReleaseBuffer** method releases the buffer space acquired in the previous call to the IAudioRenderClient::GetBuffer method.

Syntax

```
HRESULT ReleaseBuffer(
  UINT32 NumFramesWritten,
  DWORD  dwFlags
);
```

Parameters

`NumFramesWritten`

The number of audio frames written by the client to the data packet. The value of this parameter must be less than or equal to the size of the data packet, as specified in the *NumFramesRequested* parameter passed to the IAudioRenderClient::GetBuffer method.

`dwFlags`

The buffer-configuration flags. The caller can set this parameter either to 0 or to the following _AUDCLNT_BUFFERFLAGS enumeration value (a flag bit):

AUDCLNT_BUFFERFLAGS_SILENT

If this flag bit is set, the audio engine treats the data packet as though it contains silence regardless of the data values contained in the packet. This flag eliminates the need for the client to explicitly write silence data to the rendering buffer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU DCL NT_ E_IN VAL ID_S IZE	The *NumFramesWritten* value exceeds the *NumFramesRequested* value specified in the previous IAudioRenderClient::GetBuffer call.

AUDCLNT_E_BUFFER_SIZE_ERROR	The stream is exclusive mode and uses event-driven buffering, but the client attempted to release a packet that was not the size of the buffer.
AUDCLNT_E_OUT_OF_ORDER	This call was not preceded by a corresponding call to IAudioRenderClient::GetBuffer.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.
E_INVALIDARG	Parameter *dwFlags* is not a valid value.

Remarks

The client must release the same number of frames that it requested in the preceding call to the IAudioRenderClient::GetBuffer method. The single exception to this rule is that the client can always call **ReleaseBuffer** to release 0 frames (unless the stream is exclusive mode and uses event-driven buffering).

This behavior provides a convenient means for the client to "release" a previously requested packet of length 0. In this case, the call to **ReleaseBuffer** is optional. After calling GetBuffer to obtain a packet of length 0, the client has the option of not calling **ReleaseBuffer** before calling **GetBuffer** again.

In addition, if the preceding GetBuffer call obtained a packet of nonzero size, calling **ReleaseBuffer** with *NumFramesRequested* set to 0 will succeed (unless the stream is exclusive mode and uses event-driven buffering). The meaning of the call is that the client wrote no data to the packet before releasing it. Thus, the method treats the portion of the buffer represented by the packet as unused and will make this portion of the buffer available again to the client in the next **GetBuffer** call.

Clients should avoid excessive delays between the GetBuffer call that acquires a buffer and the **ReleaseBuffer** call that releases the buffer. The implementation of the audio engine assumes that the **GetBuffer** call and the corresponding **ReleaseBuffer** call occur within the same buffer-processing period. Clients that delay releasing a buffer for more than one period risk losing sample data.

For code examples that call the **ReleaseBuffer** method, see the following topics:

- Rendering a Stream
- Exclusive-Mode Streams

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient::Initialize

IAudioRenderClient Interface

IAudioRenderClient::GetBuffer

IAudioStreamVolume interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioStreamVolume** interface enables a client to control and monitor the volume levels for all of the channels in an audio stream. The client obtains a reference to the **IAudioStreamVolume** interface on a stream object by calling the IAudioClient::GetService method with parameter *riid* set to REFIID IID_IAudioStreamVolume.

The effective volume level of any channel in the session submix, as heard at the speakers, is the product of the following four volume-level factors:

- The per-channel volume levels of the streams in the session, which clients can control through the methods in the **IAudioStreamVolume** interface.
- The per-channel volume level of the session, which clients can control through the methods in the IChannelAudioVolume interface.
- The master volume level of the session, which clients can control through the methods in the ISimpleAudioVolume interface.
- The policy-based volume level of the session, which the system dynamically assigns to the session as the global mix changes.

Each of the four volume-level factors in the preceding list is a value in the range 0.0 to 1.0, where 0.0 indicates silence and 1.0 indicates full volume (no attenuation). The effective volume level is also a value in the range 0.0 to 1.0.

When releasing an **IAudioStreamVolume** interface instance, the client must call the interface's **Release** method from the same thread as the call to **IAudioClient::GetService** that created the object.

The **IAudioStreamVolume** interface controls the channel volumes in a shared-mode audio stream. This interface does not work with exclusive-mode streams. For information about volume controls for exclusive-mode streams, see EndpointVolume API.

Inheritance

The **IAudioStreamVolume** interface inherits from the IUnknown interface. **IAudioStreamVolume** also has these types of members:

- Methods

Methods

The **IAudioStreamVolume** interface has these methods.

METHOD	DESCRIPTION
IAudioStreamVolume::GetAllVolumes	The GetAllVolumes method retrieves the volume levels for all the channels in the audio stream.
IAudioStreamVolume::GetChannelCount	The GetChannelCount method retrieves the number of channels in the audio stream.
IAudioStreamVolume::GetChannelVolume	The GetChannelVolume method retrieves the volume level for the specified channel in the audio stream.

METHOD	DESCRIPTION
IAudioStreamVolume::SetAllVolumes	The SetAllVolumes method sets the individual volume levels for all the channels in the audio stream.
IAudioStreamVolume::SetChannelVolume	The SetChannelVolume method sets the volume level for the specified channel in the audio stream.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

Core Audio Interfaces

IAudioClient::GetService

IChannelAudioVolume Interface

ISimpleAudioVolume Interface

WASAPI

IAudioStreamVolume::GetAllVolumes method

1/11/2020 • 2 minutes to read • Edit Online

The **GetAllVolumes** method retrieves the volume levels for all the channels in the audio stream.

Syntax

```
HRESULT GetAllVolumes(
  UINT32 dwCount,
  float  *pfVolumes
);
```

Parameters

`dwCount`

The number of elements in the *pfVolumes* array. The *dwCount* parameter must equal the number of channels in the stream format. To get the number of channels, call the IAudioStreamVolume::GetChannelCount method.

`pfVolumes`

Pointer to an array of volume levels for the channels in the audio stream. This parameter points to a caller-allocated **float** array into which the method writes the volume levels for the individual channels. Volume levels are in the range 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *dwCount* does not equal the number of channels in the stream.
E_POINTER	Parameter *pfVolumes* is **NULL**.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

Clients can call the IAudioStreamVolume::SetAllVolumes or IAudioStreamVolume::SetChannelVolume method to set the per-channel volume levels in an audio stream.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioStreamVolume Interface

IAudioStreamVolume::GetChannelCount

IAudioStreamVolume::SetAllVolumes

IAudioStreamVolume::SetChannelVolume

IAudioStreamVolume::GetChannelCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelCount** method retrieves the number of channels in the audio stream.

Syntax

```
HRESULT GetChannelCount(
  UINT32 *pdwCount
);
```

Parameters

`pdwCount`

Pointer to a **UINT32** variable into which the method writes the channel count.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P OIN TER	Parameter *pdwCount* is **NULL**.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

Call this method to get the number of channels in the audio stream before calling any of the other methods in the IAudioStreamVolume interface.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioStreamVolume Interface

IAudioStreamVolume::GetChannelVolume method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelVolume** method retrieves the volume level for the specified channel in the audio stream.

Syntax

```
HRESULT GetChannelVolume(
  UINT32 dwIndex,
  float  *pfLevel
);
```

Parameters

`dwIndex`

The channel number. If the stream format has *N* channels, then the channels are numbered from 0 to *N*– 1. To get the number of channels, call the IAudioStreamVolume::GetChannelCount method.

`pfLevel`

Pointer to a **float** variable into which the method writes the volume level of the specified channel. The volume level is in the range 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *dwIndex* is set to an invalid channel number.
E_P OIN TER	Parameter *pfLevel* is **NULL**.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

Clients can call the IAudioStreamVolume::SetAllVolumes or IAudioStreamVolume::SetChannelVolume method to set the per-channel volume levels in an audio stream.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioStreamVolume Interface

IAudioStreamVolume::GetChannelCount

IAudioStreamVolume::SetAllVolumes

IAudioStreamVolume::SetChannelVolume

IAudioStreamVolume::SetAllVolumes method

1/11/2020 • 2 minutes to read • Edit Online

The **SetAllVolumes** method sets the individual volume levels for all the channels in the audio stream.

Syntax

```
HRESULT SetAllVolumes(
  UINT32      dwCount,
  const float *pfVolumes
);
```

Parameters

`dwCount`

The number of elements in the *pfVolumes* array. This parameter must equal the number of channels in the stream format. To get the number of channels, call the IAudioStreamVolume::GetChannelCount method.

`pfVolumes`

Pointer to an array of volume levels for the channels in the audio stream. The number of elements in the *pfVolumes* array is specified by the *dwCount* parameter. The caller writes the volume level for each channel to the array element whose index matches the channel number. If the stream format has *N* channels, the channels are numbered from 0 to *N*– 1. Valid volume levels are in the range 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *dwCount* does not equal the number of channels in the stream, or the value of a *pfVolumes* array element is not in the range 0.0 to 1.0.
E_POINTER	Parameter *pfVolumes* is **NULL**.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioStreamVolume Interface

IAudioStreamVolume::GetChannelCount

IAudioStreamVolume::SetChannelVolume method

1/11/2020 • 2 minutes to read • Edit Online

The **SetChannelVolume** method sets the volume level for the specified channel in the audio stream.

Syntax

```
HRESULT SetChannelVolume(
  UINT32      dwIndex,
  const float fLevel
);
```

Parameters

`dwIndex`

The channel number. If the stream format has *N* channels, the channels are numbered from 0 to *N*– 1. To get the number of channels, call the IAudioStreamVolume::GetChannelCount method.

`fLevel`

The volume level for the channel. Valid volume levels are in the range 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *dwIndex* is set to an invalid channel number, or parameter *fLevel* is not in the range 0.0 to 1.0.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioStreamVolume Interface

IAudioStreamVolume::GetChannelCount

IChannelAudioVolume interface

1/23/2020 • 2 minutes to read • Edit Online

The **IChannelAudioVolume** interface enables a client to control and monitor the volume levels for all of the channels in the audio session that the stream belongs to. This is the session that the client assigned the stream to during the call to the IAudioClient::Initialize method. The client obtains a reference to the **IChannelAudioVolume** interface on a stream object by calling the IAudioClient::GetService method with parameter *riid* set to REFIID IID_IChannelAudioVolume.

The effective volume level of any channel in the session submix, as heard at the speakers, is the product of the following four volume-level factors:

- The per-channel volume levels of the streams in the session, which clients can control through the methods in the IAudioStreamVolume interface.
- The per-channel volume level of the session, which clients can control through the methods in the **IChannelAudioVolume** interface.
- The master volume level of the session, which clients can control through the methods in the ISimpleAudioVolume interface.
- The policy-based volume level of the session, which the system dynamically assigns to the session as the global mix changes.

Each of the four volume-level factors in the preceding list is a value in the range 0.0 to 1.0, where 0.0 indicates silence and 1.0 indicates full volume (no attenuation). The effective volume level is also a value in the range 0.0 to 1.0.

Typical audio applications do not modify the volume levels of sessions. Instead, they rely on users to set these volume levels through the Sndvol program. Sndvol modifies only the master volume levels of sessions. By default, the session manager sets the per-channel volume levels to 1.0 at the initial activation of a session. Subsequent per-channel volume changes by clients are persistent across computer restarts.

When releasing an **IChannelAudioVolume** interface instance, the client must call the interface's **Release** method from the same thread as the call to **IAudioClient::GetService** that created the object.

The **IChannelAudioVolume** interface controls the channel volumes in an audio session. An audio session is a collection of shared-mode streams. This interface does not work with exclusive-mode streams. For information about volume controls for exclusive-mode streams, see EndpointVolume API.

Inheritance

The **IChannelAudioVolume** interface inherits from the IUnknown interface. **IChannelAudioVolume** also has these types of members:

- Methods

Methods

The **IChannelAudioVolume** interface has these methods.

METHOD	DESCRIPTION
IChannelAudioVolume::GetAllVolumes	The GetAllVolumes method retrieves the volume levels for all the channels in the audio session.

METHOD	DESCRIPTION
IChannelAudioVolume::GetChannelCount	The GetChannelCount method retrieves the number of channels in the stream format for the audio session.
IChannelAudioVolume::GetChannelVolume	The GetChannelVolume method retrieves the volume level for the specified channel in the audio session.
IChannelAudioVolume::SetAllVolumes	The SetAllVolumes method sets the individual volume levels for all the channels in the audio session.
IChannelAudioVolume::SetChannelVolume	The SetChannelVolume method sets the volume level for the specified channel in the audio session.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

Core Audio Interfaces

IAudioClient::GetService

IAudioClient::Initialize

IAudioStreamVolume Interface

ISimpleAudioVolume Interface

WASAPI

IChannelAudioVolume::GetAllVolumes method

1/11/2020 • 2 minutes to read • Edit Online

The **GetAllVolumes** method retrieves the volume levels for all the channels in the audio session.

Syntax

```
HRESULT GetAllVolumes(
  UINT32 dwCount,
  float  *pfVolumes
);
```

Parameters

`dwCount`

The number of elements in the *pfVolumes* array. The *dwCount* parameter must equal the number of channels in the stream format for the audio session. To get the number of channels, call the IChannelAudioVolume::GetChannelCount method.

`pfVolumes`

Pointer to an array of volume levels for the channels in the audio session. This parameter points to a caller-allocated **float** array into which the method writes the volume levels for the individual channels. Volume levels are in the range 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *dwCount* does not equal the number of channels in the stream format for the audio session.
E_POINTER	Parameter *pfVolumes* is **NULL**.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

Clients can call the IChannelAudioVolume::SetAllVolumes or IChannelAudioVolume::SetChannelVolume method to set the per-channel volume levels in an audio session.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IChannelAudioVolume Interface

IChannelAudioVolume::GetChannelCount

IChannelAudioVolume::SetAllVolumes

IChannelAudioVolume::SetChannelVolume

IChannelAudioVolume::GetChannelCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelCount** method retrieves the number of channels in the stream format for the audio session.

Syntax

```
HRESULT GetChannelCount(
  UINT32 *pdwCount
);
```

Parameters

`pdwCount`

Pointer to a **UINT32** variable into which the method writes the channel count.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P OIN TER	Parameter *pdwCount* is **NULL**.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

Call this method to get the number of channels in the audio session before calling any of the other methods in the IChannelAudioVolume interface.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IChannelAudioVolume Interface

IChannelAudioVolume::GetChannelVolume method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelVolume** method retrieves the volume level for the specified channel in the audio session.

Syntax

```
HRESULT GetChannelVolume(
  UINT32 dwIndex,
  float  *pfLevel
);
```

Parameters

`dwIndex`

The channel number. If the stream format for the audio session has *N* channels, then the channels are numbered from 0 to *N*– 1. To get the number of channels, call the IChannelAudioVolume::GetChannelCount method.

`pfLevel`

Pointer to a **float** variable into which the method writes the volume level of the specified channel. The volume level is in the range 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *dwIndex* is set to an invalid channel number.
E_POINTER	Parameter *pfLevel* is **NULL**.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

Clients can call the IChannelAudioVolume::SetAllVolumes or IChannelAudioVolume::SetChannelVolume method to set the per-channel volume levels in an audio session.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IChannelAudioVolume Interface

IChannelAudioVolume::GetChannelCount

IChannelAudioVolume::SetAllVolumes

IChannelAudioVolume::SetChannelVolume

IChannelAudioVolume::SetAllVolumes method

1/11/2020 • 2 minutes to read • Edit Online

The **SetAllVolumes** method sets the individual volume levels for all the channels in the audio session.

Syntax

```
HRESULT SetAllVolumes(
  UINT32      dwCount,
  const float *pfVolumes,
  LPCGUID     EventContext
);
```

Parameters

`dwCount`

The number of elements in the *pfVolumes* array. This parameter must equal the number of channels in the stream format for the audio session. To get the number of channels, call the IChannelAudioVolume::GetChannelCount method.

`pfVolumes`

Pointer to an array of volume levels for the channels in the audio session. The number of elements in the *pfVolumes* array is specified by the *dwCount* parameter. The caller writes the volume level for each channel to the array element whose index matches the channel number. If the stream format for the audio session has *N* channels, the channels are numbered from 0 to *N*– 1. Valid volume levels are in the range 0.0 to 1.0.

`EventContext`

Pointer to the event-context GUID. If a call to this method generates a channel-volume-change event, the session manager sends notifications to all clients that have registered IAudioSessionEvents interfaces with the session manager. The session manager includes the *EventContext* pointer value with each notification. Upon receiving a notification, a client can determine whether it or another client is the source of the event by inspecting the *EventContext* value. This scheme depends on the client selecting a value for this parameter that is unique among all clients in the session. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *dwCount* does not equal the number of channels in the stream format for the audio session, or the value of a *pfVolumes* array element is not in the range 0.0 to 1.0.

E_P OIN TER	Parameter *pfVolumes* is **NULL**.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

This method, if it succeeds, generates a channel-volume-change event regardless of whether any of the new channel volume levels differ in value from the previous channel volume levels.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioSessionEvents Interface

IChannelAudioVolume Interface

IChannelAudioVolume::GetChannelCount

IChannelAudioVolume::SetChannelVolume method

1/11/2020 • 2 minutes to read • Edit Online

The **SetChannelVolume** method sets the volume level for the specified channel in the audio session.

Syntax

```
HRESULT SetChannelVolume(
  UINT32      dwIndex,
  const float fLevel,
  LPCGUID     EventContext
);
```

Parameters

`dwIndex`

The channel number. If the stream format for the audio session has *N* channels, the channels are numbered from 0 to *N*– 1. To get the number of channels, call the IChannelAudioVolume::GetChannelCount method.

`fLevel`

The volume level for the channel. Valid volume levels are in the range 0.0 to 1.0.

`EventContext`

Pointer to the event-context GUID. If a call to this method generates a channel-volume-change event, the session manager sends notifications to all clients that have registered IAudioSessionEvents interfaces with the session manager. The session manager includes the *EventContext* pointer value with each notification. Upon receiving a notification, a client can determine whether it or another client is the source of the event by inspecting the *EventContext* value. This scheme depends on the client selecting a value for this parameter that is unique among all clients in the session. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *dwIndex* is set to an invalid channel number, or parameter *fLevel* is not in the range 0.0 to 1.0.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

This method, if it succeeds, generates a channel-volume-change event regardless of whether the new channel volume level differs in value from the previous channel volume level.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioclient.h

See also

IAudioSessionEvents Interface

IChannelAudioVolume Interface

IChannelAudioVolume::GetChannelCount

ISimpleAudioVolume interface

1/23/2020 • 2 minutes to read • Edit Online

The **ISimpleAudioVolume** interface enables a client to control the master volume level of an audio session. The IAudioClient::Initialize method initializes a stream object and assigns the stream to an audio session. The client obtains a reference to the **ISimpleAudioVolume** interface on a stream object by calling the IAudioClient::GetService method with parameter *riid* set to **REFIID** IID_ISimpleAudioVolume.

Alternatively, a client can obtain the **ISimpleAudioVolume** interface of an existing session without having to first create a stream object and add the stream to the session. Instead, the client calls the IAudioSessionManager::GetSimpleAudioVolume method with the session GUID.

The effective volume level of any channel in the session submix, as heard at the speakers, is the product of the following four volume-level factors:

- The per-channel volume levels of the streams in the session, which clients can control through the methods in the IAudioStreamVolume interface.
- The master volume level of the session, which clients can control through the methods in the **ISimpleAudioVolume** interface.
- The per-channel volume level of the session, which clients can control through the methods in the IChannelAudioVolume interface.
- The policy-based volume level of the session, which the system dynamically assigns to the session as the global mix changes.

Each of the four volume-level factors in the preceding list is a value in the range 0.0 to 1.0, where 0.0 indicates silence and 1.0 indicates full volume (no attenuation). The effective volume level is also a value in the range 0.0 to 1.0.

Typical audio applications do not modify the volume levels of sessions. Instead, they rely on users to set these volume levels through the Sndvol program. Sndvol modifies only the master volume levels of sessions. By default, the session manager sets the master volume level to 1.0 at the initial activation of a session. Subsequent volume changes by Sndvol or other clients are persistent across computer restarts.

When releasing an **ISimpleAudioVolume** interface instance, the client must call the interface's **Release** method from the same thread as the call to **IAudioClient::GetService** that created the object.

The **ISimpleAudioVolume** interface controls the volume of an audio session. An audio session is a collection of shared-mode streams. This interface does not work with exclusive-mode streams. For information about volume controls for exclusive-mode streams, see EndpointVolume API.

Inheritance

The **ISimpleAudioVolume** interface inherits from the IUnknown interface. **ISimpleAudioVolume** also has these types of members:

- Methods

Methods

The **ISimpleAudioVolume** interface has these methods.

METHOD	DESCRIPTION
ISimpleAudioVolume::GetMasterVolume	The GetMasterVolume method retrieves the client volume level for the audio session.
ISimpleAudioVolume::GetMute	The GetMute method retrieves the current muting state for the audio session.
ISimpleAudioVolume::SetMasterVolume	The SetMasterVolume method sets the master volume level for the audio session.
ISimpleAudioVolume::SetMute	The SetMute method sets the muting state for the audio session.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

Core Audio Interfaces

IAudioClient::GetService

IAudioClient::Initialize

IAudioStreamVolume Interface

IChannelAudioVolume Interface

WASAPI

ISimpleAudioVolume::GetMasterVolume method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMasterVolume** method retrieves the client volume level for the audio session.

Syntax

```
HRESULT GetMasterVolume(
  float *pfLevel
);
```

Parameters

`pfLevel`

Pointer to a **float** variable into which the method writes the client volume level. The volume level is a value in the range 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P OIN TER	Parameter *pfLevel* is **NULL**.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

This method retrieves the client volume level for the session. This is the volume level that the client set in a previous call to the ISimpleAudioVolume::SetMasterVolume method.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioClient::Initialize

ISimpleAudioVolume Interface

ISimpleAudioVolume::SetMasterVolume

ISimpleAudioVolume::GetMute method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMute** method retrieves the current muting state for the audio session.

Syntax

```
HRESULT GetMute(
  BOOL *pbMute
);
```

Parameters

`pbMute`

Pointer to a **BOOL** variable into which the method writes the muting state. **TRUE** indicates that muting is enabled. **FALSE** indicates that it is disabled.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P OIN TER	Parameter *pbMute* is **NULL**.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IChannelAudioVolume Interface

ISimpleAudioVolume Interface

ISimpleAudioVolume::SetMute

ISimpleAudioVolume::SetMasterVolume method

1/11/2020 • 2 minutes to read • Edit Online

The **SetMasterVolume** method sets the master volume level for the audio session.

Syntax

```
HRESULT SetMasterVolume(
  float   fLevel,
  LPCGUID EventContext
);
```

Parameters

`fLevel`

The new master volume level. Valid volume levels are in the range 0.0 to 1.0.

`EventContext`

Pointer to the event-context GUID. If a call to this method generates a volume-change event, the session manager sends notifications to all clients that have registered IAudioSessionEvents interfaces with the session manager. The session manager includes the *EventContext* pointer value with each notification. Upon receiving a notification, a client can determine whether it or another client is the source of the event by inspecting the *EventContext* value. This scheme depends on the client selecting a value for this parameter that is unique among all clients in the session. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *fLevel* is not in the range 0.0 to 1.0.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

This method generates a volume-change event only if the method call changes the volume level of the session. For example, if the volume level is 0.4 when the call occurs, and the call sets the volume level to 0.4, no event is generated.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioSessionEvents Interface

ISimpleAudioVolume Interface

ISimpleAudioVolume::GetMasterVolume

ISimpleAudioVolume::SetMute method

1/11/2020 • 2 minutes to read • Edit Online

The **SetMute** method sets the muting state for the audio session.

Syntax

```
HRESULT SetMute(
  const BOOL bMute,
  LPCGUID    EventContext
);
```

Parameters

`bMute`

The new muting state. **TRUE** enables muting. **FALSE** disables muting.

`EventContext`

Pointer to the event-context GUID. If a call to this method generates a volume-change event, the session manager sends notifications to all clients that have registered IAudioSessionEvents interfaces with the session manager. The session manager includes the *EventContext* pointer value with each notification. Upon receiving a notification, a client can determine whether it or another client is the source of the event by inspecting the *EventContext* value. This scheme depends on the client selecting a value for this parameter that is unique among all clients in the session. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU **DCL** **NT_** **E_D** **EVI** **CE_I** **NV** **ALI** **DAT** **ED**	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

This method generates a volume-change event only if the method call changes the muting state of the session from disabled to enabled, or from enabled to disabled. For example, if muting is enabled when the call occurs, and the call enables muting, no event is generated.

This method applies the same muting state to all channels in the audio session. The endpoint device always applies muting uniformly across all the channels in the session. There are no IChannelAudioVolume methods for setting the muting states of individual channels.

The client can get the muting state of the audio session by calling the SimpleAudioVolume::GetMute method.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audioclient.h

See also

IAudioSessionEvents Interface

IChannelAudioVolume Interface

ISimpleAudioVolume Interface

ISimpleAudioVolume::GetMute

audioendpoints.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs audioendpoints.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IAudioEndpointFormatControl	Used for resetting the current audio endpoint device format.

IAudioEndpointFormatControl interface

1/23/2020 • 2 minutes to read • Edit Online

Used for resetting the current audio endpoint device format.

Inheritance

The **IAudioEndpointFormatControl** interface inherits from the IUnknown interface.
IAudioEndpointFormatControl also has these types of members:

- Methods

Methods

The **IAudioEndpointFormatControl** interface has these methods.

METHOD	DESCRIPTION
IAudioEndpointFormatControl::ResetToDefault	Resets the format to the default setting provided by the device manufacturer.

Remarks

This setting is exposed to the user through the "Sounds" control panel and can be read from the endpoint property store using PKEY_AudioEngine_DeviceFormat.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioendpoints.h

See also

Core Audio Interfaces

IAudioEndpointFormatControl::ResetToDefault method

1/11/2020 • 2 minutes to read • Edit Online

Resets the format to the default setting provided by the device manufacturer.

Syntax

```
HRESULT ResetToDefault(
  DWORD ResetFlags
);
```

Parameters

`ResetFlags`

Allows the application to specify which formats are reset. If no flags are set, then this method reevaluates both the endpoint's device format and mix format and sets them to their default values.

ENDPOINT_FORMAT_RESET_MIX_ONLY: Only reset the mix format. The endpoint's device format will not be reset if this flag is set.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audioendpoints.h

See also

IAudioEndpointFormatControl

audioenginebaseapo.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs audioenginebaseapo.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IAudioProcessingObject	System Effects Audio Processing Objects (sAPOs) are typically used in or called from real-time process threads.
IAudioProcessingObjectConfiguration	The IAudioProcessingObjectConfiguration interface is used to configure the APO. This interface uses its methods to lock and unlock the APO for processing.
IAudioProcessingObjectRT	This interface can operate in real-time mode and its methods can be called form real-time processing threads.
IAudioSystemEffects	The IAudioSystemEffects interface uses the basic methods that are inherited from IUnknown, and must implement an Initialize method.
IAudioSystemEffects2	The IAudioSystemEffects2 interface was introduced with Windows 8.1 for retrieving information about the processing objects in a given mode.
IAudioSystemEffectsCustomFormats	The IAudioSystemEffectsCustomFormats interface is supported in Windows Vista and later versions of Windows.

Structures

TITLE	DESCRIPTION
APO_REG_PROPERTIES	The APO_REG_PROPERTIES structure is used by IAudioProcessingObject::GetRegistrationProperties for returning the registration properties of an audio processing object (APO).
APOInitBaseStruct	The APOInitBaseStruct structure is the base initialization header that must precede other initialization data in IAudioProcessingObject::Initialize.
APOInitSystemEffects	The APOInitSystemEffects structure gets passed to the system effects APO for initialization.
APOInitSystemEffects2	The APOInitSystemEffects2 structure was introduced with Windows 8.1, to make it possible to provide additional initialization context to the audio processing object (APO) for initialization.

TITLE	DESCRIPTION
AudioFXExtensionParams	The AudioFXExtensionParams structure is passed to the system effects ControlPanel Extension PropertyPage via IShellPropSheetExt::AddPages.

Enumerations

TITLE	DESCRIPTION
APO_FLAG	The APO_FLAG enumeration defines constants that are used as flags by an audio processing object (APO).

audioengineendpoint.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs audioengineendpoint.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IAudioDeviceEndpoint	Initializes a device endpoint object and gets the capabilities of the device that it represents.
IAudioEndpoint	Provides information to the audio engine about an audio endpoint. This interface is implemented by an audio endpoint.
IAudioEndpointControl	Controls the stream state of an endpoint.
IAudioEndpointLastBufferControl	Provides functionality to allow an offload stream client to notify the endpoint that the last buffer has been sent only partially filled.
IAudioEndpointOffloadStreamMeter	The IAudioEndpointOffloadStreamMeter interface retrieves general information about the audio channels in the offloaded audio stream.
IAudioEndpointOffloadStreamMute	The IAudioEndpointOffloadStreamMute interface allows a client to manipulate the mute status of the offloaded audio stream.
IAudioEndpointOffloadStreamVolume	The IAudioEndpointOffloadStreamVolume interface allows the client application to manipulate the volume level of the offloaded audio stream.
IAudioEndpointRT	Gets the difference between the current read and write positions in the endpoint buffer.
IAudioInputEndpointRT	Gets the input buffer for each processing pass.
IAudioLfxControl	The IAudioLfxControl interface allows the client to apply or remove local effects from the offloaded audio stream.
IAudioOutputEndpointRT	Gets the output buffer for each processing pass.
IHardwareAudioEngineBase	The IHardwareAudioEngineBase interface is implemented by audio endpoints for the audio stack to use to configure and retrieve information about the hardware audio engine.

Structures

TITLE	DESCRIPTION
AE_CURRENT_POSITION	Reports the current frame position from the device to the clients.

Enumerations

TITLE	DESCRIPTION
AE_POSITION_FLAGS	Defines constants for the AE_CURRENT_POSITION structure. These constants describe the degree of validity of the current position.

IAudioEndpointLastBufferControl interface

1/23/2020 • 2 minutes to read • Edit Online

Provides functionality to allow an offload stream client to notify the endpoint that the last buffer has been sent only partially filled.

Inheritance

The **IAudioEndpointLastBufferControl** interface inherits from the IUnknown interface. **IAudioEndpointLastBufferControl** also has these types of members:

- Methods

Methods

The **IAudioEndpointLastBufferControl** interface has these methods.

METHOD	DESCRIPTION
IAudioEndpointLastBufferControl::IsLastBufferControlSupported	Indicates if last buffer control is supported.
IAudioEndpointLastBufferControl::ReleaseOutputDataPointerForLastBuffer	Releases the output data pointer for the last buffer.

Remarks

This is an optional interface on an endpoint.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

Core Audio Interfaces

IAudioEndpointLastBufferControl::IsLastBufferControlSupported method

1/11/2020 • 2 minutes to read • Edit Online

Indicates if last buffer control is supported.

Syntax

```
BOOL IsLastBufferControlSupported();
```

Parameters

This method has no parameters.

Return value

true if last buffer control is supported; otherwise, **false**.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioEndpointLastBufferControl

IAudioEndpointLastBufferControl::ReleaseOutputDataPointerForLastBuffer method

1/11/2020 • 2 minutes to read • Edit Online

Releases the output data pointer for the last buffer.

Syntax

```
void ReleaseOutputDataPointerForLastBuffer(
  const APO_CONNECTION_PROPERTY *pConnectionProperty
);
```

Parameters

`pConnectionProperty`

The APO connection property.

Return value

This method does not return a value.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioEndpointLastBufferControl

IAudioEndpointOffloadStreamMeter interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioEndpointOffloadStreamMeter** interface retrieves general information about the audio channels in the offloaded audio stream.

Inheritance

The **IAudioEndpointOffloadStreamMeter** interface inherits from the IUnknown interface. **IAudioEndpointOffloadStreamMeter** also has these types of members:

- Methods

Methods

The **IAudioEndpointOffloadStreamMeter** interface has these methods.

METHOD	DESCRIPTION
IAudioEndpointOffloadStreamMeter::GetMeterChannelCount	Gets the number of available audio channels in the offloaded stream that can be metered.
IAudioEndpointOffloadStreamMeter::GetMeteringData	The GetMeteringData method retrieves general information about the available audio channels in the offloaded stream.

Requirements

Target Platform	Windows
Header	audioengineendpoint.h

See also

Core Audio Interfaces

IAudioEndpointOffloadStreamMeter::GetMeterChannelCount method

1/11/2020 • 2 minutes to read • Edit Online

Gets the number of available audio channels in the offloaded stream that can be metered.

Syntax

```
HRESULT GetMeterChannelCount(
  UINT32 *pu32ChannelCount
);
```

Parameters

`pu32ChannelCount`

A Pointer to a variable that indicates the number of available audio channels in the offloaded stream that can be metered.

Return value

The **GetMeterChannelCount** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioEndpointOffloadStreamMeter

IAudioEndpointOffloadStreamMeter::GetMeteringData method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMeteringData** method retrieves general information about the available audio channels in the offloaded stream.

Syntax

```
HRESULT GetMeteringData(
  UINT32  u32ChannelCount,
  FLOAT32 *pf32PeakValues
);
```

Parameters

`u32ChannelCount`

Indicates the number of available audio channels in the offloaded audio stream.

`pf32PeakValues`

A pointer to the peak values for the audio channels in the offloaded audio stream.

Return value

The **GetMeteringData** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioEndpointOffloadStreamMeter

IAudioEndpointOffloadStreamMute interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioEndpointOffloadStreamMute** interface allows a client to manipulate the mute status of the offloaded audio stream.

Inheritance

The **IAudioEndpointOffloadStreamMute** interface inherits from the IUnknown interface. **IAudioEndpointOffloadStreamMute** also has these types of members:

- Methods

Methods

The **IAudioEndpointOffloadStreamMute** interface has these methods.

METHOD	DESCRIPTION
IAudioEndpointOffloadStreamMute::GetMute	The GetMute method retrieves the mute status of the offloaded audio stream.
IAudioEndpointOffloadStreamMute::SetMute	The SetMute method sets the mute status of the offloaded audio stream.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

Core Audio Interfaces

IAudioEndpointOffloadStreamMute::GetMute method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMute** method retrieves the mute status of the offloaded audio stream.

Syntax

```
HRESULT GetMute(
  boolean *pbMuted
);
```

Parameters

`pbMuted`

Indicates whether or not the offloaded audio stream is muted. A value of**TRUE** indicates that the stream is muted, and a value of**FALSE** indicates that the stream is not muted.

Return value

The **GetMute** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioEndpointOffloadStreamMute

IAudioEndpointOffloadStreamMute::SetMute method

1/11/2020 • 2 minutes to read • Edit Online

The **SetMute** method sets the mute status of the offloaded audio stream.

Syntax

```
HRESULT SetMute(
  boolean bMuted
);
```

Parameters

`bMuted`

Indicates whether or not the offloaded audio stream is to be muted. A value of **TRUE** mutes the stream, and a value of **FALSE** sets the stream to a non-muted state.

Return value

The **SetMute** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioEndpointOffloadStreamMute

IAudioEndpointOffloadStreamVolume interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioEndpointOffloadStreamVolume** interface allows the client application to manipulate the volume level of the offloaded audio stream.

Inheritance

The **IAudioEndpointOffloadStreamVolume** interface inherits from the IUnknown interface. **IAudioEndpointOffloadStreamVolume** also has these types of members:

- Methods

Methods

The **IAudioEndpointOffloadStreamVolume** interface has these methods.

METHOD	DESCRIPTION
IAudioEndpointOffloadStreamVolume::GetChannelVolumes	The GetChannelVolumes method retrieves the volume levels for the various audio channels in the offloaded stream.
IAudioEndpointOffloadStreamVolume::GetVolumeChannelCount	The GetVolumeChannelCount method retrieves the number of available audio channels in the offloaded stream.
IAudioEndpointOffloadStreamVolume::SetChannelVolumes	The SetChannelVolumes method sets the volume levels for the various audio channels in the offloaded stream.

Requirements

Target Platform	Windows
Header	audioengineendpoint.h

See also

Core Audio Interfaces

IAudioEndpointOffloadStreamVolume::GetChannelVolumes method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelVolumes** method retrieves the volume levels for the various audio channels in the offloaded stream.

Syntax

```
HRESULT GetChannelVolumes(
  UINT32  u32ChannelCount,
  FLOAT32 *pf32Volumes
);
```

Parameters

`u32ChannelCount`

Indicates the numer of available audio channels in the offloaded stream.

`pf32Volumes`

A pointer to the volume levels for the various audio channels in the offloaded stream.

Return value

The **GetChannelVolumes** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioEndpointOffloadStreamVolume

IAudioEndpointOffloadStreamVolume::GetVolumeChannelCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetVolumeChannelCount** method retrieves the number of available audio channels in the offloaded stream.

Syntax

```
HRESULT GetVolumeChannelCount(
  UINT32 *pu32ChannelCount
);
```

Parameters

`pu32ChannelCount`

A pointer to the number of available audio channels in the offloaded stream.

Return value

The **GetVolumeChannelCount** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioEndpointOffloadStreamVolume

IAudioEndpointOffloadStreamVolume::SetChannelVolumes method

1/11/2020 • 2 minutes to read • Edit Online

The **SetChannelVolumes** method sets the volume levels for the various audio channels in the offloaded stream.

Syntax

```
HRESULT SetChannelVolumes(
  UINT32           u32ChannelCount,
  FLOAT32          *pf32Volumes,
  AUDIO_CURVE_TYPE u32CurveType,
  HNSTIME          *pCurveDuration
);
```

Parameters

`u32ChannelCount`

Indicates the number of available audio channels in the offloaded stream.

`pf32Volumes`

A pointer to the volume levels for the various audio channels in the offloaded stream.

`u32CurveType`

A value from the AUDIO_CURVE_TYPE enumeration specifying the curve to use when changing the channel volumes.

`pCurveDuration`

A **LONGLONG** value specifying the curve duration in hundred nanosecond units.

Return value

The **SetChannelVolumes** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioEndpointOffloadStreamVolume

IAudioLfxControl interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioLfxControl** interface allows the client to apply or remove local effects from the offloaded audio stream.

Inheritance

The **IAudioLfxControl** interface inherits from the IUnknown interface. **IAudioLfxControl** also has these types of members:

- Methods

Methods

The **IAudioLfxControl** interface has these methods.

METHOD	DESCRIPTION
IAudioLfxControl::GetLocalEffectsState	The GetLocalEffectsState method retrieves the local effects state that is currently applied to the offloaded audio stream.
IAudioLfxControl::SetLocalEffectsState	The SetLocalEffectsState method sets the local effects state that is to be applied to the offloaded audio stream.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

Core Audio Interfaces

IAudioLfxControl::GetLocalEffectsState method

1/11/2020 • 2 minutes to read • Edit Online

The **GetLocalEffectsState** method retrieves the local effects state that is currently applied to the offloaded audio stream.

Syntax

```
HRESULT GetLocalEffectsState(
  BOOL *pbEnabled
);
```

Parameters

`pbEnabled`

A pointer to the Boolean variable that indicates the state of the local effects that have been applied to the offloaded audio stream. A value of **TRUE** indicates that local effects have been enabled and applied to the stream. A value of **FALSE** indicates that local effects have been disabled.

Return value

The **GetLocalEffectsState** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioLfxControl

IAudioLfxControl::SetLocalEffectsState method

1/11/2020 • 2 minutes to read • Edit Online

The **SetLocalEffectsState** method sets the local effects state that is to be applied to the offloaded audio stream.

Syntax

```
HRESULT SetLocalEffectsState(
  BOOL bEnabled
);
```

Parameters

`bEnabled`

Indicates the local effects state that is to be applied to the offloaded audio stream. A value of **TRUE** enables local effects, and the local effects in the audio graph are applied to the stream. A value of **FALSE** disables local effects, so that the local effects in the audio graph are not applied to the audio stream.

Return value

The **SetLocalEffectsState** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IAudioLfxControl

IHardwareAudioEngineBase interface

1/23/2020 • 2 minutes to read • Edit Online

The **IHardwareAudioEngineBase** interface is implemented by audio endpoints for the audio stack to use to configure and retrieve information about the hardware audio engine.

Inheritance

The **IHardwareAudioEngineBase** interface inherits from the IUnknown interface.
IHardwareAudioEngineBase also has these types of members:

- Methods

Methods

The **IHardwareAudioEngineBase** interface has these methods.

METHOD	DESCRIPTION
IHardwareAudioEngineBase::GetAvailableOffloadConnectorCount	The GetAvailableOffloadConnectorCount method retrieves the number of avaialable endpoints that can handle offloaded streams on the hardware audio engine.
IHardwareAudioEngineBase::GetEngineFormat	The GetEngineFormat method retrieves the current data format of the offloaded audio stream.
IHardwareAudioEngineBase::GetGfxState	The GetGfxState method retrieves the GFX state of the offloaded audio stream.
IHardwareAudioEngineBase::SetEngineDeviceFormat	The SetEngineDeviceFormat method sets the waveform audio format for the hardware audio engine.
IHardwareAudioEngineBase::SetGfxState	The SetGfxState method sets the GFX state of the offloaded audio stream.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

Core Audio Interfaces

IHardwareAudioEngineBase::GetAvailableOffloadConnectorCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetAvailableOffloadConnectorCount** method retrieves the number of avaialable endpoints that can handle offloaded streams on the hardware audio engine.

Syntax

```
HRESULT GetAvailableOffloadConnectorCount(
  LPWSTR _pwstrDeviceId,
  UINT32 _uConnectorId,
  UINT32 *_pAvailableConnectorInstanceCount
);
```

Parameters

`_pwstrDeviceId`

A pointer to the device ID of the hardware audio engine device.

`_uConnectorId`

The identifier for the endpoint connector.

`_pAvailableConnectorInstanceCount`

A pointer to the number of available endpoint connectors that can handle offloaded audio streams.

Return value

The **GetAvailableOffloadConnectorCount** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	audioengineendpoint.h

See also

IHardwareAudioEngineBase

IHardwareAudioEngineBase::GetEngineFormat method

1/11/2020 • 2 minutes to read • Edit Online

The **GetEngineFormat** method retrieves the current data format of the offloaded audio stream.

Syntax

```
HRESULT GetEngineFormat(
  IMMDevice    *pDevice,
  BOOL         _bRequestDeviceFormat,
  WAVEFORMATEX **_ppwfxFormat
);
```

Parameters

`pDevice`

A pointer to an IMMDevice interface.

`_bRequestDeviceFormat`

A Boolean variable that indicates whether or not the **IMMDevice** interface is being accessed to retrieve the device format.

`_ppwfxFormat`

A pointer to a pointer to a WAVEFORMATEX structure that provides information about the hardware audio engine. This includes the waveform audio format type, the number of audio channels, and the sample rate of the audio engine.

Return value

The **GetEngineFormat** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Target Platform	Windows
Header	audioengineendpoint.h

See also

IHardwareAudioEngineBase

IMMDevice

IHardwareAudioEngineBase::GetGfxState method

1/11/2020 • 2 minutes to read • Edit Online

The **GetGfxState** method retrieves the GFX state of the offloaded audio stream.

Syntax

```
HRESULT GetGfxState(
  IMMDevice *pDevice,
  BOOL      *_pbEnable
);
```

Parameters

`pDevice`

Pointer to an IMMDevice interface.

`_pbEnable`

Pointer to a boolean variable.

Return value

The **GetGfxState** method returns S_OK to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Target Platform	Windows
Header	audioengineendpoint.h

See also

IHardwareAudioEngineBase

IMMDevice

IHardwareAudioEngineBase::SetEngineDeviceFormat method

1/11/2020 • 2 minutes to read • Edit Online

The **SetEngineDeviceFormat** method sets the waveform audio format for the hardware audio engine.

Syntax

```
HRESULT SetEngineDeviceFormat(
  IMMDevice    *pDevice,
  WAVEFORMATEX *_pwfxFormat
);
```

Parameters

`pDevice`

A pointer to an IMMDevice interface.

`_pwfxFormat`

A pointer to a WAVEFORMATEX structure that provides information about the hardware audio engine.

Return value

The **SetEngineDeviceFormat** method returns **S_OK** to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Target Platform	Windows
Header	audioengineendpoint.h

See also

IHardwareAudioEngineBase

IMMDevice

IHardwareAudioEngineBase::SetGfxState method

1/11/2020 • 2 minutes to read • Edit Online

The **SetGfxState** method sets the GFX state of the offloaded audio stream.

Syntax

```
HRESULT SetGfxState(
  IMMDevice *pDevice,
  BOOL      _bEnable
);
```

Parameters

`pDevice`

Pointer to an IMMDevice interface.

`_bEnable`

Pointer to a boolean variable.

Return value

The **SetGfxState** method returns S_OK to indicate that it has completed successfully. Otherwise it returns an appropriate error code.

Requirements

Target Platform	Windows
Header	audioengineendpoint.h

See also

IHardwareAudioEngineBase

IMMDevice

audiopolicy.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs audiopolicy.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IAudioSessionControl	The IAudioSessionControl interface enables a client to configure the control parameters for an audio session and to monitor events in the session.
IAudioSessionControl2	The IAudioSessionControl2 interface can be used by a client to get information about the audio session.
IAudioSessionEnumerator	The IAudioSessionEnumerator interface enumerates audio sessions on an audio device.
IAudioSessionEvents	The IAudioSessionEvents interface provides notifications of session-related events such as changes in the volume level, display name, and session state.
IAudioSessionManager	The IAudioSessionManager interface enables a client to access the session controls and volume controls for both cross-process and process-specific audio sessions.
IAudioSessionManager2	The IAudioSessionManager2 interface enables an application to manage submixes for the audio device.
IAudioSessionNotification	The IAudioSessionNotification interface provides notification when an audio session is created.
IAudioVolumeDuckNotification	The IAudioVolumeDuckNotification interface is used to by the system to send notifications about stream attenuation changes.Stream Attenuation, or ducking, is a feature introduced in Windows 7, where the system adjusts the volume of a non-communication stream when a new communication stream is opened. For more information about this feature, see Default Ducking Experience.

IAudioSessionControl interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioSessionControl** interface enables a client to configure the control parameters for an audio session and to monitor events in the session. The IAudioClient::Initialize method initializes a stream object and assigns the stream to an audio session. The client obtains a reference to the **IAudioSessionControl** interface on a stream object by calling the IAudioClient::GetService method with parameter *riid* set to **REFIID** IID_IAudioSessionControl.

Alternatively, a client can obtain the **IAudioSessionControl** interface of an existing session without having to first create a stream object and add the stream to the session. Instead, the client calls the IAudioSessionManager::GetAudioSessionControl method with parameter *AudioSessionGuid* set to the session GUID.

The client can register to receive notification from the session manager when clients change session parameters through the methods in the **IAudioSessionControl** interface.

When releasing an **IAudioSessionControl** interface instance, the client must call the interface's **Release** method from the same thread as the call to **IAudioClient::GetService** that created the object.

The **IAudioSessionControl** interface controls an audio session. An audio session is a collection of shared-mode streams. This interface does not work with exclusive-mode streams.

For a code example that uses the **IAudioSessionControl** interface, see Audio Events for Legacy Audio Applications.

Inheritance

The **IAudioSessionControl** interface inherits from the IUnknown interface. **IAudioSessionControl** also has these types of members:

- Methods

Methods

The **IAudioSessionControl** interface has these methods.

METHOD	DESCRIPTION
IAudioSessionControl::GetDisplayName	The GetDisplayName method retrieves the display name for the audio session.
IAudioSessionControl::GetGroupingParam	The GetGroupingParam method retrieves the grouping parameter of the audio session.
IAudioSessionControl::GetIconPath	The GetIconPath method retrieves the path for the display icon for the audio session.
IAudioSessionControl::GetState	The GetState method retrieves the current state of the audio session.

METHOD	DESCRIPTION
IAudioSessionControl::RegisterAudioSessionNotification	The RegisterAudioSessionNotification method registers the client to receive notifications of session events, including changes in the stream state.
IAudioSessionControl::SetDisplayName	The SetDisplayName method assigns a display name to the current session.
IAudioSessionControl::SetGroupingParam	The SetGroupingParam method assigns a session to a grouping of sessions.
IAudioSessionControl::SetIconPath	The SetIconPath method assigns a display icon to the current session.
IAudioSessionControl::UnregisterAudioSessionNotification	The UnregisterAudioSessionNotification method deletes a previous registration by the client to receive notifications.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

Core Audio Interfaces

IAudioClient::GetService

IAudioClient::Initialize

IAudioSessionManager::GetAudioSessionControl

WASAPI

IAudioSessionControl::GetDisplayName method

1/11/2020 • 2 minutes to read • Edit Online

The **GetDisplayName** method retrieves the display name for the audio session.

Syntax

```
HRESULT GetDisplayName(
  LPWSTR *pRetVal
);
```

Parameters

`pRetVal`

Pointer to a pointer variable into which the method writes the address of a null-terminated, wide-character string that contains the display name. The method allocates the storage for the string. The caller is responsible for freeing the storage, when it is no longer needed, by calling the CoTaskMemFree function. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pRetVal* is **NULL**.
E_OUTOFMEMORY	Out of memory.
AUDCLNT_E_DEVICEINVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

If the client has not called IAudioSessionControl::SetDisplayName to set the display name, the string will be empty. Rather than display an empty name string, the Sndvol program uses a default, automatically generated name to label the volume control for the audio session.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl Interface

IAudioSessionControl::SetDisplayName

IAudioSessionControl::GetGroupingParam method

1/11/2020 • 2 minutes to read • Edit Online

The **GetGroupingParam** method retrieves the grouping parameter of the audio session.

Syntax

```
HRESULT GetGroupingParam(
  GUID *pRetVal
);
```

Parameters

`pRetVal`

Output pointer for the grouping-parameter GUID. This parameter must be a valid, non-**NULL** pointer to a caller-allocated GUID variable. The method writes the grouping parameter into this variable.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P OIN TER	Parameter *pRetVal* is **NULL**.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

All of the audio sessions that have the same grouping parameter value are under the control of the same volume-level slider in the system volume-control program, Sndvol. For more information, see Grouping Parameters.

A client can call the IAudioSessionControl::SetGroupingParam method to change the grouping parameter of a session.

If a client has never called SetGroupingParam to assign a grouping parameter to an audio session, the session's grouping parameter value is GUID_NULL by default and a call to **GetGroupingParam** retrieves this value. A grouping parameter value of GUID_NULL indicates that the session does not belong to any grouping. In that case, the session has its own volume-level slider in the Sndvol program.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl Interface

IAudioSessionControl::SetGroupingParam

IAudioSessionControl::GetIconPath method

1/11/2020 • 2 minutes to read • Edit Online

The **GetIconPath** method retrieves the path for the display icon for the audio session.

Syntax

```
HRESULT GetIconPath(
  LPWSTR *pRetVal
);
```

Parameters

`pRetVal`

Pointer to a pointer variable into which the method writes the address of a null-terminated, wide-character string that specifies the fully qualified path of an .ico, .dll, or .exe file that contains the icon. The method allocates the storage for the string. The caller is responsible for freeing the storage, when it is no longer needed, by calling the CoTaskMemFree function. For information about icon paths and **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pRetVal* is **NULL**.
E_OUTOFMEMORY	Out of memory.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

If a client has not called IAudioSessionControl::SetIconPath to set the display icon, the string will be empty. If no client-specified icon is available, the Sndvol program uses the icon from the client's application window to label the volume control for the audio session.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl Interface

IAudioSessionControl::SetIconPath

IAudioSessionControl::GetState method

1/11/2020 • 2 minutes to read • Edit Online

The **GetState** method retrieves the current state of the audio session.

Syntax

```
HRESULT GetState(
  AudioSessionState *pRetVal
);
```

Parameters

`pRetVal`

Pointer to a variable into which the method writes the current session state. The state must be one of the following AudioSessionState enumeration values:

AudioSessionStateActive

AudioSessionStateInactive

AudioSessionStateExpired

These values indicate that the session state is active, inactive, or expired, respectively. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P OIN TER	Parameter *pRetVal* is **NULL**.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

This method indicates whether the state of the session is active, inactive, or expired. The state is active if the session has one or more streams that are running. The state changes from active to inactive when the last running stream in the session stops. The session state changes to expired when the client destroys the last stream in the session by releasing all references to the stream object.

The Sndvol program displays volume and mute controls for sessions that are in the active and inactive states. When a session expires, Sndvol stops displaying the controls for that session. If a session has previously expired, but the session state changes to active (because a stream in the session begins running) or inactive (because a client assigns a new stream to the session), Sndvol resumes displaying the controls for the session.

The client creates a stream by calling the IAudioClient::Initialize method. At the time that it creates a stream, the client assigns the stream to a session. A session begins when a client assigns the first stream to the session. Initially, the session is in the inactive state. The session state changes to active when the first stream in the session begins running. The session terminates when a client releases the final reference to the last remaining stream object in the session.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioClient::Initialize

IAudioSessionControl Interface

IMMDevice::Activate

IAudioSessionControl::RegisterAudioSessionNotification method

1/11/2020 • 2 minutes to read • Edit Online

The **RegisterAudioSessionNotification** method registers the client to receive notifications of session events, including changes in the stream state.

Syntax

```
HRESULT RegisterAudioSessionNotification(
  IAudioSessionEvents *NewNotifications
);
```

Parameters

`NewNotifications`

Pointer to a client-implemented IAudioSessionEvents interface. If the method succeeds, it calls the AddRef method on the client's **IAudioSessionEvents** interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P OIN TER	Parameter *NewNotifications* is **NULL**.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

This method passes a client-implemented IAudioSessionEvents interface to the session manager. Following a successful call to this method, the session manager calls the methods in the **IAudioSessionEvents** interface to notify the client of various session events. Through these methods, the client receives notifications of the following session-related events:

- Display name changes
- Volume level changes
- Session state changes (inactive to active, or active to inactive)
- Grouping parameter changes
- Disconnection of the client from the session (caused by the user removing the audio endpoint device, shutting down the session manager, or changing the stream format)

When notifications are no longer needed, the client can call the IAudioSessionControl::UnregisterAudioSessionNotification method to terminate the notifications.

Before the client releases its final reference to the IAudioSessionEvents interface, it should call UnregisterAudioSessionNotification to unregister the interface. Otherwise, the application leaks the resources held by the **IAudioSessionEvents** and IAudioSessionControl objects. Note that **RegisterAudioSessionNotification** calls the client's IAudioSessionEvents::AddRef method, and **UnregisterAudioSessionNotification** calls the IAudioSessionEvents::Release method. If the client errs by releasing its reference to the **IAudioSessionEvents** interface before calling **UnregisterAudioSessionNotification**, the session manager never releases its reference to the **IAudioSessionEvents** interface. For example, a poorly designed **IAudioSessionEvents** implementation might call **UnregisterAudioSessionNotification** from the destructor for the **IAudioSessionEvents** object. In this case, the client will not call **UnregisterAudioSessionNotification** until the session manager releases its reference to the **IAudioSessionEvents** interface, and the session manager will not release its reference to the **IAudioSessionEvents** interface until the client calls **UnregisterAudioSessionNotification**. For more information about the **AddRef** and **Release** methods, see the discussion of the IUnknown interface in the Windows SDK documentation.

In addition, the client should call UnregisterAudioSessionNotification before releasing all of its references to the IAudioSessionControl and IAudioSessionManager objects. Unless the client retains a reference to at least one of these two objects, the session manager leaks the storage that it allocated to hold the registration information. After registering a notification interface, the client continues to receive notifications for only as long as at least one of these two objects exists.

For a code example that calls the **RegisterAudioSessionNotification** method, see Audio Events for Legacy Audio Applications.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl Interface

IAudioSessionControl::UnregisterAudioSessionNotification

IAudioSessionEvents Interface

IAudioSessionManager

IAudioSessionControl::SetDisplayName method

1/11/2020 • 2 minutes to read • Edit Online

The **SetDisplayName** method assigns a display name to the current session.

Syntax

```
HRESULT SetDisplayName(
  LPCWSTR Value,
  LPCGUID EventContext
);
```

Parameters

`Value`

Pointer to a null-terminated, wide-character string that contains the display name for the session.

`EventContext`

Pointer to the event-context GUID. If a call to this method generates a name-change event, the session manager sends notifications to all clients that have registered IAudioSessionEvents interfaces with the session manager. The session manager includes the *EventContext* pointer value with each notification. Upon receiving a notification, a client can determine whether it or another client is the source of the event by inspecting the *EventContext* value. This scheme depends on the client selecting a value for this parameter that is unique among all clients in the session. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *Value* is **NULL**.
AUDCLNT_E_DEVICEINVALIDATED **EVICE_INVALIDATED** **NVALIDATED**	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_SERVICE_NOT_RUNNING	The Windows audio service is not running.

Remarks

In Windows Vista, the system-supplied program, Sndvol.exe, uses the display name to label the volume control for the session. If the client does not call **SetDisplayName** to assign a display name to the session, the Sndvol program uses a default, automatically generated name to label the session. The default name incorporates information such as the window title or version resource of the audio application.

If a client has more than one active session, client-specified display names are especially helpful for distinguishing among the volume controls for the various sessions.

In the case of a cross-process session, the session has no identifying information, such as an application name or process ID, from which to generate a default display name. Thus, the client must call **SetDisplayName** to avoid displaying a meaningless default display name.

The display name does not persist beyond the lifetime of the IAudioSessionControl object. Thus, after all references to the object are released, a subsequently created version of the object (with the same application, same session GUID, and same endpoint device) will once again have a default, automatically generated display name until the client calls **SetDisplayName**.

The client can retrieve the display name for the session by calling the IAudioSessionControl::GetDisplayName method.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl Interface

IAudioSessionControl::GetDisplayName

IAudioSessionEvents Interface

IAudioSessionControl::SetGroupingParam method

1/11/2020 • 2 minutes to read • Edit Online

The **SetGroupingParam** method assigns a session to a grouping of sessions.

Syntax

```
HRESULT SetGroupingParam(
  LPCGUID Override,
  LPCGUID EventContext
);
```

Parameters

`Override`

The new grouping parameter. This parameter must be a valid, non-**NULL** pointer to a grouping-parameter GUID. For more information, see Remarks.

`EventContext`

Pointer to the event-context GUID. If a call to this method generates a grouping-change event, the session manager sends notifications to all clients that have registered IAudioSessionEvents interfaces with the session manager. The session manager includes the *EventContext* pointer value with each notification. Upon receiving a notification, a client can determine whether it or another client is the source of the event by inspecting the *EventContext* value. This scheme depends on the client selecting a value for this parameter that is unique among all clients in the session. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *Grouping* is **NULL**.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.

Remarks

A client calls this method to change the grouping parameter of a session. All of the audio sessions that have the same grouping parameter value are under the control of the same volume-level slider in the system volume-control program, Sndvol. For more information, see Grouping Parameters.

The client can get the current grouping parameter for the session by calling the IAudioSessionControl::GetGroupingParam method.

If a client has never called **SetGroupingParam** to assign a grouping parameter to a session, the session does not belong to any grouping. A session that does not belong to any grouping has its own, dedicated volume-level slider in the Sndvol program.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl Interface

IAudioSessionControl::GetGroupingParam

IAudioSessionEvents Interface

IAudioSessionControl::SetIconPath method

1/11/2020 • 2 minutes to read • Edit Online

The **SetIconPath** method assigns a display icon to the current session.

Syntax

```
HRESULT SetIconPath(
  LPCWSTR Value,
  LPCGUID EventContext
);
```

Parameters

`Value`

Pointer to a null-terminated, wide-character string that specifies the path and file name of an .ico, .dll, or .exe file that contains the icon. For information about icon paths, see the Windows SDK documentation.

`EventContext`

Pointer to the event-context GUID. If a call to this method generates an icon-change event, the session manager sends notifications to all clients that have registered IAudioSessionEvents interfaces with the session manager. The session manager includes the *EventContext* pointer value with each notification. Upon receiving a notification, a client can determine whether it or another client is the source of the event by inspecting the *EventContext* value. This scheme depends on the client selecting a value for this parameter that is unique among all clients in the session. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *Value* is **NULL**.

AU **DCL** **NT_** **E_D** **EVI** **CE_I** **NV** **ALI** **DAT** **ED**	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU **DCL** **NT_** **E_SE** **RVI** **CE_** **NO** **T_R** **UN** **NIN** **G**	The Windows audio service is not running.

Remarks

In Windows Vista, the system-supplied program, Sndvol.exe, uses the display icon (along with the display name) to label the volume control for the session. If the client does not call **SetIconPath** to assign an icon to the session, the Sndvol program uses the icon from the application window as the default icon for the session.

In the case of a cross-process session, the session is not associated with a single application process. Thus, Sndvol has no application-specific icon to use by default, and the client must call **SetIconPath** to avoid displaying a meaningless icon.

The display icon does not persist beyond the lifetime of the IAudioSessionControl object. Thus, after all references to the object are released, a subsequently created version of the object (with the same application, same session GUID, and same endpoint device) will once again have a default icon until the client calls **SetIconPath**.

The client can retrieve the display icon for the session by calling the IAudioSessionControl::GetIconPath method.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl Interface

IAudioSessionControl::UnregisterAudioSessionNotification method

1/11/2020 • 2 minutes to read • Edit Online

The **UnregisterAudioSessionNotification** method deletes a previous registration by the client to receive notifications.

Syntax

```
HRESULT UnregisterAudioSessionNotification(
  IAudioSessionEvents *NewNotifications
);
```

Parameters

`NewNotifications`

Pointer to a client-implemented IAudioSessionEvents interface. The client passed this same interface pointer to the session manager in a previous call to the IAudioSessionControl::RegisterAudioSessionNotification method. If the **UnregisterAudioSessionNotification** method succeeds, it calls the Release method on the client's **IAudioSessionEvents** interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *NewNotifications* is **NULL**.
E_NOTFOUND	The specified interface was not previously registered by the client or has already been removed.

Remarks

The client calls this method when it no longer needs to receive notifications. The **UnregisterAudioSessionNotification** method removes the registration of an IAudioSessionEvents interface that the client previously registered with the session manager by calling the IAudioSessionControl::RegisterAudioSessionNotification method.

Before the client releases its final reference to the IAudioSessionEvents interface, it should call **UnregisterAudioSessionNotification** to unregister the interface. Otherwise, the application leaks the resources held by the **IAudioSessionEvents** and IAudioSessionControl objects. Note that RegisterAudioSessionNotification

calls the client's IAudioSessionEvents::AddRef method, and **UnregisterAudioSessionNotification** calls the IAudioSessionEvents::Release method. If the client errs by releasing its reference to the **IAudioSessionEvents** interface before calling **UnregisterAudioSessionNotification**, the session manager never releases its reference to the **IAudioSessionEvents** interface. For example, a poorly designed **IAudioSessionEvents** implementation might call **UnregisterAudioSessionNotification** from the destructor for the **IAudioSessionEvents** object. In this case, the client will not call **UnregisterAudioSessionNotification** until the session manager releases its reference to the **IAudioSessionEvents** interface, and the session manager will not release its reference to the **IAudioSessionEvents** interface until the client calls **UnregisterAudioSessionNotification**. For more information about the **AddRef** and **Release** methods, see the discussion of the IUnknown interface in the Windows SDK documentation.

For a code example that calls the **UnregisterAudioSessionNotification** method, see Audio Events for Legacy Audio Applications.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl Interface

IAudioSessionControl::RegisterAudioSessionNotification

IAudioSessionEvents Interface

IAudioSessionControl2 interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioSessionControl2** interface can be used by a client to get information about the audio session.

To get a reference to the **IAudioSessionControl2** interface, the application must call **IAudioSessionControl::QueryInterface** to request the interface pointer from the stream object's IAudioSessionControl interface. There are two ways an application can get a pointer to the **IAudioSessionControl** interface:

- By calling IAudioClient::GetService on the audio client after opening a stream on the device. The audio client opens a stream for rendering or capturing, and associates it with an audio session by calling IAudioClient::Initialize.
- By calling IAudioSessionManager::GetAudioSessionControl for an existing audio session without opening the stream.

When the application wants to release the **IAudioSessionControl2** interface instance, the application must call the interface's **Release** method from the same thread as the call to IAudioClient::GetService that created the object.

The application thread that uses this interface must be initialized for COM. For more information about COM initialization, see the description of the **CoInitializeEx** function in the Windows SDK documentation.

Inheritance

The **IAudioSessionControl2** interface inherits from IAudioSessionControl. **IAudioSessionControl2** also has these types of members:

- Methods

Methods

The **IAudioSessionControl2** interface has these methods.

METHOD	DESCRIPTION
IAudioSessionControl2::GetProcessId	The GetProcessId method retrieves the process identifier of the audio session.
IAudioSessionControl2::GetSessionIdentifier	The GetSessionIdentifier method retrieves the audio session identifier.
IAudioSessionControl2::GetSessionInstanceIdentifier	The GetSessionInstanceIdentifier method retrieves the identifier of the audio session instance.
IAudioSessionControl2::IsSystemSoundsSession	The IsSystemSoundsSession method indicates whether the session is a system sounds session.
IAudioSessionControl2::SetDuckingPreference	The SetDuckingPreference method enables or disables the default stream attenuation experience (auto-ducking) provided by the system.

Remarks

This interface supports custom implementations for *stream attenuation* or *ducking*, a new feature in Windows 7. An application playing a media stream can make it behave differently when a new communication stream is opened on the default communication device. For example, the original media stream can be paused while the new communication stream is open. For more information about this feature, see Default Ducking Experience.

An application can use this interface to perform the following tasks:

- Specify that it wants to opt out of the default stream attenuation experience provided by the system.
- Get the audio session identifier that is associated with the stream. The identifier is required during the notification registration. The application can register itself to receive ducking notifications from the system.
- Check whether the stream associated with the audio session is a system sound.

Examples

The following example code shows how to get a reference to the **IAudioSessionControl2** interface and call its methods to determine whether the stream associated with the audio session is a system sound.

```
HRESULT SetDuckingForSystemSounds()
{

    HRESULT hr = S_OK;

    IMMDevice* pDevice = NULL;
    IMMDeviceEnumerator* pEnumerator = NULL;
    IAudioSessionControl* pSessionControl = NULL;
    IAudioSessionControl2* pSessionControl2 = NULL;
    IAudioSessionManager* pSessionManager = NULL;

    CHECK_HR( hr = CoInitialize(NULL));

    // Create the device enumerator.
    CHECK_HR( hr = CoCreateInstance(
        __uuidof(MMDeviceEnumerator),
        NULL, CLSCTX_ALL,
        __uuidof(IMMDeviceEnumerator),
        (void**)&pEnumerator));

    // Get the default audio device.
    CHECK_HR( hr = pEnumerator->GetDefaultAudioEndpoint(
                    eRender, eConsole, &pDevice));

    // Get the audio client.
    CHECK_HR( hr = pDevice->Activate(
        __uuidof(IID_IAudioSessionManager), CLSCTX_ALL,
        NULL, (void**)&pSessionManager));

    // Get a reference to the session manager.
    CHECK_HR( hr = pSessionManager->GetAudioSessionControl (GUID_NULL, FALSE, &pSessionControl));

    // Get the extended session control interface pointer.
    CHECK_HR( hr = pSessionControl->QueryInterface(
        __uuidof(IAudioSessionControl2), (void**) &pSessionControl2));

    // Check whether this is a system sound.
    CHECK_HR( hr = pSessionControl2->IsSystemSoundsSession());

    // If it is a system sound, opt out of the default
    // stream attenuation experience.
    CHECK_HR( hr = pSessionControl2->SetDuckingPreference(TRUE));

done:

    // Clean up.
    SAFE_RELEASE(pSessionControl2);
    SAFE_RELEASE(pSessionControl);
    SAFE_RELEASE(pEnumerator);
    SAFE_RELEASE(pDevice);


    return hr;
}
```

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]

Target Platform	Windows
Header	audiopolicy.h

See also

Core Audio Interfaces

IAudioSessionControl

Using a Communication Device

IAudioSessionControl2::GetProcessId method

1/11/2020 • 2 minutes to read • Edit Online

The **GetProcessId** method retrieves the process identifier of the audio session.

Syntax

```
HRESULT GetProcessId(
  DWORD *pRetVal
);
```

Parameters

`pRetVal`

Pointer to a **DWORD** variable that receives the process identifier of the audio session.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN VALUE	DESCRIPTION
E_P OIN TER	*pRetVal* is **NULL**.
AU DCL NT_ S_N O_SI NGL E_P ROC ESS	The session spans more than one process. In this case, *pRetVal* receives the initial identifier of the process that created the session. To use this value , include the following definition: `#define AUDCLNT_S_NO_SINGLE_PROCESS AUDCLNT_SUCCESS (0x00d)`
AU DCL NT_ E_D EVI CE_I NVA LID ATE D	The audio session is disconnected on the default audio device.

Remarks

This method overwrites the value that was passed by the application in *pRetVal*.

GetProcessId checks whether the audio session has been disconnected on the default device or if the session has switched to another stream. In the case of stream switching, this method transfers state information for the new stream to the session. State information includes volume controls, metadata information (display name, icon path), and the session's property store.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl2

IAudioSessionControl2::GetSessionIdentifier method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSessionIdentifier** method retrieves the audio session identifier.

Syntax

```
HRESULT GetSessionIdentifier(
  LPWSTR *pRetVal
);
```

Parameters

`pRetVal`

Pointer to the address of a null-terminated, wide-character string that receives the audio session identifier. The string is allocated by this method and must be released by the caller by calling **CoTaskMemFree**. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN VALUE	DESCRIPTION
E_P OIN TER	*pRetVal* is **NULL**.
AU DCL NT_ E_D EVI CE_I NVA LID ATE D	The audio session is disconnected on the default audio device.

Remarks

Each audio session is identified by an identifier string. This session identifier string is not unique across all instances. If there are two instances of the application playing, both instances will have the same session identifier. The identifier retrieved by **GetSessionIdentifier** is different from the session instance identifier, which is unique across all sessions. To get the session instance identifier, call IAudioSessionControl2::GetSessionInstanceIdentifier.

GetSessionIdentifier checks whether the session has been disconnected on the default device. It retrieves the

identifier string that is cached by the audio client for the device. If the session identifier is not found, this method retrieves it from the audio engine.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl2

IAudioSessionControl2::GetSessionInstanceIdentifier method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSessionInstanceIdentifier** method retrieves the identifier of the audio session instance.

Syntax

```
HRESULT GetSessionInstanceIdentifier(
  LPWSTR *pRetVal
);
```

Parameters

`pRetVal`

Pointer to the address of a null-terminated, wide-character string that receives the identifier of a particular instance of the audio session. The string is allocated by this method and must be released by the caller by calling **CoTaskMemFree**. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN VALUE	DESCRIPTION
E_P OIN TER	*pRetVal* is **NULL**.
AU DCL NT_ E_D EVI CE_I NVA LID ATE D	The audio session is disconnected on the default audio device.

Remarks

Each audio session instance is identified by a unique string. This string represents a particular instance of the audio session and, unlike the session identifier, is unique across all instances. If there are two instances of the application playing, they will have different session instance identifiers. The identifier retrieved by **GetSessionInstanceIdentifier** is different from the session identifier, which is shared by all session instances. To

get the session identifier, call IAudioSessionControl2::GetSessionIdentifier.

GetSessionInstanceIdentifier checks whether the session has been disconnected on the default device. It retrieves the identifier string that is cached by the audio client for the device. If the session instance identifier is not found, this method retrieves it from the audio engine. For example code about getting a session instance identifier, see Getting Ducking Events from a Communication Device.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl2

Using a Communication Device

IAudioSessionControl2::IsSystemSoundsSession method

1/11/2020 • 2 minutes to read • Edit Online

The **IsSystemSoundsSession** method indicates whether the session is a system sounds session.

Syntax

```
HRESULT IsSystemSoundsSession();
```

Parameters

This method has no parameters.

Return value

The possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
S_OK	The session is a system sounds session.
S_FALSE	The session is not a system sounds session.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl2

IAudioSessionControl2::SetDuckingPreference method

1/11/2020 • 2 minutes to read • Edit Online

The **SetDuckingPreference** method enables or disables the default stream attenuation experience (auto-ducking) provided by the system.

Syntax

```
HRESULT SetDuckingPreference(
  BOOL optOut
);
```

Parameters

`optOut`

A **BOOL** variable that enables or disables system auto-ducking.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN VALUE	DESCRIPTION
AU DCL NT_ E_D EVI CE_I NVA LID ATE D	The audio session is disconnected on the default audio device.

Remarks

By default, the system adjusts the volume for all currently playing sounds when the system starts a communication session and receives a new communication stream on the default communication device. For more information about this feature, see Using a Communication Device.

If the application passes **TRUE** in *optOut*, the system disables the Default Ducking Experience. For more information, see Disabling the Default Ducking Experience.

To provide a custom implementation, the application needs to get notifications from the system when it opens or closes the communication stream. To receive the notifications, the application must call this method before registering itself by calling **IAudioSessionManager2::RegisterForDuckNotification**. For more information and

example code, see Getting Ducking Events.

If the application passes **FALSE** in *optOut*, the application provides the default stream attenuation experience provided by the system.

We recommend that the application call **SetDuckingPreference** during stream creation. However, this method can be called dynamically during the session to change the initial preference.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl2

IAudioSessionEnumerator interface

1/23/2020 • 3 minutes to read • Edit Online

The **IAudioSessionEnumerator** interface enumerates audio sessions on an audio device. To get a reference to the **IAudioSessionEnumerator** interface of the session enumerator object, the application must call IAudioSessionManager2::GetSessionEnumerator.

Inheritance

The **IAudioSessionEnumerator** interface inherits from the IUnknown interface. **IAudioSessionEnumerator** also has these types of members:

- Methods

Methods

The **IAudioSessionEnumerator** interface has these methods.

METHOD	DESCRIPTION
IAudioSessionEnumerator::GetCount	The GetCount method gets the total number of audio sessions that are open on the audio device.
IAudioSessionEnumerator::GetSession	The GetSession method gets the audio session specified by an audio session number.

Remarks

If an application wants to be notified when new sessions are created, it must register its implementation of IAudioSessionNotification with the session manager. Upon successful registration, the session manager sends create-session notifications to the application in the form of callbacks. These notifications contain a reference to the IAudioSessionControl pointer of the newly created session.

The session enumerator maintains a list of current sessions by holding references to each session's IAudioSessionControl pointer. However, the session enumerator might not be aware of the new sessions that are reported through IAudioSessionNotification. In that case, the application would have access to only a partial list of sessions. This might occur if the **IAudioSessionControl** pointer (in the callback) is released before the session enumerator is initialized. Therefore, if an application wants a complete set of sessions for the audio endpoint, the application should maintain its own list.

The application must perform the following steps to receive session notifications and manage a list of current sessions.

1. Initialize COM with the Multithreaded Apartment (MTA) model by calling `CoInitializeEx(NULL, COINIT_MULTITHREADED)` in a non-UI thread. If MTA is not initialized, the application does not receive session notifications from the session manager.

   > **Note** Threads that run the user interface of an application should be initialized with the apartment threading model.

2. Activate an IAudioSessionManager2 interface from the audio endpoint device. Call IMMDevice::Activate with

parameter *iid* set to **IID_IAudioSessionManager2**. This call receives a reference to the session manager's **IAudioSessionManager2** interface in the *ppInterface* parameter.

3. Implement the IAudioSessionNotification interface to provide the callback behavior.
4. Call IAudioSessionManager2::RegisterSessionNotification to register the application's implementation of IAudioSessionNotification.
5. Create and initialize the session enumerator object by calling IAudioSessionManager2::GetSessionEnumerator. This method generates a list of current sessions available for the endpoint and adds the IAudioSessionControl pointers for each session in the list, if they are not already present.
6. Use the **IAudioSessionEnumerator** interface returned in the previous step to retrieve and enumerate the list of sessions. The session control for each session can be retrieved by calling IAudioSessionEnumerator::GetSession. Make sure you call **AddRef** for each session control to maintain the reference count.
7. When the application gets a create-session notification, add the IAudioSessionControl pointer of the new session (received in IAudioSessionNotification::OnSessionCreated) to the list of existing sessions.

Because the application maintains this list of sessions and manages the lifetime of the session based on the application's requirements, there is no expiration mechanism enforced by the audio system on the session control objects.

A session control is valid as long as the application has a reference to the session control in the list.

Examples

The following example code shows how to create the session enumerator object and then enumerate sessions.

```
HRESULT EnumSessions(IAudioSessionManager2* pSessionManager)
{
    if (!pSessionManager)
    {
        return E_INVALIDARG;
    }

    HRESULT hr = S_OK;

    int cbSessionCount = 0;
    LPWSTR pswSession = NULL;

    IAudioSessionEnumerator* pSessionList = NULL;
    IAudioSessionControl* pSessionControl = NULL;

    // Get the current list of sessions.
    CHECK_HR( hr = pSessionManager->GetSessionEnumerator(&pSessionList));

    // Get the session count.
    CHECK_HR( hr = pSessionList->GetCount(&cbSessionCount));

    for (int index = 0 ; index < cbSessionCount ; index++)
    {
        CoTaskMemFree(pswSession);
        SAFE_RELEASE(pSessionControl);

        // Get the <n>th session.
        CHECK_HR(hr = pSessionList->GetSession(index, &pSessionControl));

        CHECK_HR(hr = pSessionControl->GetDisplayName(&pswSession));

        wprintf_s(L"Session Name: %s\n", pswSession);
    }

done:
    CoTaskMemFree(pswSession);
    SAFE_RELEASE(pSessionControl);
    SAFE_RELEASE(pSessionList);

    return hr;

}
```

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

Core Audio Interfaces

IAudioSessionEnumerator::GetCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetCount** method gets the total number of audio sessions that are open on the audio device.

Syntax

```
HRESULT GetCount(
  int *SessionCount
);
```

Parameters

`SessionCount`

Receives the total number of audio sessions.

Return value

If the method succeeds, it returns S_OK.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionEnumerator

IAudioSessionEnumerator::GetSession method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSession** method gets the audio session specified by an audio session number.

Syntax

```
HRESULT GetSession(
  int                 SessionCount,
  IAudioSessionControl **Session
);
```

Parameters

`SessionCount`

The session number. If there are *n* sessions, the sessions are numbered from 0 to *n* – 1. To get the number of sessions, call the IAudioSessionEnumerator::GetCount method.

`Session`

Receives a pointer to the IAudioSessionControl interface of the session object in the collection that is maintained by the session enumerator. The caller must release the interface pointer.

Return value

If the method succeeds, it returns S_OK.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionEnumerator

IAudioSessionEvents interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioSessionEvents** interface provides notifications of session-related events such as changes in the volume level, display name, and session state. Unlike the other interfaces in this section, which are implemented by the WASAPI system component, a WASAPI client implements the **IAudioSessionEvents** interface. To receive event notifications, the client passes a pointer to its **IAudioSessionEvents** interface to the IAudioSessionControl::RegisterAudioSessionNotification method.

After registering its **IAudioClientSessionEvents** interface, the client receives event notifications in the form of callbacks through the methods in the interface.

In implementing the **IAudioSessionEvents** interface, the client should observe these rules to avoid deadlocks and undefined behavior:

- The methods in the interface must be nonblocking. The client should never wait on a synchronization object during an event callback.
- The client should never call the IAudioSessionControl::UnregisterAudioSessionNotification method during an event callback.
- The client should never release the final reference on a WASAPI object during an event callback.

For a code example that implements an **IAudioSessionEvents** interface, see Audio Session Events. For a code example that registers a client's **IAudioSessionEvents** interface to receive notifications, see Audio Events for Legacy Audio Applications.

Inheritance

The **IAudioSessionEvents** interface inherits from the IUnknown interface. **IAudioSessionEvents** also has these types of members:

- Methods

Methods

The **IAudioSessionEvents** interface has these methods.

METHOD	DESCRIPTION
IAudioSessionEvents::OnChannelVolumeChanged	The OnChannelVolumeChanged method notifies the client that the volume level of an audio channel in the session submix has changed.
IAudioSessionEvents::OnDisplayNameChanged	The OnDisplayNameChanged method notifies the client that the display name for the session has changed.
IAudioSessionEvents::OnGroupingParamChanged	The OnGroupingParamChanged method notifies the client that the grouping parameter for the session has changed.
IAudioSessionEvents::OnIconPathChanged	The OnIconPathChanged method notifies the client that the display icon for the session has changed.

METHOD	DESCRIPTION
IAudioSessionEvents::OnSessionDisconnected	The OnSessionDisconnected method notifies the client that the audio session has been disconnected.
IAudioSessionEvents::OnSimpleVolumeChanged	The OnSimpleVolumeChanged method notifies the client that the volume level or muting state of the audio session has changed.
IAudioSessionEvents::OnStateChanged	The OnStateChanged method notifies the client that the stream-activity state of the session has changed.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

Core Audio Interfaces

IAudioSessionControl::RegisterAudioSessionNotification

IAudioSessionControl::UnregisterAudioSessionNotification

WASAPI

IAudioSessionEvents::OnChannelVolumeChanged method

1/11/2020 • 2 minutes to read • Edit Online

The **OnChannelVolumeChanged** method notifies the client that the volume level of an audio channel in the session submix has changed.

Syntax

```
HRESULT OnChannelVolumeChanged(
  DWORD    ChannelCount,
  float [] NewChannelVolumeArray,
  DWORD    ChangedChannel,
  LPCGUID  EventContext
);
```

Parameters

`ChannelCount`

The channel count. This parameter specifies the number of audio channels in the session submix.

`NewChannelVolumeArray`

Pointer to an array of volume levels. Each element is a value of type **float** that specifies the volume level for a particular channel. Each volume level is a value in the range 0.0 to 1.0, where 0.0 is silence and 1.0 is full volume (no attenuation). The number of elements in the array is specified by the *ChannelCount* parameter. If an audio stream contains *n* channels, the channels are numbered from 0 to *n*– 1. The array element whose index matches the channel number, contains the volume level for that channel. Assume that the array remains valid only for the duration of the call.

`ChangedChannel`

The number of the channel whose volume level changed. Use this value as an index into the *NewChannelVolumeArray* array. If the session submix contains *n* channels, the channels are numbered from 0 to *n* – 1. If more than one channel might have changed (for example, as a result of a call to the IChannelAudioVolume::SetAllVolumes method), the value of *ChangedChannel* is (**DWORD**)(–1).

`EventContext`

The event context value. This is the same value that the caller passed to the IChannelAudioVolume::SetChannelVolume or **IChannelAudioVolume::SetAllVolumes** method in the call that initiated the change in volume level of the channel. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

The session manager calls this method each time a call to the **IChannelAudioVolume::SetChannelVolume** or

IChannelAudioVolume::SetAllVolumes method successfully updates the volume level of one or more channels in the session submix. Note that the **OnChannelVolumeChanged** call occurs regardless of whether the new channel volume level or levels differ in value from the previous channel volume level or levels.

The *EventContext* parameter provides a means for a client to distinguish between a channel-volume change that it initiated and one that some other client initiated. When calling the **IChannelAudioVolume::SetChannelVolume** or **IChannelAudioVolume::SetAllVolumes** method, a client passes in an *EventContext* parameter value that its implementation of the **OnChannelVolumeChanged** method can recognize.

For a code example that implements the methods in the **IAudioSessionEvents** interface, see Audio Session Events.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionEvents Interface

IChannelAudioVolume::SetAllVolumes

IChannelAudioVolume::SetChannelVolume

IAudioSessionEvents::OnDisplayNameChanged method

1/11/2020 • 2 minutes to read • Edit Online

The **OnDisplayNameChanged** method notifies the client that the display name for the session has changed.

Syntax

```
HRESULT OnDisplayNameChanged(
  LPCWSTR NewDisplayName,
  LPCGUID EventContext
);
```

Parameters

`NewDisplayName`

The new display name for the session. This parameter points to a null-terminated, wide-character string containing the new display name. The string remains valid for the duration of the call.

`EventContext`

The event context value. This is the same value that the caller passed to IAudioSessionControl::SetDisplayName in the call that changed the display name for the session. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

The session manager calls this method each time a call to the IAudioSessionControl::SetDisplayName method changes the display name of the session. The Sndvol program uses a session's display name to label the volume slider for the session.

The *EventContext* parameter provides a means for a client to distinguish between a display-name change that it initiated and one that some other client initiated. When calling the IAudioSessionControl::SetDisplayName method, a client passes in an *EventContext* parameter value that its implementation of the **OnDisplayNameChanged** method can recognize.

For a code example that implements the methods in the IAudioSessionEvents interface, see Audio Session Events.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]

Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl::SetDisplayName

IAudioSessionEvents Interface

IAudioSessionEvents::OnGroupingParamChanged method

1/11/2020 • 2 minutes to read • Edit Online

The **OnGroupingParamChanged** method notifies the client that the grouping parameter for the session has changed.

Syntax

```
HRESULT OnGroupingParamChanged(
  LPCGUID NewGroupingParam,
  LPCGUID EventContext
);
```

Parameters

`NewGroupingParam`

The new grouping parameter for the session. This parameter points to a grouping-parameter GUID.

`EventContext`

The event context value. This is the same value that the caller passed to IAudioSessionControl::SetGroupingParam in the call that changed the grouping parameter for the session. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

The session manager calls this method each time a call to the IAudioSessionControl::SetGroupingParam method changes the grouping parameter for the session.

The *EventContext* parameter provides a means for a client to distinguish between a grouping-parameter change that it initiated and one that some other client initiated. When calling the IAudioSessionControl::SetGroupingParam method, a client passes in an *EventContext* parameter value that its implementation of the **OnGroupingParamChanged** method can recognize.

For a code example that implements the methods in the IAudioSessionEvents interface, see Audio Session Events.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]

Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl::SetGroupingParam

IAudioSessionEvents Interface

IAudioSessionEvents::OnIconPathChanged method

1/11/2020 • 2 minutes to read • Edit Online

The **OnIconPathChanged** method notifies the client that the display icon for the session has changed.

Syntax

```
HRESULT OnIconPathChanged(
  LPCWSTR NewIconPath,
  LPCGUID EventContext
);
```

Parameters

`NewIconPath`

The path for the new display icon for the session. This parameter points to a string that contains the path for the new icon. The string pointer remains valid only for the duration of the call.

`EventContext`

The event context value. This is the same value that the caller passed to IAudioSessionControl::SetIconPath in the call that changed the display icon for the session. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

The session manager calls this method each time a call to the IAudioSessionControl::SetIconPath method changes the display icon for the session. The Sndvol program uses a session's display icon to label the volume slider for the session.

The *EventContext* parameter provides a means for a client to distinguish between a display-icon change that it initiated and one that some other client initiated. When calling the IAudioSessionControl::SetIconPath method, a client passes in an *EventContext* parameter value that its implementation of the **OnIconPathChanged** method can recognize.

For a code example that implements the methods in the IAudioSessionEvents interface, see Audio Session Events.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows

Header	audiopolicy.h

See also

IAudioSessionControl::SetIconPath

IAudioSessionEvents Interface

IAudioSessionEvents::OnSessionDisconnected method

1/11/2020 • 2 minutes to read • Edit Online

The **OnSessionDisconnected** method notifies the client that the audio session has been disconnected.

Syntax

```
HRESULT OnSessionDisconnected(
  AudioSessionDisconnectReason DisconnectReason
);
```

Parameters

`DisconnectReason`

The reason that the audio session was disconnected. The caller sets this parameter to one of the **AudioSessionDisconnectReason** enumeration values shown in the following table.

VALUE	DESCRIPTION
DisconnectReasonDeviceRemoval	The user removed the audio endpoint device.
DisconnectReasonServerShutdown	The Windows audio service has stopped.
DisconnectReasonFormatChanged	The stream format changed for the device that the audio session is connected to.
DisconnectReasonSessionLogoff	The user logged off the Windows Terminal Services (WTS) session that the audio session was running in.
DisconnectReasonSessionDisconnected	The WTS session that the audio session was running in was disconnected.
DisconnectReasonExclusiveModeOverride	The (shared-mode) audio session was disconnected to make the audio endpoint device available for an exclusive-mode connection.

For more information about WTS sessions, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

When disconnecting a session, the session manager closes the streams that belong to that session and invalidates all outstanding requests for services on those streams. The client should respond to a disconnection by releasing

all of its references to the IAudioClient interface for a closed stream and releasing all references to the service interfaces that it obtained previously through calls to the IAudioClient::GetService method.

Following disconnection, many of the methods in the WASAPI interfaces that are tied to closed streams in the disconnected session return error code AUDCLNT_E_DEVICE_INVALIDATED (for example, see IAudioClient::GetCurrentPadding). For information about recovering from this error, see Recovering from an Invalid-Device Error.

If the Windows audio service terminates unexpectedly, it does not have an opportunity to notify clients that it is shutting down. In that case, clients learn that the service has stopped when they call a method such as IAudioClient::GetCurrentPadding that discovers that the service is no longer running and fails with error code AUDCLNT_E_SERVICE_NOT_RUNNING.

A client cannot generate a session-disconnected event. The system is always the source of this type of event. Thus, unlike some other IAudioSessionEvents methods, this method does not have a context parameter.

For a code example that implements the methods in the IAudioSessionEvents interface, see Audio Session Events.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioClient Interface

IAudioClient::GetService

IAudioSessionEvents Interface

IAudioSessionEvents::OnSimpleVolumeChanged method

1/11/2020 • 2 minutes to read • Edit Online

The **OnSimpleVolumeChanged** method notifies the client that the volume level or muting state of the audio session has changed.

Syntax

```
HRESULT OnSimpleVolumeChanged(
  float   NewVolume,
  BOOL    NewMute,
  LPCGUID EventContext
);
```

Parameters

`NewVolume`

The new volume level for the audio session. This parameter is a value in the range 0.0 to 1.0, where 0.0 is silence and 1.0 is full volume (no attenuation).

`NewMute`

The new muting state. If **TRUE**, muting is enabled. If **FALSE**, muting is disabled.

`EventContext`

The event context value. This is the same value that the caller passed to ISimpleAudioVolume::SetMasterVolume or ISimpleAudioVolume::SetMute in the call that changed the volume level or muting state of the session. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

The session manager calls this method each time a call to the ISimpleAudioVolume::SetMasterVolume or ISimpleAudioVolume::SetMute method changes the volume level or muting state of the session.

The *EventContext* parameter provides a means for a client to distinguish between a volume or mute change that it initiated and one that some other client initiated. When calling the ISimpleAudioVolume::SetMasterVolume or ISimpleAudioVolume::SetMute method, a client passes in an *EventContext* parameter value that its implementation of the **OnSimpleVolumeChanged** method can recognize.

For a code example that implements the methods in the IAudioSessionEvents interface, see Audio Session Events.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionEvents Interface

ISimpleAudioVolume::SetMasterVolume

ISimpleAudioVolume::SetMute

IAudioSessionEvents::OnStateChanged method

1/11/2020 • 2 minutes to read • Edit Online

The **OnStateChanged** method notifies the client that the stream-activity state of the session has changed.

Syntax

```
HRESULT OnStateChanged(
  AudioSessionState NewState
);
```

Parameters

`NewState`

The new session state. The value of this parameter is one of the following AudioSessionState enumeration values:

AudioSessionStateActive

AudioSessionStateInactive

AudioSessionStateExpired

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

A client cannot generate a session-state-change event. The system is always the source of this type of event. Thus, unlike some other IAudioSessionEvents methods, this method does not supply a context parameter.

The system changes the state of a session from inactive to active at the time that a client opens the first stream in the session. A client opens a stream by calling the IAudioClient::Initialize method. The system changes the session state from active to inactive at the time that a client closes the last stream in the session. The client that releases the last reference to an IAudioClient object closes the stream that is associated with the object.

For a code example that implements the methods in the IAudioSessionEvents interface, see Audio Session Events.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioClient Interface

IAudioClient::Initialize

IAudioSessionEvents Interface

IAudioSessionManager interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioSessionManager** interface enables a client to access the session controls and volume controls for both cross-process and process-specific audio sessions. The client obtains a reference to an **IAudioSessionManager** interface by calling the IMMDevice::Activate method with parameter *iid* set to **REFIID** IID_IAudioSessionManager.

This interface enables clients to access the controls for an existing session without first opening a stream. This capability is useful for clients of higher-level APIs that are built on top of WASAPI and use session controls internally but do not give their clients access to session controls.

In Windows Vista, the higher-level APIs that use WASAPI include Media Foundation, DirectSound, the Windows multimedia **waveInXxx**, **waveOutXxx**, and **mciXxx** functions, and third-party APIs.

When a client creates an audio stream through a higher-level API, that API typically adds the stream to the default audio session for the client's process (the session that is identified by the session GUID value, GUID_NULL), but the same API might not provide a means for the client to access the controls for that session. In that case, the client can access the controls through the **IAudioSessionManager** interface.

For a code example that uses the **IAudioSessionManager** interface, see Audio Events for Legacy Audio Applications.

Inheritance

The **IAudioSessionManager** interface inherits from the IUnknown interface. **IAudioSessionManager** also has these types of members:

- Methods

Methods

The **IAudioSessionManager** interface has these methods.

METHOD	DESCRIPTION
IAudioSessionManager::GetAudioSessionControl	The GetAudioSessionControl method retrieves an audio session control.
IAudioSessionManager::GetSimpleAudioVolume	The GetSimpleAudioVolume method retrieves a simple audio volume control.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Header	audiopolicy.h

See also

Core Audio Interfaces

IMMDevice::Activate

WASAPI

IAudioSessionManager::GetAudioSessionControl method

1/11/2020 • 2 minutes to read • Edit Online

The **GetAudioSessionControl** method retrieves an audio session control.

Syntax

```
HRESULT GetAudioSessionControl(
  LPCGUID              AudioSessionGuid,
  DWORD                StreamFlags,
  IAudioSessionControl **SessionControl
);
```

Parameters

`AudioSessionGuid`

Pointer to a session GUID. If the GUID does not identify a session that has been previously opened, the call opens a new but empty session. The Sndvol program does not display a volume-level control for a session unless it contains one or more active streams. If this parameter is **NULL** or points to the value GUID_NULL, the method assigns the stream to the default session.

`StreamFlags`

Specifies the status of the flags for the audio stream.

`SessionControl`

Pointer to a pointer variable into which the method writes a pointer to the IAudioSessionControl interface of the audio session control object. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the call fails, **SessionControl* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU **DCL** **NT_** **E_N** **OT_** **INIT** **IALI** **ZED**	The audio stream has not been successfully initialized.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.
E_P OIN TER	Parameter *SessionControl* is **NULL**.
E_M EM OR Y	Out of memory.

Remarks

For a code example that calls this method, see Audio Events for Legacy Audio Applications.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionControl Interface

IAudioSessionManager::GetSimpleAudioVolume method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSimpleAudioVolume** method retrieves a simple audio volume control.

Syntax

```
HRESULT GetSimpleAudioVolume(
  LPCGUID            AudioSessionGuid,
  DWORD              StreamFlags,
  ISimpleAudioVolume **AudioVolume
);
```

Parameters

`AudioSessionGuid`

Pointer to a session GUID. If the GUID does not identify a session that has been previously opened, the call opens a new but empty session. The Sndvol program does not display a volume-level control for a session unless it contains one or more active streams. If this parameter is **NULL** or points to the value GUID_NULL, the method assigns the stream to the default session.

`StreamFlags`

Specifies whether the request is for a cross-process session. Set to **TRUE** if the session is cross-process. Set to **FALSE** if the session is not cross-process.

`AudioVolume`

Pointer to a pointer variable into which the method writes a pointer to the ISimpleAudioVolume interface of the audio volume control object. This interface represents the simple audio volume control for the current process. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **Activate** call fails, **AudioVolume* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
AU DCL NT_ E_N OT_ INIT IALI ZED	The audio stream has not been successfully initialized.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_SE RVI CE_ NO T_R UN NIN G	The Windows audio service is not running.
E_P OIN TER	Parameter *AudioVolume* is **NULL**.
E_M EM OR Y	Out of memory.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionManager Interface

ISimpleAudioVolume Interface

IAudioSessionManager2 interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioSessionManager2** interface enables an application to manage submixes for the audio device.

To a get a reference to an **IAudioSessionManager2** interface, the application must activate it on the audio device by following these steps:

1. Use one of the techniques described on the IMMDevice interface page to obtain a reference to the **IMMDevice** interface for an audio endpoint device.
2. Call the IMMDevice::Activate method with parameter *iid* set to IID_IAudioSessionManager2.

When the application wants to release the **IAudioSessionManager2** interface instance, the application must call the interface's **Release** method.

The application thread that uses this interface must be initialized for COM. For more information about COM initialization, see the description of the **CoInitializeEx** function in the Windows SDK documentation.

Inheritance

The **IAudioSessionManager2** interface inherits from IAudioSessionManager. **IAudioSessionManager2** also has these types of members:

- Methods

Methods

The **IAudioSessionManager2** interface has these methods.

METHOD	DESCRIPTION
IAudioSessionManager2::GetSessionEnumerator	The GetSessionEnumerator method gets a pointer to the audio session enumerator object.
IAudioSessionManager2::RegisterDuckNotification	The RegisterDuckNotification method registers the application with the session manager to receive ducking notifications.
IAudioSessionManager2::RegisterSessionNotification	The RegisterSessionNotification method registers the application to receive a notification when a session is created.
IAudioSessionManager2::UnregisterDuckNotification	The UnregisterDuckNotification method deletes a previous registration by the application to receive notifications.
IAudioSessionManager2::UnregisterSessionNotification	The UnregisterSessionNotification method deletes the registration to receive a notification when a session is created.

Remarks

An application can use this interface to perform the following tasks:

- Register to receive ducking notifications.
- Register to receive a notification when a session is created.

- Enumerate sessions on the audio device that was used to get the interface pointer.

This interface supports custom implementations for *stream attenuation* or *ducking*, a new feature in Windows 7. An application playing a media stream can make the it behave differently when a new communication stream is opened on the default communication device. For example, the original media stream can be paused while the new communication stream is open. For more information about this feature, see Using a Communication Device.

An application that manages the media streams and wants to provide a custom ducking implementation, must register to receive notifications when session events occur. For stream attenuation, a session event is raised by the system when a communication stream is opened or closed on the default communication device. For more information, see Providing a Custom Ducking Behavior.

Examples

The following example code shows how to get a reference to the **IAudioSessionManager2** interface of the audio device.

```
HRESULT CreateSessionManager(IAudioSessionManager2** ppSessionManager)
{

    HRESULT hr = S_OK;

    IMMDevice* pDevice = NULL;
    IMMDeviceEnumerator* pEnumerator = NULL;
    IAudioSessionManager2* pSessionManager = NULL;


    // Create the device enumerator.
    CHECK_HR( hr = CoCreateInstance(
        __uuidof(MMDeviceEnumerator),
        NULL, CLSCTX_ALL,
        __uuidof(IMMDeviceEnumerator),
        (void**)&pEnumerator));

    // Get the default audio device.
    CHECK_HR( hr = pEnumerator->GetDefaultAudioEndpoint(
                    eRender, eConsole, &pDevice));

    // Get the session manager.
    CHECK_HR( hr = pDevice->Activate(
        __uuidof(IAudioSessionManager2), CLSCTX_ALL,
        NULL, (void**)&pSessionManager));

    // Return the pointer to the caller.
    *(ppSessionManager) = pSessionManager;
    (*ppSessionManager)->AddRef();

done:

    // Clean up.
    SAFE_RELEASE(pSessionManager);
    SAFE_RELEASE(pEnumerator);
    SAFE_RELEASE(pDevice);

    return hr;
}
```

Requirements

Minimum supported client	Windows 7 [desktop apps only]

Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

Core Audio Interfaces

IAudioSessionManager

IAudioSessionManager2::GetSessionEnumerator method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSessionEnumerator** method gets a pointer to the audio session enumerator object.

Syntax

```
HRESULT GetSessionEnumerator(
  IAudioSessionEnumerator **SessionEnum
);
```

Parameters

`SessionEnum`

Receives a pointer to the IAudioSessionEnumerator interface of the session enumerator object that the client can use to enumerate audio sessions on the audio device. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method.

Return value

If the method succeeds, it returns S_OK.

Remarks

The session manager maintains a collection of audio sessions that are active on the audio device by querying the audio engine. **GetSessionEnumerator** creates a session control for each session in the collection. To get a reference to the IAudioSessionControl interface of the session in the enumerated collection, the application must call IAudioSessionEnumerator::GetSession. For a code example, see IAudioSessionEnumerator Interface.

The session enumerator might not be aware of the new sessions that are reported through IAudioSessionNotification. So if an application exclusively relies on the session enumerator for getting all the sessions for an audio endpoint, the results might not be accurate. To work around this, the application should manually maintain a list. For more information, see IAudioSessionEnumerator.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionManager2

IAudioSessionManager2::RegisterDuckNotification method

1/11/2020 • 2 minutes to read • Edit Online

The **RegisterDuckNotification** method registers the application with the session manager to receive ducking notifications.

Syntax

```
HRESULT RegisterDuckNotification(
  LPCWSTR                      sessionID,
  IAudioVolumeDuckNotification *duckNotification
);
```

Parameters

`sessionID`

Pointer to a null-terminated string that contains a session instance identifier. Applications that are playing a media stream and want to provide custom stream attenuation or ducking behavior, pass their own session instance identifier. For more information, see Remarks.

Other applications that do not want to alter their streams but want to get all the ducking notifications must pass **NULL**.

`duckNotification`

Pointer to the application's implementation of the IAudioVolumeDuckNotification interface. The implementation is called when ducking events are raised by the audio system and notifications are sent to the registered applications.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN VALUE	DESCRIPTION
E_POINTER	*duckNotification* is **NULL**.
E_OUTOFMEMORY	Internal object could not be created due to insufficient memory.

Remarks

Stream Attenuation or *ducking* is a new feature in Windows 7. An application playing a media stream can make the stream behave differently when a new communication stream is opened on the default communication device. For example, the original media stream can be paused while the new communication stream is open. To provide this custom implementation for stream attenuation, the application can opt out of the default stream attenuation experience by calling IAudioSessionControl::SetDuckingPreference and then register itself to receive notifications when session events occur. For stream attenuation, a session event is raised by the system when a communication stream is opened or closed on the default communication device. For more information about this feature, see Getting Ducking Events.

To begin receiving notifications, the application calls the **RegisterDuckNotification** method to register its IAudioVolumeDuckNotification interface with the session manager. When the application no longer requires notifications, it calls the IAudioSessionManager2::UnregisterDuckNotification method to delete the registration.

The application receives notifications about the ducking events through the methods of the IAudioVolumeDuckNotification interface. The application implements **IAudioVolumeDuckNotification**. After the registration call has succeeded, the system calls the methods of this interface when session events occur.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionManager2

Using a Communication Device

IAudioSessionManager2::RegisterSessionNotification method

1/11/2020 • 2 minutes to read • Edit Online

The **RegisterSessionNotification** method registers the application to receive a notification when a session is created.

Syntax

```
HRESULT RegisterSessionNotification(
  IAudioSessionNotification *SessionNotification
);
```

Parameters

`SessionNotification`

A pointer to the application's implementation of the IAudioSessionNotification interface. If the method call succeeds, it calls the **AddRef** method on the application's **IAudioSessionNotification** interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN VALUE	DESCRIPTION
E_POINTER	*SessionNotification* is **NULL**.
E_OUTOFMEMORY	Internal object could not be created due to insufficient memory.

Remarks

The application can register to receive a notification when a session is created, through the methods of the IAudioSessionNotification interface. The application implements the **IAudioSessionNotification** interface. The methods defined in this interface receive callbacks from the system when a session is created. For example code that shows how to implement this interface, see

IAudioSessionNotification Interface.

To begin receiving notifications, the application calls the **IAudioSessionManager2::RegisterSessionNotification** method to register its IAudioSessionNotification

interface. When the application no longer requires notifications, it calls the IAudioSessionManager2::UnregisterSessionNotification method to delete the registration.

> **IMPORTANT**
> You must call IAudioSessionEnumerator::GetCount to begin receiving notifications. The session enumeration API discards new session notifications until the application has first retrieved the list of existing sessions. This is to prevent a race condition that can occur when a session notification arrives while the application using the session APIs is starting up. Calling **GetCount** triggers the enumeration API to begin sending session notifications.

> **Note** Make sure that the application initializes COM with Multithreaded Apartment (MTA) model by calling `CoInitializeEx(NULL, COINIT_MULTITHREADED)` in a non-UI thread. If MTA is not initialized, the application does not receive session notifications from the session manager. Threads that run the user interface of an application should be initialized apartment threading model.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionManager2

IAudioSessionManager2::UnregisterDuckNotification method

1/11/2020 • 2 minutes to read • Edit Online

The **UnregisterDuckNotification** method deletes a previous registration by the application to receive notifications.

Syntax

```
HRESULT UnregisterDuckNotification(
  IAudioVolumeDuckNotification *duckNotification
);
```

Parameters

`duckNotification`

Pointer to the IAudioVolumeDuckNotification interface that is implemented by the application. Pass the same interface pointer that was specified to the session manager in a previous call to the IAudioSessionManager2::RegisterDuckNotification method. If the **UnregisterDuckNotification** method succeeds, it calls the **Release** method on the application's **IAudioVolumeDuckNotification** interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN VALUE	DESCRIPTION
E_POINTER	*duckNotification* is **NULL**.

Remarks

The application calls this method when it no longer needs to receive notifications. The **UnregisterDuckNotification** method removes the registration of an IAudioVolumeDuckNotification interface that the application previously registered with the session manager by calling the IAudioSessionManager2::RegisterDuckNotification method.

After the application calls **UnregisterDuckNotification**, any pending events are not reported to the application.

Requirements

Minimum supported client	Windows 7 [desktop apps only]

Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

Default Ducking Experience

Getting Ducking Events

IAudioSessionManager2

IAudioSessionManager2::UnregisterSessionNotification method

1/11/2020 • 2 minutes to read • Edit Online

The **UnregisterSessionNotification** method deletes the registration to receive a notification when a session is created.

Syntax

```
HRESULT UnregisterSessionNotification(
  IAudioSessionNotification *SessionNotification
);
```

Parameters

`SessionNotification`

A pointer to the application's implementation of the IAudioSessionNotification interface. Pass the same interface pointer that was specified to the session manager in a previous call to IAudioSessionManager2::RegisterSessionNotification to register for notification.

If the **UnregisterSessionNotification** method succeeds, it calls the **Release** method on the application's IAudioSessionNotification interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN VALUE	DESCRIPTION
E_POINTER	*SessionNotification* is **NULL**.

Remarks

The application calls this method when it no longer needs to receive notifications. The **UnregisterSessionNotification** method removes the registration of an IAudioSessionNotification interface that the application previously registered with the session manager by calling the IAudioSessionControl::RegisterAudioSessionNotification method.

Requirements

Minimum supported client	Windows 7 [desktop apps only]

Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionManager2

IAudioSessionNotification interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioSessionNotification** interface provides notification when an audio session is created.

Inheritance

The **IAudioSessionNotification** interface inherits from the IUnknown interface. **IAudioSessionNotification** also has these types of members:

- Methods

Methods

The **IAudioSessionNotification** interface has these methods.

METHOD	DESCRIPTION
IAudioSessionNotification::OnSessionCreated	The OnSessionCreated method notifies the registered processes that the audio session has been created.

Remarks

Unlike the other WASAPI interfaces, which are implemented by the WASAPI system component, the **IAudioSessionNotification** interface is implemented by the application. To receive event notifications, the application passes to the IAudioSessionManager2::RegisterSessionNotification method a pointer to its **IAudioSessionNotification** implementation .

After registering its **IAudioSessionNotification** interface, the application receives event notifications in the form of callbacks through the methods in the interface.

When the application no longer needs to receive notifications, it calls the IAudioSessionManager2::UnregisterSessionNotification method. This method removes the registration of an **IAudioSessionNotification** interface that the application previously registered.

The application must not register or unregister notification callbacks during an event callback.

The session enumerator might not be aware of the new sessions that are reported through **IAudioSessionNotification**. So if an application exclusively relies on the session enumerator for getting all the sessions for an audio endpoint, the results might not be accurate. To work around this, the application should manually maintain a list. For more information, see IAudioSessionEnumerator.

> **Note** Make sure that the application initializes COM with Multithreaded Apartment (MTA) model by calling `CoInitializeEx(NULL, COINIT_MULTITHREADED)` in a non-UI thread. If MTA is not initialized, the application does not receive session notifications from the session manager. Threads that run the user interface of an application should be initialized apartment threading model.

Examples

The following code example shows a sample implementation of the **IAudioSessionNotification** interface.

```
class CSessionNotifications: public IAudioSessionNotification
{
private:

    LONG             m_cRefAll;
    HWND m_hwndMain;

    ~CSessionManager(){};

public:


    CSessionManager(HWND hWnd):
    m_cRefAll(1),
    m_hwndMain (hWnd)

    {}

    // IUnknown
    HRESULT STDMETHODCALLTYPE QueryInterface(REFIID riid, void **ppv)
    {
        if (IID_IUnknown == riid)
        {
            AddRef();
            *ppvInterface = (IUnknown*)this;
        }
        else if (__uuidof(IAudioSessionNotification) == riid)
        {
            AddRef();
            *ppvInterface = (IAudioSessionNotification*)this;
        }
        else
        {
            *ppvInterface = NULL;
            return E_NOINTERFACE;
        }
        return S_OK;
    }

    ULONG STDMETHODCALLTYPE AddRef()
    {
        return InterlockedIncrement(&m_cRefAll);
    }

    ULONG STDMETHODCALLTYPE Release)()
    {
        ULONG ulRef = InterlockedDecrement(&m_cRefAll);
        if (0 == ulRef)
        {
            delete this;
        }
        return ulRef;
    }

    HRESULT OnSessionCreated(IAudioSessionControl *pNewSession)
    {
        if (pNewSession)
        {
            PostMessage(m_hwndMain, WM_SESSION_CREATED, 0, 0);
        }
    }
};
```

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

Core Audio Interfaces

IAudioSessionNotification::OnSessionCreated method

1/11/2020 • 2 minutes to read • Edit Online

The **OnSessionCreated** method notifies the registered processes that the audio session has been created.

Syntax

```
HRESULT OnSessionCreated(
  IAudioSessionControl *NewSession
);
```

Parameters

`NewSession`

Pointer to the IAudioSessionControl interface of the audio session that was created.

Return value

If the method succeeds, it returns S_OK.

Remarks

After registering its IAudioSessionNotification interface, the application receives event notifications in the form of callbacks through the methods of the interface.

The audio engine calls **OnSessionCreated** when a new session is activated on the device endpoint. This method is called from the session manager thread. This method must take a reference to the session in the *NewSession* parameter if it wants to keep the reference after this call completes.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioSessionNotification

IAudioVolumeDuckNotification interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioVolumeDuckNotification** interface is used to by the system to send notifications about stream attenuation changes.Stream Attenuation, or ducking, is a feature introduced in Windows 7, where the system adjusts the volume of a non-communication stream when a new communication stream is opened. For more information about this feature, see Default Ducking Experience.

Inheritance

The **IAudioVolumeDuckNotification** interface inherits from the IUnknown interface. **IAudioVolumeDuckNotification** also has these types of members:

- Methods

Methods

The **IAudioVolumeDuckNotification** interface has these methods.

METHOD	DESCRIPTION
IAudioVolumeDuckNotification::OnVolumeDuckNotification	The OnVolumeDuckNotification method sends a notification about a pending system ducking event.
IAudioVolumeDuckNotification::OnVolumeUnduckNotification	The OnVolumeUnduckNotification method sends a notification about a pending system unducking event.

Remarks

If an application needs to opt out of the system attenuation experience provided by the system, it must call IAudioSessionControl2::SetDuckingPreference and specify that preference.

Unlike the other WASAPI interfaces, which are implemented by the WASAPI system component, the **IAudioVolumeDuckNotification** interface is implemented by the application to provide custom stream attenuation behavior. To receive event notifications, the application passes to the IAudioSessionManager2::RegisterDuckNotification method a pointer to the application's implementation of **IAudioVolumeDuckNotification**.

After the application has registered its **IAudioVolumeDuckNotification** interface, the session manager calls the **IAudioVolumeDuckNotification** implementation when it needs to send ducking notifications. The application receives event notifications in the form of callbacks through the methods of the interface.

When the application no longer needs to receive notifications, it calls the IAudioSessionManager2::UnregisterDuckNotification method. The **UnregisterDuckNotification** method removes the registration of an **IAudioVolumeDuckNotification** interface that the application previously registered.

The application must not register or unregister notification callbacks during an event callback.

For more information, see Implementation Considerations for Ducking Notifications.

Examples

The following example code shows a sample implementation of the **IAudioVolumeDuckNotification** interface.

```
class CDuckNotification : public IAudioVolumeDuckNotification
{
    LONG            _Cref;
    HWND            m_hwndMain;

    CDuckNotification (HWND hWnd) :
        _Cref(1),
        m_hwndMain (hWnd)
        {}


    HRESULT OnVolumeDuckNotification (LPCWSTR SessionID, UINT32 CommunicationSessionCount)
    {
         PostMessage(m_hwndMain, WM_VOLUME_DUCK, 0, 0);
         return S_OK;
    }
    HRESULT OnVolumeUnduckNotification (LPCWSTR SessionID)
    {
         PostMessage(m_hwndMain, WM_VOLUME_UNDUCK, 0, 0);
         return S_OK;
    }

protected:
    ~CDuckNotification() {}

public:
    HRESULT QueryInterface (REFIID Iid, void** ReturnValue)
    {
        if (ReturnValue == NULL)
        {
            return E_POINTER;
        }
        *ReturnValue = NULL;
        if (iid == IID_IUnknown)
        {
            *ReturnValue = static_cast<IUnknown *>(static_cast<IAudioVolumeDuckNotification *>(this));
            AddRef();
        }
        else if (iid == __uuidof(IAudioVolumeDuckNotification))
        {
            *ReturnValue = static_cast<IAudioVolumeDuckNotification *>(this);
            AddRef();
        }
        else
        {
            return E_NOINTERFACE;
        }
        return S_OK;
    }
    ULONG AddRef()
    {
        return InterlockedIncrement(&_Cref);
    }

    ULONG Release()
    {
        LONG ref = InterlockedDecrement(&_Cref);
        if (ref == 0)
        {
            delete this;
        }
        return 0;
    }
};
```

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

Core Audio Interfaces

Using a Communication Device

IAudioVolumeDuckNotification::OnVolumeDuckNotification method

1/11/2020 • 2 minutes to read • Edit Online

The **OnVolumeDuckNotification** method sends a notification about a pending system ducking event. For more information, see Implementation Considerations for Ducking Notifications.

Syntax

```
HRESULT OnVolumeDuckNotification(
  LPCWSTR sessionID,
  UINT32  countCommunicationSessions
);
```

Parameters

`sessionID`

A string containing the session instance identifier of the communications session that raises the the auto-ducking event. To get the session instance identifier, call IAudioSessionControl2::GetSessionInstanceIdentifier.

`countCommunicationSessions`

The number of active communications sessions. If there are n sessions, the sessions are numbered from 0 to −1.

Return value

If the method succeeds, it returns S_OK.

Remarks

After the application registers its implementation of the IAudioVolumeDuckNotification interface by calling IAudioSessionManager2::RegisterDuckNotification, the session manager calls **OnVolumeDuckNotification** when it wants to send a notification about when ducking begins. The application receives the event notifications in the form of callbacks.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioVolumeDuckNotification

IAudioVolumeDuckNotification::OnVolumeUnduckNotification method

1/11/2020 • 2 minutes to read • Edit Online

The **OnVolumeUnduckNotification** method sends a notification about a pending system unducking event. For more information, see Implementation Considerations for Ducking Notifications.

Syntax

```
HRESULT OnVolumeUnduckNotification(
  LPCWSTR sessionID
);
```

Parameters

`sessionID`

A string containing the session instance identifier of the terminating communications session that intiated the ducking. To get the session instance identifier, call IAudioSessionControl2::GetSessionInstanceIdentifier.

Return value

If the method succeeds, it returns S_OK.

Remarks

After the application registers its implementation of the IAudioVolumeDuckNotification interface by calling IAudioSessionManager2::RegisterDuckNotification, the session manager calls **OnVolumeUnduckNotification** when it wants to send a notification about when ducking ends. The application receives the event notifications in the form of callbacks.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	audiopolicy.h

See also

IAudioVolumeDuckNotification

Using a Communication Device

audiosessiontypes.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs audiosessiontypes.h contains the following programming interfaces:

Enumerations

TITLE	DESCRIPTION
AUDCLNT_SHAREMODE	The AUDCLNT_SHAREMODE enumeration defines constants that indicate whether an audio stream will run in shared mode or in exclusive mode.
AUDIO_STREAM_CATEGORY	Specifies the category of an audio stream.
AudioSessionState	The AudioSessionState enumeration defines constants that indicate the current state of an audio session.

AUDCLNT_SHAREMODE enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **AUDCLNT_SHAREMODE** enumeration defines constants that indicate whether an audio stream will run in shared mode or in exclusive mode.

Syntax

```
typedef enum _AUDCLNT_SHAREMODE {
  AUDCLNT_SHAREMODE_SHARED,
  AUDCLNT_SHAREMODE_EXCLUSIVE
} AUDCLNT_SHAREMODE;
```

Constants

AUDCLNT_SHAREMODE_SHARED	The audio stream will run in shared mode. For more information, see Remarks.
AUDCLNT_SHAREMODE_EXCLUSIVE	The audio stream will run in exclusive mode. For more information, see Remarks.

Remarks

The IAudioClient::Initialize and IAudioClient::IsFormatSupported methods use the constants defined in the **AUDCLNT_SHAREMODE** enumeration.

In shared mode, the client can share the audio endpoint device with clients that run in other user-mode processes. The audio engine always supports formats for client streams that match the engine's mix format. In addition, the audio engine might support another format if the Windows audio service can insert system effects into the client stream to convert the client format to the mix format.

In exclusive mode, the Windows audio service attempts to establish a connection in which the client has exclusive access to the audio endpoint device. In this mode, the audio engine inserts no system effects into the local stream to aid in the creation of the connection point. Either the audio device can handle the specified format directly or the method fails.

For more information about shared-mode and exclusive-mode streams, see User-Mode Audio Components.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Header	audiosessiontypes.h

See also

Core Audio Constants

Core Audio Enumerations

IAudioClient::Initialize

IAudioClient::IsFormatSupported

AUDIO_STREAM_CATEGORY enumeration

1/11/2020 • 2 minutes to read • Edit Online

Specifies the category of an audio stream.

Syntax

```
typedef enum _AUDIO_STREAM_CATEGORY {
  AudioCategory_Other,
  AudioCategory_ForegroundOnlyMedia,
  AudioCategory_BackgroundCapableMedia,
  AudioCategory_Communications,
  AudioCategory_Alerts,
  AudioCategory_SoundEffects,
  AudioCategory_GameEffects,
  AudioCategory_GameMedia,
  AudioCategory_GameChat,
  AudioCategory_Speech,
  AudioCategory_Movie,
  AudioCategory_Media
} AUDIO_STREAM_CATEGORY;
```

Constants

AudioCategory_Other	Other audio stream.
AudioCategory_ForegroundOnlyMedia	Media that will only stream when the app is in the foreground. This enumeration value has been deprecated. For more information, see the Remarks section.
AudioCategory_BackgroundCapableMedia	Media that can be streamed when the app is in the background. This enumeration value has been deprecated. For more information, see the Remarks section.
AudioCategory_Communications	Real-time communications, such as VOIP or chat.
AudioCategory_Alerts	Alert sounds.
AudioCategory_SoundEffects	Sound effects.
AudioCategory_GameEffects	Game sound effects.
AudioCategory_GameMedia	Background audio for games.
AudioCategory_GameChat	Game chat audio. Similar to **AudioCategory_Communications** except that **AudioCategory_GameChat** will not attenuate other streams.
AudioCategory_Speech	Speech.

AudioCategory_Movie	Stream that includes audio with dialog.
AudioCategory_Media	Stream that includes audio without dialog.

Remarks

Note that only a subset of the audio stream categories are valid for certain stream types.

STREAM TYPE	VALID CATEGORIES
Render stream	All categories are valid.
Capture stream	AudioCategory_Communications, AudioCategory_Speech, AudioCategory_Other
Loopback stream	AudioCategory_Other

Games should categorize their music streams as **AudioCategory_GameMedia** so that game music mutes automatically if another application plays music in the background. Music or video applications should categorize their streams as **AudioCategory_Media** or **AudioCategory_Movie** so they will take priority over **AudioCategory_GameMedia** streams. Game audio for in-game cinematics or cutscenes, when the audio is premixed or for creative reasons should take priority over background audio, should also be categorized as **Media** or **Movie**.

The values **AudioCategory_ForegroundOnlyMedia** and **AudioCategory_BackgroundCapableMedia** are deprecated. For Windows Store apps, these values will continue to function the same when running on Windows 10 as they did on Windows 8.1. Attempting to use these values in a Universal Windows Platform (UWP) app, will result in compilation errors and an exception at runtime. Using these values in a Windows desktop application built with the Windows 10 SDK the will result in a compilation error.

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Header	audiosessiontypes.h (include Audioclient.h)

See also

Core Audio Enumerations

AudioSessionState enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **AudioSessionState** enumeration defines constants that indicate the current state of an audio session.

Syntax

```
typedef enum _AudioSessionState {
  AudioSessionStateInactive,
  AudioSessionStateActive,
  AudioSessionStateExpired
} AudioSessionState;
```

Constants

AudioSessionStateInactive	The audio session is inactive. (It contains at least one stream, but none of the streams in the session is currently running.)
AudioSessionStateActive	The audio session is active. (At least one of the streams in the session is running.)
AudioSessionStateExpired	The audio session has expired. (It contains no streams.)

Remarks

When a client opens a session by assigning the first stream to the session (by calling the IAudioClient::Initialize method), the initial session state is inactive. The session state changes from inactive to active when a stream in the session begins running (because the client has called the IAudioClient::Start method). The session changes from active to inactive when the client stops the last running stream in the session (by calling the IAudioClient::Stop method). The session state changes to expired when the client destroys the last stream in the session by releasing all references to the stream object.

The system volume-control program, Sndvol, displays volume controls for both active and inactive sessions. Sndvol stops displaying the volume control for a session when the session state changes to expired. For more information about Sndvol, see Audio Sessions.

The IAudioSessionControl::GetState and IAudioSessionEvents::OnStateChanged methods use the constants defined in the **AudioSessionState** enumeration.

For more information about session states, see Audio Sessions.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]

Header	audiosessiontypes.h

See also

Core Audio Constants

Core Audio Enumerations

IAudioClient::Initialize

IAudioClient::Start

IAudioClient::Stop

IAudioSessionControl::GetState

IAudioSessionEvents::OnStateChanged

devicetopology.h header

1/18/2020 • 3 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs devicetopology.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IAudioAutoGainControl	The IAudioAutoGainControl interface provides access to a hardware automatic gain control (AGC).
IAudioBass	The IAudioBass interface provides access to a hardware bass-level control.
IAudioChannelConfig	The IAudioChannelConfig interface provides access to a hardware channel-configuration control.
IAudioInputSelector	The IAudioInputSelector interface provides access to a hardware multiplexer control (input selector).
IAudioLoudness	The IAudioLoudness interface provides access to a "loudness" compensation control.
IAudioMidrange	The IAudioMidrange interface provides access to a hardware midrange-level control.
IAudioMute	The IAudioMute interface provides access to a hardware mute control.
IAudioOutputSelector	The IAudioOutputSelector interface provides access to a hardware demultiplexer control (output selector).
IAudioPeakMeter	The IAudioPeakMeter interface provides access to a hardware peak-meter control.
IAudioTreble	The IAudioTreble interface provides access to a hardware treble-level control.
IAudioVolumeLevel	The IAudioVolumeLevel interface provides access to a hardware volume control.
IConnector	The IConnector interface represents a point of connection between components.
IControlChangeNotify	The IControlChangeNotify interface provides notifications when the status of a part (connector or subunit) changes.

TITLE	DESCRIPTION
IControlInterface	The IControlInterface interface represents a control interface on a part (connector or subunit) in a device topology. The client obtains a reference to a part's IControlInterface interface by calling the IPart::GetControlInterface method.
IDeviceSpecificProperty	The IDeviceSpecificProperty interface provides access to the control value of a device-specific hardware control.
IDeviceTopology	The IDeviceTopology interface provides access to the topology of an audio device.
IKsFormatSupport	The IKsFormatSupport interface provides information about the audio data formats that are supported by a software-configured I/O connection (typically a DMA channel) between an audio adapter device and system memory.
IKsJackDescription	The IKsJackDescription interface provides information about the jacks or internal connectors that provide a physical connection between a device on an audio adapter and an external or internal endpoint device (for example, a microphone or CD player).
IKsJackDescription2	The IKsJackDescription2 interface provides information about the jacks or internal connectors that provide a physical connection between a device on an audio adapter and an external or internal endpoint device (for example, a microphone or CD player).
IKsJackSinkInformation	The IKsJackSinkInformation interface provides access to jack sink information if the jack is supported by the hardware.
IPart	The IPart interface represents a part (connector or subunit) of a device topology.
IPartsList	The IPartsList interface represents a list of parts, each of which is an object with an IPart interface that represents a connector or subunit.
IPerChannelDbLevel	The IPerChannelDbLevel interface represents a generic subunit control interface that provides per-channel control over the volume level, in decibels, of an audio stream or of a frequency band in an audio stream.
ISubunit	The ISubunit interface represents a hardware subunit (for example, a volume control) that lies in the data path between a client and an audio endpoint device.

Structures

TITLE	DESCRIPTION
KSJACK_DESCRIPTION	The KSJACK_DESCRIPTION structure describes an audio jack.

TITLE	DESCRIPTION
KSJACK_DESCRIPTION2	The KSJACK_DESCRIPTION2 structure describes an audio jack.To get the description of an audio jack of a connector, call IKsJackDescription2::GetJackDescription2.
KSJACK_SINK_INFORMATION	The KSJACK_SINK_INFORMATION structure stores information about an audio jack sink.
LUID	The LUID structure stores the video port identifier. This structure is stored in the PortId member of the KSJACK_SINK_INFORMATION structure.

Enumerations

TITLE	DESCRIPTION
ConnectorType	The ConnectorType enumeration indicates the type of connection that a connector is part of.
DataFlow	The DataFlow enumeration indicates the data-flow direction of an audio stream through a connector.
KSJACK_SINK_CONNECTIONTYPE	The KSJACK_SINK_CONNECTIONTYPE enumeration defines constants that specify the type of connection. These values are used in the KSJACK_SINK_INFORMATION structure that stores information about an audio jack sink.
PartType	The PartType enumeration defines constants that indicate whether a part in a device topology is a connector or subunit.

ConnectorType enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **ConnectorType** enumeration indicates the type of connection that a connector is part of.

Syntax

```
typedef enum __MIDL___MIDL_itf_devicetopology_0000_0000_0013 {
  Unknown_Connector,
  Physical_Internal,
  Physical_External,
  Software_IO,
  Software_Fixed,
  Network
} ConnectorType;
```

Constants

Unknown_Connector	The connector is part of a connection of unknown type.
Physical_Internal	The connector is part of a physical connection to an auxiliary device that is installed inside the system chassis (for example, a connection to the analog output of an internal CD player, or to a built-in microphone or built-in speakers in a laptop computer).
Physical_External	The connector is part of a physical connection to an external device. That is, the connector is a user-accessible jack that connects to a microphone, speakers, headphones, S/PDIF input or output device, or line input or output device.
Software_IO	The connector is part of a software-configured I/O connection (typically a DMA channel) between system memory and an audio hardware device on an audio adapter.
Software_Fixed	The connector is part of a permanent connection that is fixed and cannot be configured under software control. This type of connection is typically used to connect two audio hardware devices that reside on the same adapter.
Network	The connector is part of a connection to a network.

Remarks

The IConnector::GetType method uses the constants defined in the **ConnectorType** enumeration.

For more information about connector types, see Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	devicetopology.h

See also

Core Audio Constants

Core Audio Enumerations

IConnector::GetType

DataFlow enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **DataFlow** enumeration indicates the data-flow direction of an audio stream through a connector.

Syntax

```
typedef enum __MIDL___MIDL_itf_devicetopology_0000_0000_0011 {
  In,
  Out
} DataFlow;
```

Constants

In	Input stream. The audio stream flows into the device through the connector.
Out	Output stream. The audio stream flows out of the device through the connector.

Remarks

The IConnector::GetDataFlow method uses the constants defined in the **DataFlow** enumeration.

The topology of a rendering or capture device on an audio adapter typically has one or more connectors with a data-flow direction of "In" through which audio data enters the device, and one or more connectors with a data-flow direction of "Out" through which audio data exits the device. For example, a typical rendering device on an adapter has a connector with data-flow direction "In" through which the Windows audio engine streams PCM data into the device. The same device has a connector with data-flow direction "Out" through which the device transmits an audio signal to speakers or headphones.

The topology of a rendering endpoint device (for example, headphones) has a single connector with data-flow direction "In" through which audio data (in the form of an analog signal) enters the device.

The topology of a capture endpoint device (for example, a microphone) has a single connector with data-flow direction "Out" through which audio data exits the device.

For more information, see Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	devicetopology.h

See also

Core Audio Constants

Core Audio Enumerations

IConnector::GetDataFlow

IAudioAutoGainControl interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioAutoGainControl** interface provides access to a hardware automatic gain control (AGC). The client obtains a reference to the **IAudioAutoGainControl** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioAutoGainControl. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioAutoGainControl** interface. Only a subunit object that represents a hardware AGC function will support this interface.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioAutoGainControl** interface provides convenient access to the KSPROPERTY_AUDIO_AGC property of a subunit that has a subtype GUID value of KSNODETYPE_AGC. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The **IAudioAutoGainControl** interface inherits from the IUnknown interface. **IAudioAutoGainControl** also has these types of members:

- Methods

Methods

The **IAudioAutoGainControl** interface has these methods.

METHOD	DESCRIPTION
IAudioAutoGainControl::GetEnabled	The GetEnabled method gets the current state (enabled or disabled) of the AGC.
IAudioAutoGainControl::SetEnabled	The SetEnabled method enables or disables the AGC.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IAudioAutoGainControl::GetEnabled method

1/11/2020 • 2 minutes to read • Edit Online

The **GetEnabled** method gets the current state (enabled or disabled) of the AGC.

Syntax

```
HRESULT GetEnabled(
  BOOL *pbEnabled
);
```

Parameters

`pbEnabled`

Pointer to a **BOOL** variable into which the method writes the current AGC state. If the state is **TRUE**, AGC is enabled. If **FALSE**, AGC is disabled.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pbEnabled* is **NULL**.

Remarks

A disabled AGC operates in pass-through mode. In this mode, the audio stream passes through the AGC without modification.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioAutoGainControl::SetEnabled method

1/11/2020 • 2 minutes to read • Edit Online

The **SetEnabled** method enables or disables the AGC.

Syntax

```
HRESULT SetEnabled(
  BOOL    bEnable,
  LPCGUID pguidEventContext
);
```

Parameters

`bEnable`

The new AGC state. If this parameter is **TRUE** (nonzero), the method enables AGC. If **FALSE**, it disables AGC.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetEnabled** call changes the state of the AGC control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_OUTOFMEMORY	Out of memory.

Remarks

A disabled AGC control operates in pass-through mode. In this mode, the audio stream passes through the control without modification.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioAutoGainControl Interface

IAudioBass interface

1/24/2020 • 2 minutes to read • Edit Online

The **IAudioBass** interface provides access to a hardware bass-level control. The client obtains a reference to the **IAudioBass** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioBass. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioBass** interface. Only a subunit object that represents a hardware function for controlling the level of the bass frequencies in each channel will support this interface.

Inheritance

The IAudioBass interface inherits from the IPerChannelDbLevel interface.

Methods

The **IAudioBass** interface has these methods.

METHOD	DESCRIPTION

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IPerChannelDbLevel Interface

IAudioChannelConfig interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioChannelConfig** interface provides access to a hardware channel-configuration control. The client obtains a reference to the **IAudioChannelConfig** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioChannelConfig. The call to IPart::Activate succeeds only if the subunit supports the **IAudioChannelConfig** interface. Only a subunit object that represents a hardware channel-configuration control will support this interface.

A client of the **IAudioChannelConfig** interface programs a hardware channel-configuration control by writing a channel-configuration mask to the control. The mask specifies the assignment of audio channels to speakers. For more information about channel-configuration masks, see the following:

- The discussion of the KSPROPERTY_AUDIO_CHANNEL_CONFIG property in the Windows DDK documentation.
- The white paper titled "Audio Driver Support for Home Theater Speaker Configurations" at the Audio Device Technologies for Windows website.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioChannelConfig** interface provides convenient access to the KSPROPERTY_AUDIO_CHANNEL_CONFIG property of a subunit that has a subtype GUID value of KSNODETYPE_3D_EFFECTS, KSNODETYPE_DAC, KSNODETYPE_VOLUME, or KSNODETYPE_PROLOGIC_DECODER. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The **IAudioChannelConfig** interface inherits from the IUnknown interface. **IAudioChannelConfig** also has these types of members:

- Methods

Methods

The **IAudioChannelConfig** interface has these methods.

METHOD	DESCRIPTION
IAudioChannelConfig::GetChannelConfig	The GetChannelConfig method gets the current channel-configuration mask from a channel-configuration control.
IAudioChannelConfig::SetChannelConfig	The SetChannelConfig method sets the channel-configuration mask in a channel-configuration control.

Requirements

Minimum supported client	Windows Vista [desktop apps only]

Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IAudioChannelConfig::GetChannelConfig method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelConfig** method gets the current channel-configuration mask from a channel-configuration control.

Syntax

```
HRESULT GetChannelConfig(
  DWORD *pdwConfig
);
```

Parameters

`pdwConfig`

Pointer to a **DWORD** variable into which the method writes the current channel-configuration mask value.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pdwConfig* is **NULL**.

Remarks

For information about channel-configuration masks, see the discussion of the KSPROPERTY_AUDIO_CHANNEL_CONFIG property in the Windows DDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioChannelConfig Interface

IAudioChannelConfig::SetChannelConfig

IAudioChannelConfig::SetChannelConfig method

1/11/2020 • 2 minutes to read • Edit Online

The **SetChannelConfig** method sets the channel-configuration mask in a channel-configuration control.

Syntax

```
HRESULT SetChannelConfig(
  DWORD   dwConfig,
  LPCGUID pguidEventContext
);
```

Parameters

`dwConfig`

The channel-configuration mask.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetChannelConfig** call changes the state of the channel-configuration control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_OUTOFMEMORY	Out of memory.

Remarks

For information about channel-configuration masks, see the discussion of the KSPROPERTY_AUDIO_CHANNEL_CONFIG property in the Windows DDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioChannelConfig Interface

IAudioChannelConfig::GetChannelConfig

IAudioInputSelector interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioInputSelector** interface provides access to a hardware multiplexer control (input selector). The client obtains a reference to the **IAudioInputSelector** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioInputSelector. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioInputSelector** interface. Only a subunit object that represents a hardware input selector will support this interface.

Each input of an input selector is identified by the local ID of the part (a connector or subunit of a device topology) that has a direct link to the input. A local ID is a number that uniquely identifies a part among all the parts in a device topology.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioInputSelector** interface provides convenient access to the KSPROPERTY_AUDIO_MUX_SOURCE property of a subunit that has a subtype GUID value of KSNODETYPE_MUX. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

For a code example that uses the **IAudioInputSelector** interface, see the implementation of the SelectCaptureDevice function in Device Topologies.

Inheritance

The **IAudioInputSelector** interface inherits from the IUnknown interface. **IAudioInputSelector** also has these types of members:

- Methods

Methods

The **IAudioInputSelector** interface has these methods.

METHOD	DESCRIPTION
IAudioInputSelector::GetSelection	The GetSelection method gets the local ID of the part that is connected to the selector input that is currently selected.
IAudioInputSelector::SetSelection	The SetSelection method selects one of the inputs of the input selector.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IAudioInputSelector::GetSelection method

1/11/2020 • 2 minutes to read • Edit Online

The GetSelection method gets the local ID of the part that is connected to the selector input that is currently selected.

Syntax

```
HRESULT GetSelection(
  UINT *pnIdSelected
);
```

Parameters

`pnIdSelected`

Pointer to a **UINT** variable into which the method writes the local ID of the part that directly links to the currently selected selector input.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pnIdSelected* is **NULL**.

Remarks

A local ID is a number that uniquely identifies a part among all parts in a device topology. To obtain a pointer to the IPart interface of a part from its local ID, call the IDeviceTopology::GetPartById method.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioInputSelector Interface

IDeviceTopology::GetPartById

IPart Interface

IAudioInputSelector::SetSelection method

1/11/2020 • 2 minutes to read • Edit Online

The **SetSelection** method selects one of the inputs of the input selector.

Syntax

```
HRESULT SetSelection(
  UINT    nIdSelect,
  LPCGUID pguidEventContext
);
```

Parameters

`nIdSelect`

The new selector input. The caller should set this parameter to the local ID of a part that has a direct link to one of the selector inputs.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetSelection** call changes the state of the input-selector control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *nIdSelect* is not the local ID of a part at a selector input.
E_OUTOFMEMORY	Out of memory.

Remarks

A local ID is a number that uniquely identifies a part among all parts in a device topology. To obtain the local ID of a part, call the IPart::GetLocalId method on the part object.

For a code example that calls the **SetSelection** method, see the implementation of the SelectCaptureDevice function in Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioInputSelector Interface

IPart::GetLocalId

IAudioLoudness interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioLoudness** interface provides access to a "loudness" compensation control. The client obtains a reference to the **IAudioLoudness** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioLoudness. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioLoudness** interface. Only a subunit object that represents a hardware loudness control function will support this interface.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioLoudness** interface provides convenient access to the KSPROPERTY_AUDIO_LOUDNESS property of a subunit that has a subtype GUID value of KSNODETYPE_LOUDNESS. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The **IAudioLoudness** interface inherits from the IUnknown interface. **IAudioLoudness** also has these types of members:

- Methods

Methods

The **IAudioLoudness** interface has these methods.

METHOD	DESCRIPTION
IAudioLoudness::GetEnabled	The GetEnabled method gets the current state (enabled or disabled) of the loudness control.
IAudioLoudness::SetEnabled	The SetEnabled method enables or disables the loudness control.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IAudioLoudness::GetEnabled method

1/11/2020 • 2 minutes to read • Edit Online

The **GetEnabled** method gets the current state (enabled or disabled) of the loudness control.

Syntax

```
HRESULT GetEnabled(
  BOOL *pbEnabled
);
```

Parameters

`pbEnabled`

Pointer to a **BOOL** variable into which the method writes the current loudness state. If the state is **TRUE**, loudness is enabled. If **FALSE**, loudness is disabled.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pbEnabled* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioLoudness Interface

IAudioLoudness::SetEnabled method

1/11/2020 • 2 minutes to read • Edit Online

The **SetEnabled** method enables or disables the loudness control.

Syntax

```
HRESULT SetEnabled(
  BOOL    bEnable,
  LPCGUID pguidEventContext
);
```

Parameters

`bEnable`

The new loudness state. If *bEnable* is **TRUE** (nonzero), the method enables loudness. If **FALSE**, it disables loudness.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetEnabled** call changes the state of the loudness control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_OUTOFMEMORY	Out of memory.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

IAudioLoudness Interface

IAudioMidrange interface

1/24/2020 • 2 minutes to read • Edit Online

The **IAudioMidrange** interface provides access to a hardware midrange-level control. The client obtains a reference to the **IAudioMidrange** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioMidrange. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioMidrange** interface. Only a subunit object that represents a hardware function for controlling the level of the mid-range frequencies in each channel will support this interface.

The **IAudioMidrange** interface provides per-channel controls for setting and getting the gain or attenuation level of the midrange frequencies in the audio stream. If a midrange-level hardware control can only attenuate the channels in the audio stream, then the maximum midrange level for any channel is 0 dB. If a midrange-level control can provide gain (amplification), then the maximum midrange level is greater than 0 dB.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioMidrange** interface provides convenient access to the KSPROPERTY_AUDIO_MID property of a subunit that has a subtype GUID value of KSNODETYPE_TONE. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The IAudioMidrange interface inherits from the IPerChannelDbLevel interface.

Methods

The **IAudioMidrange** interface has these methods.

METHOD	DESCRIPTION

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IAudioMute interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioMute** interface provides access to a hardware mute control. The client obtains a reference to the **IAudioMute** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioMute. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioMute** interface. Only a subunit object that represents a hardware mute control function will support this interface.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioMute** interface provides convenient access to the KSPROPERTY_AUDIO_MUTE property of a subunit that has a subtype GUID value of KSNODETYPE_MUTE. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The **IAudioMute** interface inherits from the IUnknown interface. **IAudioMute** also has these types of members:

- Methods

Methods

The **IAudioMute** interface has these methods.

METHOD	DESCRIPTION
IAudioMute::GetMute	The GetMute method gets the current state (enabled or disabled) of the mute control.
IAudioMute::SetMute	The SetMute method enables or disables the mute control.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IAudioMute::GetMute method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMute** method gets the current state (enabled or disabled) of the mute control.

Syntax

```
HRESULT GetMute(
  BOOL *pbMuted
);
```

Parameters

`pbMuted`

Pointer to a **BOOL** variable into which the method writes the current state of the mute control. If the state is **TRUE**, muting is enabled. If **FALSE**, it is disabled.

Return value

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pbMuted* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioMute Interface

IAudioMute::SetMute method

1/11/2020 • 2 minutes to read • Edit Online

The **SetMute** method enables or disables the mute control.

Syntax

```
HRESULT SetMute(
  BOOL    bMuted,
  LPCGUID pguidEventContext
);
```

Parameters

`bMuted`

The new muting state. If *bMuted* is **TRUE** (nonzero), the method enables muting. If **FALSE**, the method disables muting.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetMute** call changes the state of the mute control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_OUTOFMEMORY	Out of memory.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

IAudioMute Interface

IAudioOutputSelector interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioOutputSelector** interface provides access to a hardware demultiplexer control (output selector). The client obtains a reference to the **IAudioOutputSelector** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioOutputSelector. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioOutputSelector** interface. Only a subunit object that represents a hardware output selector will support this interface.

Each output of an output selector is identified by the local ID of the part (a connector or subunit of a device topology) with a direct link to the output. A local ID is a number that uniquely identifies a part among all the parts in a device topology.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioOutputSelector** interface provides convenient access to the KSPROPERTY_AUDIO_DEMUX_DEST property of a subunit that has a subtype GUID value of KSNODETYPE_DEMUX. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The **IAudioOutputSelector** interface inherits from the IUnknown interface. **IAudioOutputSelector** also has these types of members:

- Methods

Methods

The **IAudioOutputSelector** interface has these methods.

METHOD	DESCRIPTION
IAudioOutputSelector::GetSelection	The GetSelection method gets the local ID of the part that is connected to the selector output that is currently selected.
IAudioOutputSelector::SetSelection	The SetSelection method selects one of the outputs of the output selector.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IAudioOutputSelector::GetSelection method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSelection** method gets the local ID of the part that is connected to the selector output that is currently selected.

Syntax

```
HRESULT GetSelection(
  UINT *pnIdSelected
);
```

Parameters

`pnIdSelected`

Pointer to a **UINT** variable into which the method writes the local ID of the part that has a direct link to the currently selected selector output.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pnIdSelected* is **NULL**.

Remarks

A local ID is a number that uniquely identifies a part among all parts in a device topology. To obtain a pointer to the IPart interface of a part from its local ID, call the IDeviceTopology::GetPartById method.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioOutputSelector Interface

IDeviceTopology::GetPartById

IPart Interface

IAudioOutputSelector::SetSelection method

1/11/2020 • 2 minutes to read • Edit Online

The **SetSelection** method selects one of the outputs of the output selector.

Syntax

```
HRESULT SetSelection(
  UINT    nIdSelect,
  LPCGUID pguidEventContext
);
```

Parameters

`nIdSelect`

The new selector output. The caller should set this parameter to the local ID of a part that has a direct link to one of the selector outputs.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetSelection** call changes the state of the output-selector control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *nIdSelect* is not the local ID of a part at a selector output.
E_OUTOFMEMORY	Out of memory.

Remarks

A local ID is a number that uniquely identifies a part among all parts in a device topology. To obtain the local ID of a part, call the IPart::GetLocalId method on the part object.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioOutputSelector Interface

IPart::GetLocalId

IAudioPeakMeter interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioPeakMeter** interface provides access to a hardware peak-meter control. The client obtains a reference to the **IAudioPeakMeter** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioPeakMeter. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioPeakMeter** interface. Only a subunit object that represents a hardware peak meter will support this interface.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioPeakMeter** interface provides convenient access to the KSPROPERTY_AUDIO_PEAKMETER property of a subunit that has a subtype GUID value of KSNODETYPE_PEAKMETER. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The **IAudioPeakMeter** interface inherits from the IUnknown interface. **IAudioPeakMeter** also has these types of members:

- Methods

Methods

The **IAudioPeakMeter** interface has these methods.

METHOD	DESCRIPTION
IAudioPeakMeter::GetChannelCount	The GetChannelCount method gets the number of channels in the audio stream.
IAudioPeakMeter::GetLevel	The GetLevel method gets the peak level that the peak meter recorded for the specified channel since the peak level for that channel was previously read.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IAudioPeakMeter::GetChannelCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelCount** method gets the number of channels in the audio stream.

Syntax

```
HRESULT GetChannelCount(
  UINT *pcChannels
);
```

Parameters

`pcChannels`

Pointer to a **UINT** variable into which the method writes the channel count.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pcChannels* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioPeakMeter Interface

IAudioPeakMeter::GetLevel method

1/11/2020 • 2 minutes to read • Edit Online

The **GetLevel** method gets the peak level that the peak meter recorded for the specified channel since the peak level for that channel was previously read.

Syntax

```
HRESULT GetLevel(
  UINT  nChannel,
  float *pfLevel
);
```

Parameters

`nChannel`

The channel number. If the audio stream has *N* channels, the channels are numbered from 0 to *N*– 1. To get the number of channels in the stream, call the IAudioPeakMeter::GetChannelCount method.

`pfLevel`

Pointer to a **float** variable into which the method writes the peak meter level in decibels.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *nChannel* is out of range.
E_P OIN TER	Pointer *pfLevel* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

IAudioPeakMeter Interface

IAudioPeakMeter::GetChannelCount

IAudioTreble interface

1/24/2020 • 2 minutes to read • Edit Online

The **IAudioTreble** interface provides access to a hardware treble-level control. The client obtains a reference to the **IAudioTreble** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioTreble. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioTreble** interface. Only a subunit object that represents a hardware function for controlling the level of the treble frequencies in each channel will support this interface.

The **IAudioTreble** interface provides per-channel controls for setting and getting the gain or attenuation level of the treble frequencies in the audio stream. If a treble-level hardware control can only attenuate the channels in the audio stream, then the maximum treble level for any channel is 0 dB. If a treble-level control can provide gain (amplification), then the maximum treble level is greater than 0 dB.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioTreble** interface provides convenient access to the KSPROPERTY_AUDIO_TREBLE property of a subunit that has a subtype GUID value of KSNODETYPE_TONE. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The IAudioTreble interface inherits from the IPerChannelDbLevel interface.

Methods

The **IAudioTreble** interface has these methods.

METHOD	DESCRIPTION

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IAudioVolumeLevel interface

1/24/2020 • 2 minutes to read • Edit Online

The **IAudioVolumeLevel** interface provides access to a hardware volume control. The client obtains a reference to the **IAudioVolumeLevel** interface of a subunit by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IAudioVolumeLevel. The call to **IPart::Activate** succeeds only if the subunit supports the **IAudioVolumeLevel** interface. Only a subunit object that represents a hardware volume-level control will support this interface.

The **IAudioVolumeLevel** interface provides per-channel controls for setting and getting the gain or attenuation levels in an the audio stream. If a volume-level hardware control can only attenuate the channels in the audio stream, then the maximum volume level for any channel is 0 dB. If a volume-level control can provide gain (amplification), then the maximum volume level is greater than 0 dB.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IAudioVolumeLevel** interface provides convenient access to the KSPROPERTY_AUDIO_VOLUMELEVEL property of a subunit that has a subtype GUID value of KSNODETYPE_VOLUME. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The IAudioVolumeLevel interface inherits from the IPerChannelDbLevel interface.

Methods

The **IAudioVolumeLevel** interface has these methods.

METHOD	DESCRIPTION

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IConnector interface

1/23/2020 • 2 minutes to read • Edit Online

The **IConnector** interface represents a point of connection between components. The client obtains a reference to an **IConnector** interface by calling the IDeviceTopology::GetConnector or IConnector::GetConnectedTo method, or by calling the **IPart::QueryInterface** method with parameter *iid* set to **REFIID** IID_IConnector.

An **IConnector** interface instance can represent:

- An audio jack on a piece of hardware
- An internal connection to an integrated endpoint device (for example, a built-in microphone in a laptop computer)
- A software connection implemented through DMA transfers

The methods in the **IConnector** interface can describe various kinds of connectors. A connector has a type (a ConnectorType enumeration constant) and a subtype (a GUID obtained from the IPart::GetSubType method).

A part in a device topology can be either a connector or a subunit. The IPart interface provides methods that are common to connectors and subunits.

For code examples that use the **IConnector** interface, see the implementations of the GetHardwareDeviceTopology and SelectCaptureDevice functions in Device Topologies.

Inheritance

The **IConnector** interface inherits from the IUnknown interface. **IConnector** also has these types of members:

- Methods

Methods

The **IConnector** interface has these methods.

METHOD	DESCRIPTION
IConnector::ConnectTo	The ConnectTo method connects this connector to a connector in another device-topology object.
IConnector::Disconnect	The Disconnect method disconnects this connector from another connector.
IConnector::GetConnectedTo	The GetConnectedTo method gets the connector to which this connector is connected.
IConnector::GetConnectorIdConnectedTo	The GetConnectorIdConnectedTo method gets the global ID of the connector, if any, that this connector is connected to.
IConnector::GetDataFlow	The GetDataFlow method gets the direction of data flow through this connector.
IConnector::GetDeviceIdConnectedTo	The GetDeviceIdConnectedTo method gets the device identifier of the audio device, if any, that this connector is connected to.

METHOD	DESCRIPTION
IConnector::GetType	The GetType method gets the type of this connector.
IConnector::IsConnected	The IsConnected method indicates whether this connector is connected to another connector.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IConnector::GetConnectedTo

IDeviceTopology::GetConnector

IConnector::ConnectTo method

1/11/2020 • 2 minutes to read • Edit Online

The **ConnectTo** method connects this connector to a connector in another device-topology object.

Syntax

```
HRESULT ConnectTo(
  IConnector *pConnectTo
);
```

Parameters

`pConnectTo`

The other connector. This parameter points to the IConnector interface of the connector object that represents the connector in the other device topology. The caller is responsible for releasing its counted reference to the **IConnector** interface when it is no longer needed. The **ConnectTo** method obtains its own reference to this interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pConnectTo* is **NULL**.
E_INVALIDARG	The current connector and remote connector pointed to by *pConnectTo*, have the same direction of data flow. A connector with data-flow direction "In" must be connected to another connector with data-flow direction "Out" to create a valid connection in the topology. To determine the data flow of a connector, call IConnector::GetDataFlow.
E_NOINTERFACE	The object pointed to by *pConnectTo* is not a valid connector object.

HRE SUL T_F RO M_ WI N32 (ER RO R_D EVI CE_ ALR EAD Y_A TTA CHE D)	One of the two connectors is already attached to another connector. For information about this macro, see the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IConnector Interface

IConnector::Disconnect method

1/11/2020 • 2 minutes to read • Edit Online

The **Disconnect** method disconnects this connector from another connector.

Syntax

```
HRESULT Disconnect();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_NOTFOUND	This connector is already disconnected.
HRESULT_FROM_WIN32(ERROR_FILE_READ_ONLY)	A permanent connection cannot be disconnected. For information about this macro, see the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

IConnector Interface

IConnector::GetConnectedTo method

1/11/2020 • 2 minutes to read • Edit Online

The **GetConnectedTo** method gets the connector to which this connector is connected.

Syntax

```
HRESULT GetConnectedTo(
  IConnector **ppConTo
);
```

Parameters

`ppConTo`

Pointer to a pointer variable into which the method writes the address of the IConnector interface of the other connector object. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetConnectedTo** call fails, **ppConTo* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppConTo* is **NULL**.
E_NOTFOUND	This connector is not connected, or the other side of the connection is not another device topology (for example, a Software_IO connection).

HRE SUL T_F RO M_ WI N32 (ER RO R_P AT H_N OT_ FOU ND)	The device topology on the other side of the connection is not active (that is, the device state is not DEVICE_STATE_ACTIVE).

Remarks

For code examples that call this method, see the implementations of the GetHardwareDeviceTopology and SelectCaptureDevice functions in Device Topologies.

For information about Software_IO connections, see ConnectorType Enumeration. For information about the HRESULT_FROM_WIN32 macro, see the Windows SDK documentation. For information about the DEVICE_STATE_NOTPRESENT device state, see DEVICE_STATE_XXX Constants.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IConnector Interface

IConnector::GetConnectorIdConnectedTo method

1/11/2020 • 2 minutes to read • Edit Online

The **GetConnectorIdConnectedTo** method gets the global ID of the connector, if any, that this connector is connected to.

Syntax

```
HRESULT GetConnectorIdConnectedTo(
  LPWSTR *ppwstrConnectorId
);
```

Parameters

`ppwstrConnectorId`

Pointer to a string pointer into which the method writes the address of a null-terminated, wide-character string that contains the other connector's global ID. The method allocates the storage for the string. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the **GetConnectorIdConnectedTo** call fails, **ppwstrConnectorId* is **NULL**. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_NOTFOUND	This connector is not connected, or the other side of the connection is not another device topology (for example, a Software_IO connection).
E_POINTER	Parameter *ppwstrConnectorId* is **NULL**.
E_OUTOFMEMORY	Out of memory.

Remarks

A global ID is a string that uniquely identifies a part among all parts in all device topologies in the system. Clients should treat this string as opaque. That is, clients should not attempt to parse the contents of the string to obtain information about the part. The reason is that the string format is undefined and might change from one implementation of the DeviceTopology API to the next.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IConnector Interface

IConnector::GetDataFlow method

1/11/2020 • 2 minutes to read • Edit Online

The **GetDataFlow** method gets the direction of data flow through this connector.

Syntax

```
HRESULT GetDataFlow(
  DataFlow *pFlow
);
```

Parameters

`pFlow`

Pointer to a variable into which the method writes the data-flow direction. The direction is one of the following DataFlow enumeration values:

In

Out

If data flows into the device through the connector, the data-flow direction is In. Otherwise, the data-flow direction is Out.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pFlow* is **NULL**.

Remarks

For a code example that calls this method, see the implementation of the SelectCaptureDevice function in Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

IConnector Interface

IConnector::GetDeviceIdConnectedTo method

1/11/2020 • 2 minutes to read • Edit Online

The **GetDeviceIdConnectedTo** method gets the device identifier of the audio device, if any, that this connector is connected to.

Syntax

```
HRESULT GetDeviceIdConnectedTo(
  LPWSTR *ppwstrDeviceId
);
```

Parameters

`ppwstrDeviceId`

Pointer to a string pointer into which the method writes the address of a null-terminated, wide-character string that contains the device identifier of the connected device. The method allocates the storage for the string. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the **GetDeviceIdConnectedTo** call fails, **ppwstrDeviceId* is **NULL**. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppwstrDeviceId* is **NULL**.
E_NOTFOUND	This connector is not connected, or the other side of the connection is not another device topology (for example, a Software_IO connection).
E_MEMORY	Out of memory.

Remarks

The device identifier obtained from this method can be used as an input parameter to the IMMDeviceEnumerator::GetDevice method.

This method is functionally equivalent to, but more efficient than, the following series of method calls:

- Call the IConnector::GetConnectedTo method to obtain the IConnector interface of the "to" connector.
- Call the **IConnector::QueryInterface** method (with parameter *iid* set to **REFIID** IID_IPart) to obtain the IPart interface of the "to" connector.
- Call the IPart::GetTopologyObject method to obtain the IDeviceTopology interface of the "to" device (the device that contains the "to" connector).
- Call the IDeviceTopology::GetDeviceId method to obtain the device ID of the "to" device.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IConnector Interface

IMMDeviceEnumerator::GetDevice

IConnector::GetType method

1/11/2020 • 2 minutes to read • Edit Online

The **GetType** method gets the type of this connector.

Syntax

```
HRESULT GetType(
  ConnectorType *pType
);
```

Parameters

`pType`

Pointer to a variable into which the method writes the connector type. The connector type is one of the following ConnectorType enumeration constants:

Unknown_Connector

Physical_Internal

Physical_External

Software_IO

Software_Fixed

Network

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pType* is **NULL**.

Remarks

A connector corresponds to a "pin" in kernel streaming (KS) terminology. The mapping of KS pins to connectors is as follows:

- If the KS pin communication type is KSPIN_COMMUNICATION_SINK, KSPIN_COMMUNICATION_SOURCE, or KSPIN_COMMUNICATION_BOTH, then the connector type is Software_IO.
- Else, if the pin is part of a physical connection between two KS filters (devices) in the same audio adapter or in different audio adapters, then the connector type is Software_Fixed.

- Else, if the KS pin category is KSNODETYPE_SPEAKER, KSNODETYPE_MICROPHONE, KSNODETYPE_LINE_CONNECTOR, or KSNODETYPE_SPDIF_INTERFACE, the connector type is Physical_External.
- Else, for a pin that does not meet any of the preceding criteria, the connector type is Physical_Internal.

For more information about KS pins, see the Windows DDK documentation.

For a code example that calls the **GetType** method, see the implementation of the SelectCaptureDevice function in Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IConnector Interface

IConnector::IsConnected method

1/11/2020 • 2 minutes to read • Edit Online

The **IsConnected** method indicates whether this connector is connected to another connector.

Syntax

```
HRESULT IsConnected(
  BOOL *pbConnected
);
```

Parameters

`pbConnected`

Pointer to a **BOOL** variable into which the method writes the connection state. If the state is **TRUE**, this connector is connected to another connector. If **FALSE**, this connector is unconnected.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pbConnected* is **NULL**.

Remarks

For a code example that calls the **IsConnected** method, see the implementation of the SelectCaptureDevice function in Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IControlChangeNotify interface

1/23/2020 • 2 minutes to read • Edit Online

The **IControlChangeNotify** interface provides notifications when the status of a part (connector or subunit) changes. Unlike the other interfaces in this section, which are implemented by the DeviceTopology API, the **IControlChangeNotify** interface must be implemented by a client. To receive notifications, the client passes a pointer to its **IControlChangeNotify** interface instance as a parameter to the IPart::RegisterControlChangeCallback method.

After registering its **IControlChangeNotify** interface, the client receives event notifications in the form of callbacks through the **OnNotify** method in the interface.

In implementing the **IControlChangeNotify** interface, the client should observe these rules to avoid deadlocks and undefined behavior:

- The methods in the interface must be nonblocking. The client should never wait on a synchronization object during an event callback.
- The client should never call the IPart::UnregisterControlChangeCallback method during an event callback.
- The client should never release the final reference on an MMDevice API object during an event callback.

Inheritance

The **IControlChangeNotify** interface inherits from the IUnknown interface. **IControlChangeNotify** also has these types of members:

- Methods

Methods

The **IControlChangeNotify** interface has these methods.

METHOD	DESCRIPTION
IControlChangeNotify::OnNotify	The OnNotify method notifies the client when the status of a connector or subunit changes.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::RegisterControlChangeCallback

IPart::UnregisterControlChangeCallback

IControlChangeNotify::OnNotify method

1/11/2020 • 2 minutes to read • Edit Online

The **OnNotify** method notifies the client when the status of a connector or subunit changes.

Syntax

```
HRESULT OnNotify(
  DWORD   dwSenderProcessId,
  LPCGUID pguidEventContext
);
```

Parameters

`dwSenderProcessId`

The process ID of the client that changed the state of the control. If a notification is generated by a hardware event, this process ID will differ from the client's process ID. For more information, see Remarks.

`pguidEventContext`

A pointer to the context GUID for the control-change event. The client that initiates the control change supplies this GUID. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

A client can use this method to keep track of control changes made by other processes and by the hardware. However, a client that changes a control setting can typically disregard the notification that the control change generates. In its implementation of the **OnNotify** method, a client can inspect the *dwSenderProcessId* and *pguidEventContext* parameters to discover whether it or another client is the source of the control-change event.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IControlChangeNotify Interface

IControlInterface interface

1/23/2020 • 2 minutes to read • Edit Online

The **IControlInterface** interface represents a control interface on a part (connector or subunit) in a device topology. The client obtains a reference to a part's **IControlInterface** interface by calling the IPart::GetControlInterface method.

Inheritance

The **IControlInterface** interface inherits from the IUnknown interface. **IControlInterface** also has these types of members:

- Methods

Methods

The **IControlInterface** interface has these methods.

METHOD	DESCRIPTION
IControlInterface::GetIID	The GetIID method gets the interface ID of the function-specific control interface of the part.
IControlInterface::GetName	The GetName method gets the friendly name for the audio function that the control interface encapsulates.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::GetControlInterface

IControlInterface::GetIID method

1/11/2020 • 2 minutes to read • Edit Online

The **GetIID** method gets the interface ID of the function-specific control interface of the part.

Syntax

```
HRESULT GetIID(
  GUID *pIID
);
```

Parameters

`pIID`

Pointer to a GUID variable into which the method writes the interface ID of the function-specific control interface of the part. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pIID* is **NULL**.

Remarks

An object that represents a part (connector or subunit) has two control interfaces. The first is a generic control interface, IControlInterface, which has methods that are common to all types of controls. The second is a function-specific control interface that has methods that apply to a particular type of control. The **GetIID** method gets the interface ID of the second control interface. The client can supply this interface ID to the IPart::Activate method to create an instance of the part's function-specific interface.

The method gets one of the function-specific interface IDs shown in the following table.

INTERFACE ID	INTERFACE NAME
IID_IAudioAutoGainControl	IAudioAutoGainControl
IID_IAudioBass	IAudioBass
IID_IAudioChannelConfig	IAudioChannelConfig
IID_IAudioInputSelector	IAudioInputSelector

IID_IAudioLoudness	IAudioLoudness
IID_IAudioMidrange	IAudioMidrange
IID_IAudioMute	IAudioMute
IID_IAudioOutputSelector	IAudioOutputSelector
IID_IAudioPeakMeter	IAudioPeakMeter
IID_IAudioTreble	IAudioTreble
IID_IAudioVolumeLevel	IAudioVolumeLevel
IID_IDeviceSpecificProperty	IDeviceSpecificProperty
IID_IKsFormatSupport	IKsFormatSupport
IID_IKsJackDescription	IKsJackDescription

To obtain the interface ID of an interface, use the **__uuidof** operator. For example, the interface ID of the **IAudioAutoGainControl** interface is defined as follows:

```
const IID IID_IAudioAutoGainControl  __uuidof(IAudioAutoGainControl)
```

For more information about the **__uuidof** operator, see the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IControlInterface Interface

IControlInterface::GetName method

1/11/2020 • 2 minutes to read • Edit Online

The **GetName** method gets the friendly name for the audio function that the control interface encapsulates.

Syntax

```
HRESULT GetName(
  LPWSTR *ppwstrName
);
```

Parameters

`ppwstrName`

Pointer to a string pointer into which the method writes the address of a null-terminated, wide-character string that contains the friendly name. The method allocates the storage for the string. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the **GetName** call fails, **ppwstrName* is **NULL**. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppwstrName* is **NULL**.
E_OUTOFMEMORY	Out of memory.

Remarks

As an example of a friendly name, a subunit with an IAudioPeakMeter interface might have the friendly name "peak meter".

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioPeakMeter Interface

IControlInterface Interface

IDeviceSpecificProperty interface

1/23/2020 • 2 minutes to read • Edit Online

The **IDeviceSpecificProperty** interface provides access to the control value of a device-specific hardware control. A client obtains a reference to an **IDeviceSpecificProperty** interface of a part by calling the IPart::Activate method with parameter *refiid* set to **REFIID** IID_IDeviceSpecificProperty. The call to **IPart::Activate** succeeds only if the part supports the **IDeviceSpecificProperty** interface. A part supports this interface only if the underlying hardware control has a device-specific control value and the control cannot be adequately represented by any other interface in the DeviceTopology API.

Typically, a device-specific property is useful only to a client that can infer the meaning of the property value from information such as the part type, part subtype, and part name. The client can obtain this information by calling the IPart::GetPartType, IPart::GetSubType, and IPart::GetName methods.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware control parameters in subunits (referred to as KS nodes). The **IDeviceSpecificProperty** interface provides convenient access to the KSPROPERTY_AUDIO_DEV_SPECIFIC property of a subunit that has a subtype GUID value of KSNODETYPE_DEV_SPECIFIC. To obtain the subtype GUID of a subunit, call the IPart::GetSubType method. For more information about KS properties and KS node types, see the Windows DDK documentation.

Inheritance

The **IDeviceSpecificProperty** interface inherits from the IUnknown interface. **IDeviceSpecificProperty** also has these types of members:

- Methods

Methods

The **IDeviceSpecificProperty** interface has these methods.

METHOD	DESCRIPTION
IDeviceSpecificProperty::Get4BRange	The Get4BRange method gets the 4-byte range of the device-specific property value.
IDeviceSpecificProperty::GetType	The GetType method gets the data type of the device-specific property value.
IDeviceSpecificProperty::GetValue	The GetValue method gets the current value of the device-specific property.
IDeviceSpecificProperty::SetValue	The SetValue method sets the value of the device-specific property.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IPart::GetName

IPart::GetPartType

IPart::GetSubType

IDeviceSpecificProperty::Get4BRange method

1/11/2020 • 2 minutes to read • Edit Online

The **Get4BRange** method gets the 4-byte range of the device-specific property value.

Syntax

```
HRESULT Get4BRange(
  LONG *plMin,
  LONG *plMax,
  LONG *plStepping
);
```

Parameters

`plMin`

Pointer to a **LONG** variable into which the method writes the minimum property value.

`plMax`

Pointer to a **LONG** variable into which the method writes the maximum property value.

`plStepping`

Pointer to a **LONG** variable into which the method writes the stepping value between consecutive property values in the range **plMin* to **plMax*. If the difference between the maximum and minimum property values is *d*, and the range is divided into *n* steps (uniformly sized intervals), then the property can take *n* + 1 discrete values and the size of the step between consecutive values is d / n.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *plMin*, *plMax*, or *plStepping* is **NULL**.

HRE SUL T_F RO M_ WI N32 (ER RO R_N OT_ SUP PO RTE D)	The property value is not a 32-bit signed or unsigned integer. For information about this macro, see the Windows SDK documentation.

Remarks

This method reports the range and step size for a property value that is a 32-bit signed or unsigned integer. These two data types are represented by **VARENUM** enumeration constants VT_I4 and VT_UI4, respectively. If the property value is not a 32-bit integer, then the method returns an error status code. For more information about **VARENUM**, see the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IDeviceSpecificProperty Interface

IDeviceSpecificProperty::GetType method

1/11/2020 • 2 minutes to read • Edit Online

The **GetType** method gets the data type of the device-specific property value.

Syntax

```
HRESULT GetType(
  VARTYPE *pVType
);
```

Parameters

`pVType`

Pointer to a **VARTYPE** variable into which the method writes a **VARTYPE** enumeration value that indicates the data type of the device-specific property value. For more information about **VARTYPE** and **VARTYPE**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pVType* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IDeviceSpecificProperty Interface

IDeviceSpecificProperty::GetValue method

1/11/2020 • 2 minutes to read • Edit Online

The **GetValue** method gets the current value of the device-specific property.

Syntax

```
HRESULT GetValue(
  void  *pvValue,
  DWORD *pcbValue
);
```

Parameters

`pvValue`

Pointer to a caller-allocated buffer into which the method writes the property value.

`pcbValue`

[inout] Pointer to a **DWORD** variable that specifies the size in bytes of the property value. On entry, **pcbValue* contains the size of the caller-allocated buffer (or 0 if *pvValue* is **NULL**). Before returning, the method writes the actual size of the property value written to the buffer (or the required size if the buffer is too small or if *pvValue* is **NULL**).

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pcbValue* is **NULL**.

HRE **SUL** **T_F** **RO** **M_** **WI** **N32** **(ER** **RO** **R_I** **NS** **UFFI** **CIE** **NT_** **BUF** **FER)**	The buffer pointed to by parameter *pvValue* is too small to contain the property value, or *pvValue* is **NULL** and the size of the property value is fixed rather than variable. For information about this macro, see the Windows SDK documentation.

Remarks

If the size of the property value is variable rather than fixed, the caller can obtain the required buffer size by calling **GetValue** with parameter *pvValue* = **NULL** and **pcbValue* = 0. The method writes the required buffer size to **pcbValue*. With this information, the caller can allocate a buffer of the required size and call **GetValue** a second time to obtain the property value.

If the caller-allocated buffer is too small to hold the property value, **GetValue** writes the required buffer size to **pcbValue* and returns an error status code. In this case, it writes nothing to the buffer pointed by *pvValue*.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IDeviceSpecificProperty Interface

IDeviceSpecificProperty::SetValue method

1/11/2020 • 2 minutes to read • Edit Online

The **SetValue** method sets the value of the device-specific property.

Syntax

```
HRESULT SetValue(
  void    *pvValue,
  DWORD   cbValue,
  LPCGUID pguidEventContext
);
```

Parameters

`pvValue`

Pointer to the new value for the device-specific property.

`cbValue`

The size in bytes of the device-specific property value.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetValue** call changes the state of the control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pvValue* is **NULL**.
E_INVALIDARG	Parameter *cbValue* does not match the required size of the property value.

E_OUTOFMEMORY	Out of memory.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IDeviceSpecificProperty Interface

IDeviceTopology interface

1/23/2020 • 2 minutes to read • Edit Online

The **IDeviceTopology** interface provides access to the topology of an audio device. The topology of an audio *adapter* device consists of the data paths that lead to and from audio endpoint devices and the control points that lie along the paths. An audio *endpoint* device also has a topology, but it is trivial, as explained in Device Topologies. A client obtains a reference to the **IDeviceTopology** interface for an audio endpoint device by following these steps:

1. By using one of the techniques described in IMMDevice Interface, obtain a reference to the **IMMDevice** interface for an audio endpoint device.
2. Call the IMMDevice::Activate method with parameter *refiid* set to **REFIID** IID_IDeviceTopology.

After obtaining the **IDeviceTopology** interface for an audio endpoint device, an application can explore the topologies of the audio adapter devices to which the endpoint device is connected.

For code examples that use the **IDeviceTopology** interface, see the implementations of the GetHardwareDeviceTopology and SelectCaptureDevice functions in Device Topologies.

Inheritance

The **IDeviceTopology** interface inherits from the IUnknown interface. **IDeviceTopology** also has these types of members:

- Methods

Methods

The **IDeviceTopology** interface has these methods.

METHOD	DESCRIPTION
IDeviceTopology::GetConnector	The GetConnector method gets the connector that is specified by a connector number.
IDeviceTopology::GetConnectorCount	The GetConnectorCount method gets the number of connectors in the device-topology object.
IDeviceTopology::GetDeviceId	The GetDeviceId method gets the device identifier of the device that is represented by the device-topology object.
IDeviceTopology::GetPartById	The GetPartById method gets a part that is identified by its local ID.
IDeviceTopology::GetSignalPath	The GetSignalPath method gets a list of parts in the signal path that links two parts, if the path exists.
IDeviceTopology::GetSubunit	The GetSubunit method gets the subunit that is specified by a subunit number.
IDeviceTopology::GetSubunitCount	The GetSubunitCount method gets the number of subunits in the device topology.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IMMDevice::Activate

IDeviceTopology::GetConnector method

1/11/2020 • 2 minutes to read • Edit Online

The **GetConnector** method gets the connector that is specified by a connector number.

Syntax

```
HRESULT GetConnector(
  UINT       nIndex,
  IConnector **ppConnector
);
```

Parameters

`nIndex`

The connector number. If a device topology contains n connectors, the connectors are numbered 0 to n – 1. To get the number of connectors in the device topology, call the IDeviceTopology::GetConnectorCount method.

`ppConnector`

Pointer to a pointer variable into which the method writes the address of the IConnector interface of the connector object. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetConnector** call fails, **ppConnector* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDA RG	Parameter *nIndex* is out of range.
E_POINTER	Pointer *ppConnector* is **NULL**.

Remarks

For code examples that call the **GetConnector** method, see the implementations of the GetHardwareDeviceTopology and SelectCaptureDevice functions in Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IConnector Interface

IDeviceTopology Interface

IDeviceTopology::GetConnectorCount

IDeviceTopology::GetConnectorCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetConnectorCount** method gets the number of connectors in the device-topology object.

Syntax

```
HRESULT GetConnectorCount(
  UINT *pCount
);
```

Parameters

`pCount`

Pointer to a **UINT** pointer variable into which the method writes the connector count (the number of connectors in the device topology).

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pCount* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IDeviceTopology Interface

IDeviceTopology::GetDeviceId method

1/11/2020 • 2 minutes to read • Edit Online

The **GetDeviceId** method gets the device identifier of the device that is represented by the device-topology object.

Syntax

```
HRESULT GetDeviceId(
  LPWSTR *ppwstrDeviceId
);
```

Parameters

`ppwstrDeviceId`

Pointer to a pointer variable into which the method writes the address of a null-terminated, wide-character string that contains the device identifier. The method allocates the storage for the string. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the **GetDeviceId** call fails, **ppwstrDeviceId* is **NULL**. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
D_POINTER	Pointer *ppwstrDeviceId* is **NULL**.
E_OUTOFMEMORY	Out of memory.

Remarks

The device identifier obtained from this method can be used as an input parameter to the IMMDeviceEnumerator::GetDevice method.

For a code example that uses the **GetDeviceId** method, see Using the IKsControl Interface to Access Audio Properties.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IDeviceTopology Interface

IMMDeviceEnumerator::GetDevice

IDeviceTopology::GetPartById method

1/11/2020 • 2 minutes to read • Edit Online

The **GetPartById** method gets a part that is identified by its local ID.

Syntax

```
HRESULT GetPartById(
  UINT  nId,
  IPart **ppPart
);
```

Parameters

`nId`

The part to get. This parameter is the local ID of the part. For more information, see Remarks.

`ppPart`

Pointer to a pointer variable into which the method writes the address of the IPart interface of the part object that is identified by *nId*. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetPartById** call fails, **ppPart* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDA RG	Parameter *nId* is not a valid local ID.
E_POINTER	Pointer *ppPart* is **NULL**.

Remarks

A local ID is a number that uniquely identifies a part among all the parts in a device topology. The IAudioInputSelector::GetSelection and IAudioOutputSelector::GetSelection methods retrieve the local ID of a connected part. The IAudioInputSelector::SetSelection and IAudioOutputSelector::SetSelection methods select the input or output that is connected to a part that is identified by its local ID. When you have a pointer to a part object, you can call the IPart::GetLocalId method to get the local ID of the part.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioInputSelector::GetSelection

IAudioInputSelector::SetSelection

IAudioOutputSelector::GetSelection

IAudioOutputSelector::SetSelection

IDeviceTopology Interface

IPart Interface

IPart::GetLocalId

IDeviceTopology::GetSignalPath method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSignalPath** method gets a list of parts in the signal path that links two parts, if the path exists.

Syntax

```
HRESULT GetSignalPath(
  IPart      *pIPartFrom,
  IPart      *pIPartTo,
  BOOL       bRejectMixedPaths,
  IPartsList **ppParts
);
```

Parameters

`pIPartFrom`

Pointer to the "from" part. This parameter is a pointer to the IPart interface of the part at the beginning of the signal path.

`pIPartTo`

Pointer to the "to" part. This parameter is a pointer to the **IPart** interface of the part at the end of the signal path.

`bRejectMixedPaths`

Specifies whether to reject paths that contain mixed data. If *bRejectMixedPaths* is **TRUE** (nonzero), the method ignores any data path that contains a mixer (that is, a processing node that sums together two or more input signals). If **FALSE**, the method will try to find a path that connects the "from" and "to" parts regardless of whether the path contains a mixer.

`ppParts`

Pointer to a pointer variable into which the method writes the address of an IPartsList interface instance. This interface encapsulates the list of parts in the signal path that connects the "from" part to the "to" part. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetSignalPath** call fails, **ppParts* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pIPartFrom*, *pIPartTo*, or *ppParts* is **NULL**.

E_NOTFOUND	No path linking the two parts was found.
E_NOINTERFACE	Parameter *pIPartFrom* or *pIPartTo* does not point to a valid **IPart** interface.
E_OUTOFMEMORY	Out of memory.

Remarks

This method creates an **IPartsList** interface instance that contains a list of the parts that lie along the specified signal path. The parts in the parts list are ordered according to their relative positions in the signal path. The "to" part is the first item in the list and the "from" part is the last item in the list.

If the list contains *n* parts, the "to" and "from" parts are identified by list indexes 0 and *n*– 1, respectively. To get the number of parts in a parts list, call the IPartsList::GetCount method. To retrieve a part by its index, call the IPartsList::GetPart method.

The parts in the signal path must all be part of the same device topology. The path cannot span boundaries between device topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IDeviceTopology Interface

IPart Interface

IPartsList Interface

IPartsList::GetCount

IDeviceTopology::GetSubunit method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSubunit** method gets the subunit that is specified by a subunit number.

Syntax

```
HRESULT GetSubunit(
  UINT     nIndex,
  ISubunit **ppSubunit
);
```

Parameters

`nIndex`

The subunit number. If a device topology contains *n* subunits, the subunits are numbered from 0 to *n*– 1. To get the number of subunits in the device topology, call the IDeviceTopology::GetSubunitCount method.

`ppSubunit`

Pointer to a pointer variable into which the method writes the address of the ISubunit interface of the subunit object. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetSubunit** call fails, **ppSubunit* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDA RG	Parameter *nIndex* is out of range.
E_POINTER	Pointer *ppSubunit* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]

Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IDeviceTopology Interface

IDeviceTopology::GetSubunitCount

ISubunit Interface

IDeviceTopology::GetSubunitCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSubunitCount** method gets the number of subunits in the device topology.

Syntax

```
HRESULT GetSubunitCount(
  UINT *pCount
);
```

Parameters

`pCount`

Pointer to a **UINT** variable into which the method writes the subunit count (the number of subunits in the device topology).

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pCount* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IDeviceTopology Interface

IKsFormatSupport interface

1/23/2020 • 2 minutes to read • Edit Online

The **IKsFormatSupport** interface provides information about the audio data formats that are supported by a software-configured I/O connection (typically a DMA channel) between an audio adapter device and system memory. The client obtains a reference to the **IKsFormatSupport** interface of a part by calling the IPart::Activate method with parameter *refiid* set to REFIID IID_IKsFormatSupport. The call to **IPart::Activate** succeeds only if the part supports the **IKsFormatSupport** interface. Only a part object that represents a connector with a Software_IO connection type will support this interface. For more information about Software_IO, see ConnectorType Enumeration.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware description parameters in connectors (referred to as KS pins). The **IKsFormatSupport** interface provides convenient access to the KSPROPERTY_PIN_DATAINTERSECTION and KSPROPERTY_PIN_PROPOSEDDATAFORMAT properties of a connector to a system bus (typically, PCI or PCI Express) or an external bus (for example, USB). Not all drivers support the KSPROPERTY_PIN_PROPOSEDDATAFORMAT property. If a driver does not support this property, **IKsFormatSupport** uses the information in the KS data ranges for the connector to determine whether the connector supports the proposed format. For more information about KS properties, KS pins, and KS data ranges, see the Windows DDK documentation.

Inheritance

The **IKsFormatSupport** interface inherits from the IUnknown interface. **IKsFormatSupport** also has these types of members:

- Methods

Methods

The **IKsFormatSupport** interface has these methods.

METHOD	DESCRIPTION
IKsFormatSupport::GetDevicePreferredFormat	The GetDevicePreferredFormat method gets the preferred audio stream format for the connection.
IKsFormatSupport::IsFormatSupported	The IsFormatSupported method indicates whether the audio endpoint device supports the specified audio stream format.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IKsFormatSupport::GetDevicePreferredFormat method

1/11/2020 • 2 minutes to read • Edit Online

The **GetDevicePreferredFormat** method gets the preferred audio stream format for the connection.

Syntax

```
HRESULT GetDevicePreferredFormat(
  PKSDATAFORMAT *ppKsFormat
);
```

Parameters

`ppKsFormat`

Pointer to a pointer variable into which the method writes the address of a buffer that contains the format specifier for the preferred format. The specifier begins with a **KSDATAFORMAT** structure that might be followed by additional format information. The method allocates the storage for the format specifier. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the method fails, **ppKsFormat* is **NULL**. For more information about **KSDATAFORMAT**, format specifiers, and **CoTaskMemFree**, see the Windows DDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppKsFormat* is **NULL**.
E_OUTOFMEMORY	Out of memory.

Requirements

Minimum supported client	Windows Vista [desktop apps only]

Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IKsFormatSupport Interface

IKsFormatSupport::IsFormatSupported method

1/11/2020 • 2 minutes to read • Edit Online

The **IsFormatSupported** method indicates whether the audio endpoint device supports the specified audio stream format.

Syntax

```
HRESULT IsFormatSupported(
  PKSDATAFORMAT pKsFormat,
  DWORD         cbFormat,
  BOOL          *pbSupported
);
```

Parameters

`pKsFormat`

Pointer to an audio-stream format specifier. This parameter points to a caller-allocated buffer that contains a format specifier. The specifier begins with a KSDATAFORMAT structure that might be followed by additional format information. For more information about **KSDATAFORMAT** and format specifiers, see the Windows DDK documentation.

`cbFormat`

The size in bytes of the buffer that contains the format specifier.

`pbSupported`

Pointer to a **BOOL** variable into which the method writes a value to indicate whether the format is supported. The method writes **TRUE** if the device supports the format and **FALSE** if the device does not support the format.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pKsFormat* or *pbSupported* is **NULL**.
E_INVALIDARG	The format specifier is not valid.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IKsFormatSupport Interface

IKsJackDescription interface

1/23/2020 • 3 minutes to read • Edit Online

The **IKsJackDescription** interface provides information about the jacks or internal connectors that provide a physical connection between a device on an audio adapter and an external or internal endpoint device (for example, a microphone or CD player). The client obtains a reference to the **IKsJackDescription** interface of a part by calling the IPart::Activate method with parameter *refiid* set to **REFIID** IID_IKsJackDescription. The call to **IPart::Activate** succeeds only if the part supports the **IKsJackDescription** interface. Only a part object that represents a connector with a Physical_External or Physical_Internal connection type will support this interface.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware description parameters in connectors (referred to as KS pins). The **IKsJackDescription** interface provides convenient access to the KSPROPERTY_JACK_DESCRIPTION property of a connector to an endpoint device. For more information about KS properties and KS pins, see the Windows DDK documentation.

Inheritance

The **IKsJackDescription** interface inherits from the IUnknown interface. **IKsJackDescription** also has these types of members:

- Methods

Methods

The **IKsJackDescription** interface has these methods.

METHOD	DESCRIPTION
IKsJackDescription::GetJackCount	The GetJackCount method gets the number of jacks required to connect to an audio endpoint device.
IKsJackDescription::GetJackDescription	The GetJackDescription method gets a description of an audio jack.

Remarks

If an audio endpoint device supports the **IKsJackDescription** interface, the Windows multimedia control panel, Mmsys.cpl, displays the jack information. To view the jack information, follow these steps:

1. To run Mmsys.cpl, open a Command Prompt window and enter the following command:

   control mmsys.cpl

   Alternatively, you can run Mmsys.cpl by right-clicking the speaker icon in the notification area, which is located on the right side of the taskbar, and selecting either **Playback Devices** or **Recording Devices**.

2. After the Mmsys.cpl window opens, select a device from either the list of playback devices or the list of recording devices, and click **Properties**.
3. When the properties window opens, click **General**. If the selected property page displays the jack information for the device, the device supports the **IKsJackDescription** interface. If the property page displays the text "No jack information is available", the device does not support the interface.

The following code example shows how to obtain the **IKsJackDescription** interface for an audio endpoint device:

```
//-----------------------------------------------------------
// Get the IKsJackDescription interface that describes the
// audio jack or jacks that the endpoint device plugs into.
//-----------------------------------------------------------
#define EXIT_ON_ERROR(hres)  \
              if (FAILED(hres)) { goto Exit; }
#define SAFE_RELEASE(punk)  \
              if ((punk) != NULL)  \
                { (punk)->Release(); (punk) = NULL; }

HRESULT GetJackInfo(IMMDevice *pDevice,
                    IKsJackDescription **ppJackDesc)
{
    HRESULT hr = S_OK;
    IDeviceTopology *pDeviceTopology = NULL;
    IConnector *pConnFrom = NULL;
    IConnector *pConnTo = NULL;
    IPart *pPart = NULL;
    IKsJackDescription *pJackDesc = NULL;

    if (NULL != ppJackDesc)
    {
        *ppJackDesc = NULL;
    }
    if (NULL == pDevice || NULL == ppJackDesc)
    {
        return E_POINTER;
    }

    // Get the endpoint device's IDeviceTopology interface.
    hr = pDevice->Activate(__uuidof(IDeviceTopology), CLSCTX_ALL,
                           NULL, (void**)&pDeviceTopology);
    EXIT_ON_ERROR(hr)

    // The device topology for an endpoint device always
    // contains just one connector (connector number 0).
    hr = pDeviceTopology->GetConnector(0, &pConnFrom);
    EXIT_ON_ERROR(hr)

    // Step across the connection to the jack on the adapter.
    hr = pConnFrom->GetConnectedTo(&pConnTo);
    if (HRESULT_FROM_WIN32(ERROR_PATH_NOT_FOUND) == hr)
    {
        // The adapter device is not currently active.
        hr = E_NOINTERFACE;
    }
    EXIT_ON_ERROR(hr)

    // Get the connector's IPart interface.
    hr = pConnTo->QueryInterface(__uuidof(IPart), (void**)&pPart);
    EXIT_ON_ERROR(hr)

    // Activate the connector's IKsJackDescription interface.
    hr = pPart->Activate(CLSCTX_INPROC_SERVER,
                         __uuidof(IKsJackDescription), (void**)&pJackDesc);
    EXIT_ON_ERROR(hr)

    *ppJackDesc = pJackDesc;

Exit:
    SAFE_RELEASE(pDeviceTopology)
    SAFE_RELEASE(pConnFrom)
    SAFE_RELEASE(pConnTo)
    SAFE_RELEASE(pPart)
    return hr;
}
```

In the preceding code example, the GetJackInfo function takes two parameters. Input parameter *pDevice* points to the IMMDevice interface of an endpoint device. Output parameter *ppJackDesc* points to a pointer value into which the function writes the address of the corresponding **IKsJackDescription** interface, if the interface exists. If the interface does not exist, the function writes **NULL** to **ppJackDesc* and returns error code E_NOINTERFACE.

In the preceding code example, the call to IMMDevice::Activate retrieves the IDeviceTopology interface of the endpoint device. The device topology of an endpoint device contains a single connector (connector number 0) that connects to the adapter device. At the other side of this connection, the connector on the adapter device represents the audio jack or jacks that the endpoint device plugs into. The call to the IDeviceTopology::GetConnector method retrieves the IConnector interface of the connector on the endpoint device, and the IConnector::GetConnectedTo method call retrieves the corresponding connector on the adapter device. Finally, the **IConnector::QueryInterface** method call retrieves the IPart interface of the adapter device's connector, and the IPart::Activate method call retrieves the connector's **IKsJackDescription** interface, if it exists.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IKsJackDescription::GetJackCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetJackCount** method gets the number of jacks required to connect to an audio endpoint device.

Syntax

```
HRESULT GetJackCount(
  UINT *pcJacks
);
```

Parameters

`pcJacks`

Pointer to a **UINT** variable into which the method writes the number of jacks associated with the connector.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pcJacks* is **NULL**.

Remarks

An audio endpoint device that plays or records a stream that contains multiple channels might require a connection with more than one jack (physical connector).

For example, a set of surround speakers that plays a 6-channel audio stream might require three stereo jacks. In this example, the first jack transmits the channels for the front-left and front-right speakers, the second jack transmits the channels for the front-center and low-frequency-effects (subwoofer) speakers, and the third jack transmits the channels for the side-left and side-right speakers.

After calling this method to retrieve the jack count, call the IKsJackDescription::GetJackDescription method once for each jack to obtain a description of the jack.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

IKsJackDescription Interface

IKsJackDescription::GetJackDescription

IKsJackDescription::GetJackDescription method

1/11/2020 • 2 minutes to read • Edit Online

The **GetJackDescription** method gets a description of an audio jack.

Syntax

```
HRESULT GetJackDescription(
  UINT               nJack,
  KSJACK_DESCRIPTION *pDescription
);
```

Parameters

`nJack`

The jack index. If the connection consists of *n* jacks, the jacks are numbered from 0 to *n*– 1. To get the number of jacks, call the IKsJackDescription::GetJackCount method.

`pDescription`

Pointer to a caller-allocated buffer into which the method writes a structure of type KSJACK_DESCRIPTION that contains information about the jack. The buffer size must be at least sizeof(KSJACK_DESCRIPTION).

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *nJack* is not a valid jack index.
E_P OIN TER	Pointer *pDescription* is **NULL**.

Remarks

When a user needs to plug an audio endpoint device into a jack or unplug it from a jack, an audio application can use the descriptive information that it retrieves from this method to help the user to find the jack. This information includes:

- The physical location of the jack on the computer chassis or external box.
- The color of the jack.
- The type of physical connector used for the jack.

- The mapping of channels to the jack.

For more information, see KSJACK_DESCRIPTION.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IKsJackDescription Interface

IKsJackDescription::GetJackCount

KSJACK_DESCRIPTION

IKsJackDescription2 interface

1/23/2020 • 2 minutes to read • Edit Online

The **IKsJackDescription2** interface provides information about the jacks or internal connectors that provide a physical connection between a device on an audio adapter and an external or internal endpoint device (for example, a microphone or CD player).

In addition to getting jack information such as type of connection, the IKsJackDescription is primarily used to report whether the jack was connected to the device. In Windows 7, if the connected device driver supports **IKsJackDescription2**, the audio stack or an application can use this interface to get information additional jack information. This includes the jack's detection capability and if the format of the device has changed dynamically.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware description parameters in connectors (referred to as KS pins). The **IKsJackDescription2** interface provides convenient access to the **KSPROPERTY_JACK_DESCRIPTION2** property of a connector to an endpoint device. For more information about KS properties and KS pins, see the Windows DDK documentation.

An application obtains a reference to the **IKsJackDescription2** interface of a part by calling the IPart::Activate method with parameter *refiid* set to **REFIIDIID_IKsJackDescription2**. The call to **IPart::Activate** succeeds only if the part supports the **IKsJackDescription2** interface. Only a part object that represents a bridge pin connector on a KS filter device topology object supports this interface.

For a code example, see IKsJackDescription.

Inheritance

The **IKsJackDescription2** interface inherits from the IUnknown interface. **IKsJackDescription2** also has these types of members:

- Methods

Methods

The **IKsJackDescription2** interface has these methods.

METHOD	DESCRIPTION
IKsJackDescription2::GetJackCount	The GetJackCount method gets the number of jacks on the connector, which are required to connect to an endpoint device.
IKsJackDescription2::GetJackDescription2	The GetJackDescription2 method gets the description of a specified audio jack.

Requirements

Minimum supported client	Windows 7 [desktop apps only]

Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IPart::Activate

IKsJackDescription2::GetJackCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetJackCount** method gets the number of jacks on the connector, which are required to connect to an endpoint device.

Syntax

```
HRESULT GetJackCount(
  UINT *pcJacks
);
```

Parameters

`pcJacks`

Receives the number of audio jacks associated with the connector.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pcJacks* is **NULL**.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IKsJackDescription2

IKsJackDescription2::GetJackDescription2 method

1/11/2020 • 2 minutes to read • Edit Online

The **GetJackDescription2** method gets the description of a specified audio jack.

Syntax

```
HRESULT GetJackDescription2(
  UINT               nJack,
  KSJACK_DESCRIPTION2 *pDescription2
);
```

Parameters

`nJack`

The index of the jack to get a description for. If the connection consists of *n* jacks, the jacks are numbered from 0 to *n*– 1. To get the number of jacks, call the IKsJackDescription::GetJackCount method.

`pDescription2`

Pointer to a caller-allocated buffer into which the method writes a structure of type KSJACK_DESCRIPTION2 that contains information about the jack. The buffer size must be at least `sizeof(KSJACK_DESCRIPTION2)`.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *nJack* is not a valid jack index.
E_P OIN TER	Pointer *pDescription* is **NULL**.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

IKsJackDescription2

IKsJackSinkInformation interface

1/23/2020 • 2 minutes to read • Edit Online

The **IKsJackSinkInformation** interface provides access to jack sink information if the jack is supported by the hardware.

The client obtains a reference to the **IKsJackSinkInformation** interface by activating it on the IPart interface of a bridge pin connector on a KS filter device topology object. To activate the object, call the IPart::Activate method with parameter refiid set to REFIID **IID_IKsJackSinkInformation**.

Most Windows audio adapter drivers support the Windows Driver Model (WDM) and use kernel-streaming (KS) properties to represent the hardware description parameters in connectors (referred to as KS pins). The **IKsJackSinkInformation** interface provides convenient access to the **KSPROPERTY_JACK_SINK_INFO** property of a connector to an endpoint device. For more information about KS properties and KS pins, see the Windows DDK documentation.

Inheritance

The **IKsJackSinkInformation** interface inherits from the IUnknown interface. **IKsJackSinkInformation** also has these types of members:

- Methods

Methods

The **IKsJackSinkInformation** interface has these methods.

METHOD	DESCRIPTION
IKsJackSinkInformation::GetJackSinkInformation	The GetJackSinkInformation method retrieves the sink information for the specified jack.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IKsJackDescription

IKsJackSinkInformation::GetJackSinkInformation method

1/11/2020 • 2 minutes to read • Edit Online

The **GetJackSinkInformation** method retrieves the sink information for the specified jack.

Syntax

```
HRESULT GetJackSinkInformation(
  KSJACK_SINK_INFORMATION *pJackSinkInformation
);
```

Parameters

`pJackSinkInformation`

Pointer to a caller-allocated buffer that receives the sink information of the jack in a KSJACK_SINK_INFORMATION structure. The buffer size must be at least `sizeof(KSJACK_SINK_INFORMATION)`.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDA_RG	Parameter *nJack* is not a valid jack index.
E_POINTER	Pointer *pDescription* is **NULL**.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IKsJackSinkInformation

IPart interface

1/23/2020 • 2 minutes to read • Edit Online

The **IPart** interface represents a part (connector or subunit) of a device topology. A client obtains a reference to an **IPart** interface by calling the IDeviceTopology::GetPartById or IPartsList::GetPart method, or by calling the **QueryInterface** method of the IConnector or ISubunit interface on a part object and setting the method's *iid* parameter to **REFIID** IID_IPart.

An object with an **IPart** interface can encapsulate one of the following device topology parts:

- **Connector.** This is a part that connects to another device to form a data path for transmitting an audio stream between devices.
- **Subunit.** This is a part that processes an audio stream (for example, volume control).

The **IPart** interface of a connector or subunit object represents the generic functions that are common to all parts, and the object's **IConnector** or **ISubunit** interface represents the functions that are specific to a connector or subunit. In addition, a part might support one or more control interfaces for controlling or monitoring the function of the part. For example, the client controls a volume-control subunit through its IAudioVolumeLevel interface.

The **IPart** interface provides methods for getting the name, local ID, global ID, and part type of a connector or subunit. In addition, **IPart** can activate a control interface on a connector or subunit.

For code examples that use the **IPart** interface, see the implementations of the GetHardwareDeviceTopology and SelectCaptureDevice functions in Device Topologies.

Inheritance

The **IPart** interface inherits from the IUnknown interface. **IPart** also has these types of members:

- Methods

Methods

The **IPart** interface has these methods.

METHOD	DESCRIPTION
IPart::Activate	The Activate method activates a function-specific interface on a connector or subunit.
IPart::EnumPartsIncoming	The EnumPartsIncoming method gets a list of all the incoming parts—that is, the parts that reside on data paths that are upstream from this part.
IPart::EnumPartsOutgoing	The EnumPartsOutgoing method retrieves a list of all the outgoing parts—that is, the parts that reside on data paths that are downstream from this part.
IPart::GetControlInterface	The GetControlInterface method gets a reference to the specified control interface, if this part supports it.
IPart::GetControlInterfaceCount	The GetControlInterfaceCount method gets the number of control interfaces that this part supports.

METHOD	DESCRIPTION
IPart::GetGlobalId	The GetGlobalId method gets the global ID of this part.
IPart::GetLocalId	The GetLocalId method gets the local ID of this part.
IPart::GetName	The GetName method gets the friendly name of this part.
IPart::GetPartType	The GetPartType method gets the part type of this part.
IPart::GetSubType	The GetSubType method gets the part subtype of this part.
IPart::GetTopologyObject	The GetTopologyObject method gets a reference to the IDeviceTopology interface of the device-topology object that contains this part.
IPart::RegisterControlChangeCallback	The RegisterControlChangeCallback method registers the IControlChangeNotify interface, which the client implements to receive notifications of status changes in this part.
IPart::UnregisterControlChangeCallback	The UnregisterControlChangeCallback method removes the registration of an IControlChangeNotify interface that the client previously registered by a call to the IPart::RegisterControlChangeCallback method.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IAudioVolumeLevel Interface

IConnector Interface

IDeviceTopology::GetPartById

IPartsList::GetPart

ISubunit Interface

IPart::Activate method

1/11/2020 • 2 minutes to read • Edit Online

The **Activate** method activates a function-specific interface on a connector or subunit.

Syntax

```
HRESULT Activate(
  DWORD  dwClsContext,
  REFIID refiid,
  void   **ppvObject
);
```

Parameters

`dwClsContext`

The execution context in which the code that manages the newly created object will run. The caller can restrict the context by setting this parameter to the bitwise **OR** of one or more **CLSCTX** enumeration values. The client can avoid imposing any context restrictions by specifying CLSCTX_ALL. For more information about **CLSCTX**, see the Windows SDK documentation.

`refiid`

The interface ID for the requested control function. The client should set this parameter to one of the following **REFIID** values:

IID_IAudioAutoGainControl

IID_IAudioBass

IID_IAudioChannelConfig

IID_IAudioInputSelector

IID_IAudioLoudness

IID_IAudioMidrange

IID_IAudioMute

IID_IAudioOutputSelector

IID_IAudioPeakMeter

IID_IAudioTreble

IID_IAudioVolumeLevel

IID_IDeviceSpecificProperty

IID_IKsFormatSupport

IID_IKsJackDescription

For more information, see Remarks.

`ppvObject`

Pointer to a pointer variable into which the method writes the address of the interface that is specified by parameter *refiid*. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **Activate** call fails, **ppObject* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	The CLSCTX_INPROC_SERVER bit in *dwClsContext* is zero.
E_P OIN TER	Pointer *ppvObject* is **NULL**.
E_N OIN TER FAC E	The part object does not support the requested interface.

Remarks

The **Activate** method supports the following function-specific control interfaces:

- IAudioAutoGainControl
- IAudioBass
- IAudioChannelConfig
- IAudioInputSelector
- IAudioLoudness
- IAudioMidrange
- IAudioMute
- IAudioOutputSelector
- IAudioPeakMeter
- IAudioTreble
- IAudioVolumeLevel
- IDeviceSpecificProperty
- IKsFormatSupport
- IKsJackDescription

To obtain the interface ID of the function-specific control interface of a part, call the part's IControlInterface::GetIID method. To obtain the interface ID of a function-specific control interface type, use the **__uuidof** operator. For example,

the interface ID of **IAudioAutoGainControl** is defined as follows:

```
const IID IID_IAudioAutoGainControl  __uuidof(IAudioAutoGainControl)
```

For more information about the **__uuidof** operator, see the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IControlInterface::GetIID

IPart Interface

IPart::EnumPartsIncoming method

1/11/2020 • 2 minutes to read • Edit Online

The **EnumPartsIncoming** method gets a list of all the incoming parts—that is, the parts that reside on data paths that are upstream from this part.

Syntax

```
HRESULT EnumPartsIncoming(
  IPartsList **ppParts
);
```

Parameters

`ppParts`

Pointer to a pointer variable into which the method writes the address of an IPartsList interface that encapsulates the list of parts that are immediately upstream from this part. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **EnumPartsIncoming** call fails, **ppParts* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppParts* is **NULL**.
E_NOTFOUND	This part has no links to upstream parts.
E_OUTOFMEMORY	Out of memory.

Remarks

A client application can traverse a device topology against the direction of audio data flow by iteratively calling this

method at each step in the traversal to get the list of parts that lie immediately upstream from the current part.

If this part has no links to upstream parts, the method returns error code E_NOTFOUND and does not create a parts list (**ppParts* is **NULL**). For example, the method returns this error code if the **IPart** interface represents a connector through which data enters a device topology.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPart Interface

IPartsList Interface

IPart::EnumPartsOutgoing method

1/11/2020 • 2 minutes to read • Edit Online

The **EnumPartsOutgoing** method retrieves a list of all the outgoing parts—that is, the parts that reside on data paths that are downstream from this part.

Syntax

```
HRESULT EnumPartsOutgoing(
  IPartsList **ppParts
);
```

Parameters

`ppParts`

Pointer to a pointer variable into which the method writes the address of an IPartsList interface that encapsulates the list of parts that are immediately downstream from this part. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **EnumPartsOutgoing** call fails, **ppParts* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppParts* is **NULL**.
E_NOTFOUND	This part has no links to downstream parts.
E_OUTOFMEMORY	Out of memory.

Remarks

A client application can traverse a device topology in the direction of audio data flow by iteratively calling this

method at each step in the traversal to get the list of parts that lie immediately downstream from the current part.

If this part has no links to downstream parts, the method returns error code E_NOTFOUND and does not create a parts list (**ppParts* is **NULL**). For example, the method returns this error code if the **IPart** interface represents a connector through which data exits a device topology.

For a code example that uses the **EnumPartsOutgoing** method, see the implementation of the SelectCaptureDevice function in Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPart Interface

IPartsList Interface

IPart::GetControlInterface method

1/11/2020 • 2 minutes to read • Edit Online

The **GetControlInterface** method gets a reference to the specified control interface, if this part supports it.

Syntax

```
HRESULT GetControlInterface(
  UINT              nIndex,
  IControlInterface **ppInterfaceDesc
);
```

Parameters

`nIndex`

The control interface number. If a part supports *n* control interfaces, the control interfaces are numbered from 0 to *n*– 1.

`ppInterfaceDesc`

Pointer to a pointer variable into which the method writes the address of the IControlInterface interface of the specified audio function. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetControlInterface** call fails, **ppFunction* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppFunction* is **NULL**.
E_INVALIDARG	Parameter *nIndex* is out of range.
E_NOTFOUND	The part does not have a control interface.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IControlInterface Interface

IPart Interface

IPart::GetControlInterfaceCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetControlInterfaceCount** method gets the number of control interfaces that this part supports.

Syntax

```
HRESULT GetControlInterfaceCount(
  UINT *pCount
);
```

Parameters

`pCount`

Pointer to a **UINT** variable into which the method writes the number of control interfaces on this part.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pCount* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPart Interface

IPart::GetGlobalId method

1/11/2020 • 2 minutes to read • Edit Online

The **GetGlobalId** method gets the global ID of this part.

Syntax

```
HRESULT GetGlobalId(
  LPWSTR *ppwstrGlobalId
);
```

Parameters

`ppwstrGlobalId`

Pointer to a pointer variable into which the method writes the address of a null-terminated, wide-character string that contains the global ID. The method allocates the storage for the string. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the **GetGlobalId** call fails, **ppwstrGlobalId* is **NULL**. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppwstrGlobalId* is **NULL**.
E_OUTOFMEMORY	Out of memory.

Remarks

A global ID is a string that uniquely identifies a part among all parts in all device topologies in the system. Clients should treat this string as opaque. That is, clients should *not* attempt to parse the contents of the string to obtain information about the part. The reason is that the string format is undefined and might change from one implementation of the DeviceTopology API to the next.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPart Interface

IPart::GetLocalId method

1/11/2020 • 2 minutes to read • Edit Online

The **GetLocalId** method gets the local ID of this part.

Syntax

```
HRESULT GetLocalId(
  UINT *pnId
);
```

Parameters

`pnId`

Pointer to a **UINT** variable into which the method writes the local ID of this part.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pnId* is **NULL**.

Remarks

When you have a pointer to a part object, you can call this method to get the local ID of the part. A local ID is a number that uniquely identifies a part among all parts in a device topology.

The IAudioInputSelector::GetSelection and IAudioOutputSelector::GetSelection methods retrieve the local ID of a connected part. The IAudioInputSelector::SetSelection and IAudioOutputSelector::SetSelection methods select the input or output that is connected to a part that is identified by its local ID. The IDeviceTopology::GetPartById method gets a part that is identified by its local ID.

For code examples that use the **GetLocalId** method, see the following topics:

- Device Topologies
- Using the IKsControl Interface to Access Audio Properties

Requirements

Minimum supported client	Windows Vista [desktop apps only]

Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IAudioInputSelector::GetSelection

IAudioInputSelector::SetSelection

IAudioOutputSelector::GetSelection

IAudioOutputSelector::SetSelection

IDeviceTopology::GetPartById

IPart Interface

IPart::GetName method

1/11/2020 • 2 minutes to read • Edit Online

The **GetName** method gets the friendly name of this part.

Syntax

```
HRESULT GetName(
  LPWSTR *ppwstrName
);
```

Parameters

`ppwstrName`

Pointer to a pointer variable into which the method writes the address of a null-terminated, wide-character string that contains the friendly name of this part. The method allocates the storage for the string. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the **GetName** call fails, **ppwstrName* is **NULL**. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppwstrName* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPart Interface

IPart::GetPartType method

1/11/2020 • 2 minutes to read • Edit Online

The **GetPartType** method gets the part type of this part.

Syntax

```
HRESULT GetPartType(
  PartType *pPartType
);
```

Parameters

`pPartType`

Pointer to a PartType variable into which the method writes the part type. The part type is one of the following **PartType** enumeration values, which indicate whether the part is a connector or subunit:

Connector

Subunit

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pPartType* is **NULL**.

Remarks

For a code example that uses this method, see the implementation of the SelectCaptureDevice function in Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Header	devicetopology.h

See also

IPart Interface

IPart::GetSubType method

1/11/2020 • 2 minutes to read • Edit Online

The **GetSubType** method gets the part subtype of this part.

Syntax

```
HRESULT GetSubType(
  GUID *pSubType
);
```

Parameters

`pSubType`

Pointer to a GUID variable into which the method writes the subtype GUID for this part.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pSubType* is **NULL**.

Remarks

This method typically retrieves one of the KSNODETYPE_*Xxx* GUID values from header file Ksmedia.h, although some custom drivers might provide other GUID values. For more information about KSNODETYPE_*Xxx* GUIDs, see the Windows DDK documentation.

As explained in IPart Interface, a part can be either a connector or a subunit.

For a part that is a connector, this method retrieves the pin-category GUID that the driver has assigned to the connector. The following are examples of pin-category GUIDs:

- KSNODETYPE_ANALOG_CONNECTOR, if the connector is part of the data path to or from an analog device such as a microphone or speakers.
- KSNODETYPE_SPDIF_INTERFACE, if the connector is part of the data path to or from an S/PDIF port.

For more information, see the discussion of the pin-category property, KSPROPERTY_PIN_CATEGORY, in the Windows DDK documentation.

For a part that is a subunit, this method retrieves a subtype GUID that indicates the stream-processing function that the subunit performs. For example, for a volume-control subunit, the method retrieves GUID value KSNODETYPE_VOLUME.

The following table lists some of the subtype GUIDs that can be retrieved by the **GetSubType** method for a

subunit.

SUBTYPE GUID	CONTROL INTERFACE	REQUIRED OR OPTIONAL
KSNODETYPE_3D_EFFECTS	IAudioChannelConfig	Optional
KSNODETYPE_AGC	IAudioAutoGainControl	Required
KSNODETYPE_DAC	IAudioChannelConfig	Optional
KSNODETYPE_DEMUX	IAudioOutputSelector	Required
KSNODETYPE_DEV_SPECIFIC	IDeviceSpecificProperty	Required
KSNODETYPE_LOUDNESS	IAudioLoudness	Required
KSNODETYPE_MUTE	IAudioMute	Required
KSNODETYPE_MUX	IAudioInputSelector	Required
KSNODETYPE_PEAKMETER	IAudioPeakMeter	Required
KSNODETYPE_PROLOGIC_DECODER	IAudioChannelConfig	Optional
KSNODETYPE_TONE	IAudioBass IAudioMidrange IAudioTreble	OptionalOptional Optional
KSNODETYPE_VOLUME	IAudioChannelConfig IAudioVolumeLevel	OptionalRequired

In the preceding table, the middle column lists the control interfaces that are supported by subunits of the subtype specified in the left column. The right column indicates whether the subunit's support for a control interface is required or optional. If support is required, an application can rely on a subunit of the specified subtype to support the control interface. If support is optional, a subunit of the specified subtype can, but does not necessarily, support the control interface.

The control interfaces in the preceding table provide convenient access to the properties of subunits. However, some subunits have properties for which no corresponding control interfaces exist. Applications can access these properties through the IKsControl interface. For more information, see Using the IKsControl Interface to Access Audio Properties.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

IPart Interface

IPart::GetTopologyObject method

1/11/2020 • 2 minutes to read • Edit Online

The **GetTopologyObject** method gets a reference to the IDeviceTopology interface of the device-topology object that contains this part.

Syntax

```
HRESULT GetTopologyObject(
  IDeviceTopology **ppTopology
);
```

Parameters

`ppTopology`

Pointer to a pointer variable into which the method writes the address of the **IDeviceTopology** interface of the device-topology object. The caller obtains a counted reference to the interface from this method. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetTopologyObject** call fails, **ppTopology* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *ppTopology* is **NULL**.

Remarks

For code examples that use this method, see the following topics:

- Device Topologies
- Using the IKsControl Interface to Access Audio Properties

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Target Platform	Windows
Header	devicetopology.h

See also

IDeviceTopology Interface

IPart Interface

IPart::RegisterControlChangeCallback method

1/11/2020 • 2 minutes to read • Edit Online

The **RegisterControlChangeCallback** method registers the IControlChangeNotify interface, which the client implements to receive notifications of status changes in this part.

Syntax

```
HRESULT RegisterControlChangeCallback(
  REFGUID              riid,
  IControlChangeNotify *pNotify
);
```

Parameters

`riid`

The function-specific control interface that is to be monitored for control changes. For more information, see Remarks.

`pNotify`

Pointer to the client's IControlChangeNotify interface. If the method succeeds, it calls the AddRef method on the client's **IControlChangeNotify** interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *riid* is not a valid control-interface identifier.
E_POINTER	Pointer *pNotify* is **NULL**.

Remarks

Set parameter *riid* to one of the following GUID values:

- IID_IAudioAutoGainControl
- IID_IAudioBass
- IID_IAudioChannelConfig
- IID_IAudioInputSelector

- IID_IAudioLoudness
- IID_IAudioMidrange
- IID_IAudioMute
- IID_IAudioOutputSelector
- IID_IAudioPeakMeter
- IID_IAudioTreble
- IID_IAudioVolumeLevel
- IID_IDeviceSpecificProperty
- IID_IKsFormatSupport
- IID_IKsJackDescription

To obtain the interface ID of the function-specific control interface for a part, call the part's IControlInterface::GetIID method. To obtain the interface ID of a function-specific control interface type, use the **__uuidof** operator. For example, the interface ID of IAudioAutoGainControl is defined as follows:

```
const IID IID_IAudioAutoGainControl  __uuidof(IAudioAutoGainControl)
```

For more information about the **__uuidof** operator, see the Windows SDK documentation.

Before the client releases its final reference to the IControlChangeNotify interface, it should call the IPart::UnregisterControlChangeCallback method to unregister the interface. Otherwise, the application leaks the resources held by the **IControlChangeNotify** and IPart objects. Note that **RegisterControlChangeCallback** calls the client's IControlChangeNotify::AddRef method, and **UnregisterControlChangeCallback** calls the IControlChangeNotify::Release method. If the client errs by releasing its reference to the **IControlChangeNotify** interface before calling **UnregisterControlChangeCallback**, the **IPart** object never releases its reference to the **IControlChangeNotify** interface. For example, a poorly designed **IControlChangeNotify** implementation might call **UnregisterControlChangeCallback** from the destructor for the **IControlChangeNotify** object. In this case, the client will not call **UnregisterControlChangeCallback** until the **IPart** object releases its reference to the **IControlChangeNotify** interface, and the **IPart** object will not release its reference to the **IControlChangeNotify** interface until the client calls **UnregisterControlChangeCallback**. For more information about the **AddRef** and **Release** methods, see the discussion of the IUnknown interface in the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IControlChangeNotify Interface

IControlInterface::GetIID

IPart Interface

IPart::UnregisterControlChangeCallback

IPart::UnregisterControlChangeCallback method

1/11/2020 • 2 minutes to read • Edit Online

The **UnregisterControlChangeCallback** method removes the registration of an IControlChangeNotify interface that the client previously registered by a call to the IPart::RegisterControlChangeCallback method.

Syntax

```
HRESULT UnregisterControlChangeCallback(
  IControlChangeNotify *pNotify
);
```

Parameters

`pNotify`

Pointer to the **IControlChangeNotify** interface whose registration is to be deleted. The client passed this same interface pointer to the part object in a previous call to the **IPart::RegisterControlChangeCallback** method. If the **UnregisterControlChangeCallback** method succeeds, it calls the **Release** method on the client's **IControlChangeNotify** interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Pointer *pNotify* is **NULL**.
E_N OTF OU ND	Interface instance **pNotify* is not currently registered.

Remarks

Before the client releases its final reference to the **IControlChangeNotify** interface, it should call **UnregisterControlChangeCallback** to unregister the interface. Otherwise, the application leaks the resources held by the **IControlChangeNotify** and **IPart** objects. Note that the **IPart::RegisterControlChangeCallback** method calls the client's **IControlChangeNotify::AddRef** method, and **UnregisterControlChangeCallback** calls the **IControlChangeNotify::Release** method. If the client errs by releasing its reference to the **IControlChangeNotify** interface before calling **UnregisterControlChangeCallback**, the **IPart** object never releases its reference to the **IControlChangeNotify** interface. For example, a poorly designed **IControlChangeNotify** implementation might call **UnregisterControlChangeCallback** from the destructor for

the **IControlChangeNotify** object. In this case, the client will not call **UnregisterControlChangeCallback** until the **IPart** object releases its reference to the **IControlChangeNotify** interface, and the **IPart** object will not release its reference to the **IControlChangeNotify** interface until the client calls **UnregisterControlChangeCallback**. For more information about the **AddRef** and **Release** methods, see the discussion of the **IUnknown** interface in the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IControlChangeNotify Interface

IPart Interface

IPart::RegisterControlChangeCallback

IPartsList interface

1/23/2020 • 2 minutes to read • Edit Online

The **IPartsList** interface represents a list of parts, each of which is an object with an IPart interface that represents a connector or subunit. A client obtains a reference to an **IPartsList** interface by calling the IPart::EnumPartsIncoming, IPart::EnumPartsOutgoing, or IDeviceTopology::GetSignalPath method.

For a code example that uses the **IPartsList** interface, see the implementation of the SelectCaptureDevice function in Device Topologies.

Inheritance

The **IPartsList** interface inherits from the IUnknown interface. **IPartsList** also has these types of members:

- Methods

Methods

The **IPartsList** interface has these methods.

METHOD	DESCRIPTION
IPartsList::GetCount	The GetCount method gets the number of parts in the parts list.
IPartsList::GetPart	The GetPart method gets a part from the parts list.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IDeviceTopology::GetSignalPath

IPart Interface

IPart::EnumPartsIncoming

IPart::EnumPartsOutgoing

IPartsList::GetCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetCount** method gets the number of parts in the parts list.

Syntax

```
HRESULT GetCount(
  UINT *pCount
);
```

Parameters

`pCount`

Pointer to a **UINT** variable into which the method writes the parts count (the number of parts in the parts list).

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pCount* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPartsList Interface

IPartsList::GetPart method

1/11/2020 • 2 minutes to read • Edit Online

The **GetPart** method gets a part from the parts list.

Syntax

```
HRESULT GetPart(
  UINT  nIndex,
  IPart **ppPart
);
```

Parameters

`nIndex`

The part number of the part to retrieve. If the parts list contains *n* parts, the parts are numbered 0 to *n*– 1. Call the IPartsList::GetCount method to get the number of parts in the list.

`ppPart`

Pointer to a pointer variable into which the method writes the address of the IPart interface of the part object. Through this method, the caller obtains a counted reference to the **IPart** interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetPart** call fails, **ppPart* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *nIndex* is out of range.
E_P OIN TER	Pointer *ppPart* is **NULL**.

Remarks

For a code example that calls the **GetPart** method, see the implementation of the SelectCaptureDevice function in Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPart Interface

IPartsList Interface

IPartsList::GetCount

IPerChannelDbLevel interface

1/23/2020 • 2 minutes to read • Edit Online

The **IPerChannelDbLevel** interface represents a generic subunit control interface that provides per-channel control over the volume level, in decibels, of an audio stream or of a frequency band in an audio stream. A positive volume level represents gain, and a negative value represents attenuation.

Clients do not call the methods in this interface directly. Instead, this interface serves as the base interface for the following interfaces, which clients do call directly:

- IAudioBass Interface
- IAudioMidrange Interface
- IAudioTreble Interface
- IAudioVolumeLevel Interface

Inheritance

The **IPerChannelDbLevel** interface inherits from the IUnknown interface. **IPerChannelDbLevel** also has these types of members:

- Methods

Methods

The **IPerChannelDbLevel** interface has these methods.

METHOD	DESCRIPTION
IPerChannelDbLevel::GetChannelCount	The GetChannelCount method gets the number of channels in the audio stream.
IPerChannelDbLevel::GetLevel	The GetLevel method gets the volume level, in decibels, of the specified channel.
IPerChannelDbLevel::GetLevelRange	The GetLevelRange method gets the range, in decibels, of the volume level of the specified channel.
IPerChannelDbLevel::SetLevel	The SetLevel method sets the volume level, in decibels, of the specified channel.
IPerChannelDbLevel::SetLevelAllChannels	The SetLevelAllChannels method sets the volume levels, in decibels, of all the channels in the audio stream.
IPerChannelDbLevel::SetLevelUniform	The SetLevelUniform method sets all channels in the audio stream to the same uniform volume level, in decibels.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IAudioBass Interface

IAudioMidrange Interface

IAudioTreble Interface

IAudioVolumeLevel Interface

IPerChannelDbLevel::GetChannelCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelCount** method gets the number of channels in the audio stream.

Syntax

```
HRESULT GetChannelCount(
  UINT *pcChannels
);
```

Parameters

`pcChannels`

Pointer to a **UINT** variable into which the method writes the channel count (the number of channels in the audio stream).

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Pointer *pcChannels* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPerChannelDbLevel Interface

IPerChannelDbLevel::GetLevel method

1/11/2020 • 2 minutes to read • Edit Online

The **GetLevel** method gets the volume level, in decibels, of the specified channel.

Syntax

```
HRESULT GetLevel(
  UINT  nChannel,
  float *pfLevelDB
);
```

Parameters

`nChannel`

The channel number. If the audio stream has *N* channels, the channels are numbered from 0 to *N*– 1. To get the number of channels in the stream, call the IPerChannelDbLevel::GetChannelCount method.

`pfLevelDB`

Pointer to a **float** variable into which the method writes the volume level, in decibels, of the specified channel.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *nChannel* is out of range.
E_P OIN TER	Pointer *pfLevelDB* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Header	devicetopology.h

See also

IPerChannelDbLevel Interface

IPerChannelDbLevel::GetChannelCount

IPerChannelDbLevel::GetLevelRange method

1/11/2020 • 2 minutes to read • Edit Online

The **GetLevelRange** method gets the range, in decibels, of the volume level of the specified channel.

Syntax

```
HRESULT GetLevelRange(
  UINT  nChannel,
  float *pfMinLevelDB,
  float *pfMaxLevelDB,
  float *pfStepping
);
```

Parameters

`nChannel`

The number of the selected channel. If the audio stream has *n* channels, the channels are numbered from 0 to *n*– 1. To get the number of channels in the stream, call the IPerChannelDbLevel::GetChannelCount method.

`pfMinLevelDB`

Pointer to a **float** variable into which the method writes the minimum volume level in decibels.

`pfMaxLevelDB`

Pointer to a **float** variable into which the method writes the maximum volume level in decibels.

`pfStepping`

Pointer to a **float** variable into which the method writes the stepping value between consecutive volume levels in the range **pfMinLevelDB* to **pfMaxLevelDB*. If the difference between the maximum and minimum volume levels is *d* decibels, and the range is divided into *n* steps (uniformly sized intervals), then the volume can have *n* + 1 discrete levels and the size of the step between consecutive levels is *d* / *n* decibels.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *nChannel* is out of range.

E_POINTER	Pointer *pfminLevelDB*, *pfmaxLevelDB*, or *pfmaxLevelDB* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPerChannelDbLevel Interface

IPerChannelDbLevel::GetChannelCount

IPerChannelDbLevel::SetLevel method

1/11/2020 • 2 minutes to read • Edit Online

The **SetLevel** method sets the volume level, in decibels, of the specified channel.

Syntax

```
HRESULT SetLevel(
  UINT    nChannel,
  float   fLevelDB,
  LPCGUID pguidEventContext
);
```

Parameters

`nChannel`

The number of the selected channel. If the audio stream has *N* channels, the channels are numbered from 0 to *N*–1. To get the number of channels in the stream, call the IPerChannelDbLevel::GetChannelCount method.

`fLevelDB`

The new volume level in decibels. A positive value represents gain, and a negative value represents attenuation.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetLevel** call changes the state of the level control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDA RG	Parameter *nChannel* is out of range.

E_OUTOFMEMORY	Out of memory.

Remarks

If the caller specifies a value for *fLevelDB* that is an exact stepping value, the **SetLevel** method completes successfully. A subsequent call to the IPerChannelDbLevel::GetLevel method will return either the value that was set, or one of the following values:

- If the set value was below the minimum, the **GetLevel** method returns the minimum value.
- If the set value was above the maximum, the **GetLevel** method returns the maximum value.
- If the set value was between two stepping values, the **GetLevel** method returns a value that could be the next stepping value above or the stepping value below the set value; the relative distances from the set value to the neighboring stepping values is unimportant. The value that the **GetLevel** method returns is whichever value has more of an impact on the signal path.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPerChannelDbLevel Interface

IPerChannelDbLevel::GetChannelCount

IPerChannelDbLevel::GetLevel

IPerChannelDbLevel::GetLevelRange

IPerChannelDbLevel::SetLevelAllChannels method

1/11/2020 • 2 minutes to read • Edit Online

The **SetLevelAllChannels** method sets the volume levels, in decibels, of all the channels in the audio stream.

Syntax

```
HRESULT SetLevelAllChannels(
  float [] aLevelsDB,
  ULONG    cChannels,
  LPCGUID  pguidEventContext
);
```

Parameters

`aLevelsDB`

Pointer to an array of volume levels. This parameter points to a caller-allocated **float** array into which the method writes the new volume levels, in decibels, for all the channels. The method writes the level for a particular channel into the array element whose index matches the channel number. If the audio stream contains *n* channels, the channels are numbered 0 to *n*– 1. To get the number of channels in the stream, call the IPerChannelDbLevel::GetChannelCount method.

`cChannels`

The number of elements in the *aLevelsDB* array. If this parameter does not match the number of channels in the audio stream, the method fails without modifying the *aLevelsDB* array.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetLevelAllChannels** call changes the state of the level control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN **VAL** **IDA** **RG**	Parameter *cChannels* does not equal the number of channels.

E_POINTER	Pointer *aLevelsDB* is **NULL**.
E_OUTOFMEMORY	Out of memory.

Remarks

If the specified level value for any channel is beyond the range that the IPerChannelDbLevel::GetLevelRange method reports for that channel, the **SetLevelAllChannels** call clamps the value to the supported range and completes successfully. A subsequent call to the IPerChannelDbLevel::GetLevel method retrieves the actual value used for that channel.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPerChannelDbLevel Interface

IPerChannelDbLevel::GetChannelCount

IPerChannelDbLevel::GetLevel

IPerChannelDbLevel::GetLevelRange

IPerChannelDbLevel::SetLevelUniform method

1/11/2020 • 2 minutes to read • Edit Online

The **SetLevelUniform** method sets all channels in the audio stream to the same uniform volume level, in decibels.

Syntax

```
HRESULT SetLevelUniform(
  float   fLevelDB,
  LPCGUID pguidEventContext
);
```

Parameters

`fLevelDB`

The new uniform level in decibels.

`pguidEventContext`

Context value for the IControlChangeNotify::OnNotify method. This parameter points to an event-context GUID. If the **SetLevelUniform** call changes the state of the level control, all clients that have registered IControlChangeNotify interfaces with that control receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_OUTOFMEMORY	Out of memory.

Remarks

If the specified uniform level is beyond the range that the IPerChannelDbLevel::GetLevelRange method reports for a particular channel, the **SetLevelUniform** call clamps the value for that channel to the supported range and completes successfully. A subsequent call to the IPerChannelDbLevel::GetLevel method retrieves the actual value used for that channel.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

IPerChannelDbLevel Interface

IPerChannelDbLevel::GetLevel

IPerChannelDbLevel::GetLevelRange

ISubunit interface

1/24/2020 • 2 minutes to read • Edit Online

The **ISubunit** interface represents a hardware subunit (for example, a volume control) that lies in the data path between a client and an audio endpoint device. The client obtains a reference to an **ISubunit** interface by calling the IDeviceTopology::GetSubunit method, or by calling the **IPart::QueryInterface** method with parameter *iid* set to **REFIID** IID_ISubunit.

Inheritance

The ISubunit interface inherits from the IUnknown interface.

Methods

The **ISubunit** interface has these methods.

METHOD	DESCRIPTION

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	devicetopology.h

See also

Core Audio Interfaces

DeviceTopology API

IDeviceTopology::GetSubunit

IPart Interface

KSJACK_DESCRIPTION structure

1/23/2020 • 3 minutes to read • Edit Online

The **KSJACK_DESCRIPTION** structure describes an audio jack.

Syntax

```
typedef struct __MIDL___MIDL_itf_devicetopology_0000_0000_0009 {
  DWORD              ChannelMapping;
  COLORREF           Color;
  EPcxConnectionType ConnectionType;
  EPcxGeoLocation    GeoLocation;
  EPcxGenLocation    GenLocation;
  EPxcPortConnection PortConnection;
  BOOL               IsConnected;
} KSJACK_DESCRIPTION, *PKSJACK_DESCRIPTION;
```

Members

`ChannelMapping`

Specifies the mapping of the two audio channels in a stereo jack to speaker positions.

In Windows Vista, the value of this member is one of the **EChannelMapping** enumeration values shown in the following table.

VALUE	FIRST CHANNEL	SECOND CHANNEL
ePcxChanMap_FL_FR	Front-left speaker	Front-right speaker
ePcxChanMap_FC_LFE	Front-center speaker	Low-frequency-effects speaker (subwoofer)
ePcxChanMap_BL_BR	Back-left speaker	Back-right speakers
ePcxChanMap_FLC_FRC	Front-left-center speaker	Front-right-center speaker
ePcxChanMap_SL_SR	Side-left speaker	Side-right speaker
ePcxChanMap_Unknown	Unknown	Unknown

For a physical connector with one, three, or more channels, the value of this member is ePcxChanMap_Unknown.

In Windows 7, the **EChannelMapping**enumeration has been deprecated. The datatype of this member is a **DWORD**. This member stores either 0 or the bitwise-OR combination of one or more of the following values that are defined in Ksmedia.h.

```
#define SPEAKER_FRONT_LEFT              0x1
#define SPEAKER_FRONT_RIGHT             0x2
#define SPEAKER_FRONT_CENTER            0x4
#define SPEAKER_LOW_FREQUENCY           0x8
#define SPEAKER_BACK_LEFT               0x10
#define SPEAKER_BACK_RIGHT              0x20
#define SPEAKER_FRONT_LEFT_OF_CENTER    0x40
#define SPEAKER_FRONT_RIGHT_OF_CENTER   0x80
#define SPEAKER_BACK_CENTER             0x100
#define SPEAKER_SIDE_LEFT               0x200
#define SPEAKER_SIDE_RIGHT              0x400
#define SPEAKER_TOP_CENTER              0x800
#define SPEAKER_TOP_FRONT_LEFT          0x1000
#define SPEAKER_TOP_FRONT_CENTER        0x2000
#define SPEAKER_TOP_FRONT_RIGHT         0x4000
#define SPEAKER_TOP_BACK_LEFT           0x8000
#define SPEAKER_TOP_BACK_CENTER         0x10000
#define SPEAKER_TOP_BACK_RIGHT          0x20000
```

`Color`

The jack color. The color is expressed as a 32-bit RGB value that is formed by concatenating the 8-bit blue, green, and red color components. The blue component occupies the 8 least-significant bits (bits 0-7), the green component occupies bits 8-15, and the red component occupies bits 16-23. The 8 most-significant bits are zeros. If the jack color is unknown or the physical connector has no identifiable color, the value of this member is 0x00000000, which is black.

`ConnectionType`

The connection type. The value of this member is one of the **EPcxConnectionType** enumeration values shown in the following table.

VALUE	CONNECTOR TYPE
eConnTypeUnknown	Unknown
eConnTypeEighth (Windows Vista) eConnType3Point5mm (Windows 7)	1/8-inch jack
eConnTypeQuarter	1/4-inch jack
eConnTypeAtapiInternal	ATAPI internal connector
eConnTypeRCA	RCA jack
eConnTypeOptical	Optical connector
eConnTypeOtherDigital	Generic digital connector
eConnTypeOtherAnalog	Generic analog connector
eConnTypeMultichannelAnalogDIN	Multichannel analog DIN connector
eConnTypeXlrProfessional	XLR connector

eConnTypeRJ11Modem	RJ11 modem connector
eConnTypeCombination	Combination of connector types

`GeoLocation`

The geometric location of the jack. The value of this member is one of the **EPcxGeoLocation** enumeration values shown in the following table.

VALUE	GEOMETRIC LOCATION
eGeoLocRear	Rear-mounted panel
eGeoLocFront	Front-mounted panel
eGeoLocLeft	Left-mounted panel
eGeoLocRight	Right-mounted panel
eGeoLocTop	Top-mounted panel
eGeoLocBottom	Bottom-mounted panel
eGeoLocRearOPanel(Windows Vista) eGeoLocRearPanel(Windows 7)	Rear slide-open or pull-open panel
eGeoLocRiser	Riser card
eGeoLocInsideMobileLid	Inside lid of mobile computer
eGeoLocDrivebay	Drive bay
eGeoLocHDMI	HDMI connector
eGeoLocOutsideMobileLid	Outside lid of mobile computer
eGeoLocATAPI	ATAPI connector

`GenLocation`

The general location of the jack. The value of this member is one of the **EPcxGenLocation** enumeration values shown in the following table.

VALUE	GENERAL LOCATION
eGenLocPrimaryBox	On primary chassis
eGenLocInternal	Inside primary chassis
eGenLocSeperate(Windows Vista) eGenLocSeparate(Windows 7)	On separate chassis

eGenLocOther	Other location

`PortConnection`

The type of port represented by the jack. The value of this member is one of the **EPxcPortConnection** enumeration values shown in the following table.

VALUE	PORT CONNECTION TYPE
ePortConnJack	Jack
ePortConnIntegratedDevice	Slot for an integrated device
ePortConnBothIntegratedAndJack	Both a jack and a slot for an integrated device
ePortConnUnknown	Unknown

`IsConnected`

If the audio adapter supports jack-presence detection on the jack, the value of **IsConnected** indicates whether an endpoint device is plugged into the jack. If **IsConnected** is **TRUE**, a device is plugged in. If it is **FALSE**, the jack is empty. For devices that do not support jack-presence detection, this member is always **TRUE**. For more information about jack-presence detection, see Audio Endpoint Devices.

Remarks

This structure is used by the IKsJackDescription::GetJackDescription method in the DeviceTopology API. It describes an audio jack that is part of a connection between an endpoint device and a hardware device in an audio adapter. When a user needs to plug an endpoint device into a jack or unplug it from a jack, an audio application can use the descriptive information in the structure to help the user to find the jack.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	devicetopology.h

See also

Core Audio Structures

IKsJackDescription::GetJackDescription

KSJACK_DESCRIPTION2 structure

1/23/2020 • 2 minutes to read • Edit Online

The **KSJACK_DESCRIPTION2** structure describes an audio jack.

To get the description of an audio jack of a connector, call IKsJackDescription2::GetJackDescription2.

Syntax

```
typedef struct _tagKSJACK_DESCRIPTION2 {
  DWORD DeviceStateInfo;
  DWORD JackCapabilities;
} KSJACK_DESCRIPTION2, *PKSJACK_DESCRIPTION2;
```

Members

`DeviceStateInfo`

Reserved for future use.

`JackCapabilities`

Stores the audio jack's capabilities: jack presence detection capability or dynamic format changing capability. The constants that can be stored in this member of the structure are defined in Ksmedia.h as follows:

- JACKDESC2_PRESENCE_DETECT_CAPABILITY (0x00000001)
- JACKDESC2_DYNAMIC_FORMAT_CHANGE_CAPABILITY (0x00000002)

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	devicetopology.h

See also

Core Audio Structures

IKsJackDescription2

KSJACK_SINK_CONNECTIONTYPE enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **KSJACK_SINK_CONNECTIONTYPE** enumeration defines constants that specify the type of connection. These values are used in the KSJACK_SINK_INFORMATION structure that stores information about an audio jack sink.

Syntax

```
typedef enum __MIDL___MIDL_itf_devicetopology_0000_0000_0010 {
  KSJACK_SINK_CONNECTIONTYPE_HDMI,
  KSJACK_SINK_CONNECTIONTYPE_DISPLAYPORT
} KSJACK_SINK_CONNECTIONTYPE;
```

Constants

KSJACK_SINK_CONNECTIONTYPE_HDMI	High-Definition Multimedia Interface (HDMI) connection.
KSJACK_SINK_CONNECTIONTYPE_DISPLAYPORT	Display port.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	devicetopology.h

See also

Core Audio Enumerations

IKsJackSinkInformation::GetJackSinkInformation

KSJACK_SINK_INFORMATION

KSJACK_SINK_INFORMATION structure

1/23/2020 • 2 minutes to read • Edit Online

The **KSJACK_SINK_INFORMATION** structure stores information about an audio jack sink.

Syntax

```
typedef struct _tagKSJACK_SINK_INFORMATION {
  KSJACK_SINK_CONNECTIONTYPE ConnType;
  WORD                       ManufacturerId;
  WORD                       ProductId;
  WORD                       AudioLatency;
  BOOL                       HDCPCapable;
  BOOL                       AICapable;
  UCHAR                      SinkDescriptionLength;
  WCHAR                      SinkDescription[32];
  LUID                       PortId;
} KSJACK_SINK_INFORMATION;
```

Members

`ConnType`

Specifies the type of connection. The connection type values are defined in the KSJACK_SINK_CONNECTIONTYPE enumeration.

`ManufacturerId`

Specifies the sink manufacturer identifier.

`ProductId`

Specifies the sink product identifier.

`AudioLatency`

Specifies the latency of the audio sink.

`HDCPCapable`

Specifies whether the sink supports High-bandwidth Digital Content Protection (HDCP).

`AICapable`

Specifies whether the sink supports ACP Packet, ISRC1, or ISRC2.

`SinkDescriptionLength`

Specifies the length of the string in the **SinkDescription** member.

`SinkDescription`

String containing the monitor sink name. The maximum length is defined by the constant **MAX_SINK_DESCRIPTION_NAME_LENGTH** (32 wide characters).

`PortId`

Specifies the video port identifier in a LUID structure.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	devicetopology.h

See also

Core Audio Structures

IKsJackSinkInformation::GetJackSinkInformation

LUID structure

1/23/2020 • 2 minutes to read • Edit Online

The **LUID** structure stores the video port identifier. This structure is stored in the **PortId** member of the KSJACK_SINK_INFORMATION structure.

Syntax

```
typedef struct _LUID {
  DWORD LowPart;
  LONG  HighPart;
} LUID, *PLUID;
```

Members

`LowPart`

LowPart of the video port identifier.

`HighPart`

HighPart of the video port identifier.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	devicetopology.h

See also

Core Audio Structures

IKsJackSinkInformation::GetJackSinkInformation

PartType enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **PartType** enumeration defines constants that indicate whether a part in a device topology is a connector or subunit.

Syntax

```
typedef enum __MIDL___MIDL_itf_devicetopology_0000_0000_0012 {
  Connector,
  Subunit
} PartType;
```

Constants

Connector	The part is a connector. A connector can represent an audio jack, an internal connection to an integrated endpoint device, or a software connection implemented through DMA transfers. For more information about connector types, see ConnectorType Enumeration.
Subunit	The part is a subunit. A subunit is an audio-processing node in a device topology. A subunit frequently has one or more hardware control parameters that can be set under program control. For example, an audio application can change the volume setting of a volume-control subunit.

Remarks

The IPart::GetPartType method uses the constants defined in the **PartType** enumeration to indicate whether an IPart object represents a connector or a subunit. If an **IPart** object represents a connector, a client can query that that object for its IConnector interface. If an **IPart** object represents a subunit, a client can query that that object for its ISubunit interface.

For more information about connectors and subunits, see Device Topologies.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	devicetopology.h

See also

Core Audio Constants

Core Audio Enumerations

IConnector Interface

IPart Interface

ISubunit Interface

endpointvolume.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs endpointvolume.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IAudioEndpointVolume	The IAudioEndpointVolume interface represents the volume controls on the audio stream to or from an audio endpoint device.
IAudioEndpointVolumeCallback	The IAudioEndpointVolumeCallback interface provides notifications of changes in the volume level and muting state of an audio endpoint device.
IAudioEndpointVolumeEx	The IAudioEndpointVolumeEx interface provides volume controls on the audio stream to or from a device endpoint.
IAudioMeterInformation	The IAudioMeterInformation interface represents a peak meter on an audio stream to or from an audio endpoint device.

Structures

TITLE	DESCRIPTION
AUDIO_VOLUME_NOTIFICATION_DATA	The AUDIO_VOLUME_NOTIFICATION_DATA structure describes a change in the volume level or muting state of an audio endpoint device.

AUDIO_VOLUME_NOTIFICATION_DATA structure

1/23/2020 • 2 minutes to read • Edit Online

The **AUDIO_VOLUME_NOTIFICATION_DATA** structure describes a change in the volume level or muting state of an audio endpoint device.

Syntax

```
typedef struct AUDIO_VOLUME_NOTIFICATION_DATA {
  GUID  guidEventContext;
  BOOL  bMuted;
  float fMasterVolume;
  UINT  nChannels;
  float afChannelVolumes[1];
} AUDIO_VOLUME_NOTIFICATION_DATA, *PAUDIO_VOLUME_NOTIFICATION_DATA;
```

Members

`guidEventContext`

Context value for the IAudioEndpointVolumeCallback::OnNotify method. This member is the value of the event-context GUID that was provided as an input parameter to the IAudioEndpointVolume method call that changed the endpoint volume level or muting state. For more information, see Remarks.

`bMuted`

Specifies whether the audio stream is currently muted. If **bMuted** is **TRUE**, the stream is muted. If **FALSE**, the stream is not muted.

`fMasterVolume`

Specifies the current master volume level of the audio stream. The volume level is normalized to the range from 0.0 to 1.0, where 0.0 is the minimum volume level and 1.0 is the maximum level. Within this range, the relationship of the normalized volume level to the attenuation of signal amplitude is described by a nonlinear, audio-tapered curve. For more information about audio tapers, see Audio-Tapered Volume Controls.

`nChannels`

Specifies the number of channels in the audio stream, which is also the number of elements in the **afChannelVolumes** array. If the audio stream contains *n* channels, the channels are numbered from 0 to *n*-1. The volume level for a particular channel is contained in the array element whose index matches the channel number.

`afChannelVolumes`

The first element in an array of channel volumes. This element contains the current volume level of channel 0 in the audio stream. If the audio stream contains more than one channel, the volume levels for the additional channels immediately follow the **AUDIO_VOLUME_NOTIFICATION_DATA** structure. The volume level for each channel is normalized to the range from 0.0 to 1.0, where 0.0 is the minimum volume level and 1.0 is the maximum level. Within this range, the relationship of the normalized volume level to the attenuation of signal amplitude is described by a nonlinear, audio-tapered curve.

Remarks

This structure is used by the **IAudioEndpointVolumeCallback::OnNotify** method.

A client can register to be notified when the volume level or muting state of an endpoint device changes. The following methods can cause such a change:

- IAudioEndpointVolume::SetChannelVolumeLevel
- IAudioEndpointVolume::SetChannelVolumeLevelScalar
- IAudioEndpointVolume::SetMasterVolumeLevel
- IAudioEndpointVolume::SetMasterVolumeLevelScalar
- IAudioEndpointVolume::SetMute
- IAudioEndpointVolume::VolumeStepDown
- IAudioEndpointVolume::VolumeStepUp

When a call to one of these methods causes a volume-change event (that is, a change in the volume level or muting state), the method sends notifications to all clients that have registered to receive them. The method notifies a client by calling the client's **IAudioEndpointVolumeCallback::OnNotify** method. Through the **OnNotify** call, the client receives a pointer to an **AUDIO_VOLUME_NOTIFICATION_DATA** structure that describes the change.

Each of the methods in the preceding list accepts an input parameter named *pguidEventContext*, which is a pointer to an event-context GUID. Before sending notifications to clients, the method copies the event-context GUID pointed to by *pguidEventContext* into the **guidEventContext** member of the **AUDIO_VOLUME_NOTIFICATION_DATA** structure that it supplies to clients through their **OnNotify** methods. If *pguidEventContext* is **NULL**, the value of the **guidEventContext** member is set to GUID_NULL.

In its implementation of the **OnNotify** method, a client can inspect the event-context GUID from that call to discover whether it or another client is the source of the volume-change event.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Header	endpointvolume.h

See also

Core Audio Structures

IAudioEndpointVolume Interface

IAudioEndpointVolume::SetChannelVolumeLevel

IAudioEndpointVolume::SetChannelVolumeLevelScalar

IAudioEndpointVolume::SetMasterVolumeLevel

IAudioEndpointVolume::SetMasterVolumeLevelScalar

IAudioEndpointVolume::SetMute

IAudioEndpointVolume::VolumeStepDown

IAudioEndpointVolume::VolumeStepUp

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolumeCallback::OnNotify

IAudioEndpointVolume interface

1/23/2020 • 6 minutes to read • Edit Online

The **IAudioEndpointVolume** interface represents the volume controls on the audio stream to or from an audio endpoint device. A client obtains a reference to the **IAudioEndpointVolume** interface of an endpoint device by calling the IMMDevice::Activate method with parameter *iid* set to REFIID IID_IAudioEndpointVolume.

Audio applications that use the MMDevice API and WASAPI typically use the ISimpleAudioVolume interface to manage stream volume levels on a per-session basis. In rare cases, a specialized audio application might require the use of the **IAudioEndpointVolume** interface to control the master volume level of an audio endpoint device. A client of **IAudioEndpointVolume** must take care to avoid the potentially disruptive effects on other audio applications of altering the master volume levels of audio endpoint devices. Typically, the user has exclusive control over the master volume levels through the Windows volume-control program, Sndvol.exe.

If the adapter device that streams audio data to or from the endpoint device has hardware volume and mute controls, the **IAudioEndpointVolume** interface uses those controls to manage the volume and mute settings of the audio stream. If the audio device lacks a hardware volume control for the stream, the audio engine automatically implements volume and mute controls in software.

For applications that manage shared-mode streams to and from endpoint devices, the behavior of the **IAudioEndpointVolume** is different for rendering streams and capture streams.

For a shared-mode rendering stream, the endpoint volume control that the client accesses through the **IAudioEndpointVolume** interface operates independently of the per-session volume controls that the **ISimpleAudioVolume** and IChannelAudioVolume interfaces implement. Thus, the volume level of the rendering stream results from the combined effects of the endpoint volume control and per-session controls.

For a shared-mode capture stream, the per-session volume controls that the **ISimpleAudioVolume** and **IChannelAudioVolume** interfaces implement are tied directly to the endpoint volume control implemented by the **IAudioEndpointVolume** interface. Changing the per-session volume control through the methods in the **ISimpleAudioVolume** and **IChannelAudioVolume** interfaces changes the setting of the **IAudioEndpointVolume** interface's volume control, and the reverse is also true. The volume levels represented by each of the interfaces correspond to each other as follows:

- For each channel in a stream, **IAudioEndpointVolume** provides audio-tapered volume levels expressed in decibels (dB), that are mapped to normalized values in the range from 0.0 (minimum volume) to 1.0 (maximum volume). The possible range is dependent on the audio driver. See IAudioEndpointVolume::GetVolumeRange for details.
- The session volume represented by ISimpleAudioVolume::GetMasterVolume is the scalar value ranging from 0.0 to 1.0 that corresponds to the highest volume setting across the various channels. So, for example, if the left channel is set to 0.8, and the right channel is set to 0.4, then **ISimpleAudioVolume::GetMasterVolume** will return 0.8.
- When the per-channel volume level is controlled through the methods in the IChannelAudioVolume interface, the scalar indicating volume is always relative to the session volume. This means that the channel or channels with the highest volume has a volume of 1.0. Given the example of two channels, set to volumes of 0.8 and 0.4 by IAudioEndpointVolume::SetChannelVolumeLevelScalar, IChannelAudioVolume::GetChannelVolume will indicate volumes of 1.0 and 0.5.

> **Note** Clients of the **EndpointVolume** API should not rely on the preceding behavior because it might change in future releases.

If a device has hardware volume and mute controls, changes made to the device's volume and mute settings through the **IAudioEndpointVolume** interface affect the volume level in both shared mode and exclusive mode. If a device lacks hardware volume and mute controls, changes made to the software volume and mute controls through the **IAudioEndpointVolume** interface affect the volume level in shared mode, but not in exclusive mode. In exclusive mode, the client and the device exchange audio data directly, bypassing the software controls. However, the software controls are persistent, and volume changes made while the device operates in exclusive mode take effect when the device switches to shared-mode operation.

To determine whether a device has hardware volume and mute controls, call the IAudioEndpointVolume::QueryHardwareSupport method.

The methods of the **IAudioEndpointVolume** interface enable the client to express volume levels either in decibels or as normalized, audio-tapered values. In the latter case, a volume level is expressed as a floating-point value in the normalized range from 0.0 (minimum volume) to 1.0 (maximum volume). Within this range, the relationship of the normalized volume level to the attenuation of signal amplitude is described by a nonlinear, audio-tapered curve. For more information about audio-tapered curves, see Audio-Tapered Volume Controls.

In addition, to conveniently support volume sliders in user interfaces, the **IAudioEndpointVolume** interface enables clients to set and get volume levels that are expressed as discrete values or "steps". The steps are uniformly distributed over a nonlinear, audio-tapered curve. If the range contains *n* steps, the steps are numbered from 0 to *n* – 1, where step 0 represents the minimum volume level and step *n*– 1 represents the maximum.

For a code example that uses the **IAudioEndpointVolume** interface, see Endpoint Volume Controls.

Inheritance

The **IAudioEndpointVolume** interface inherits from the IUnknown interface. **IAudioEndpointVolume** also has these types of members:

- Methods

Methods

The **IAudioEndpointVolume** interface has these methods.

METHOD	DESCRIPTION
IAudioEndpointVolume::GetChannelCount	The GetChannelCount method gets a count of the channels in the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::GetChannelVolumeLevel	The GetChannelVolumeLevel method gets the volume level, in decibels, of the specified channel in the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::GetChannelVolumeLevelScalar	The GetChannelVolumeLevelScalar method gets the normalized, audio-tapered volume level of the specified channel of the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::GetMasterVolumeLevel	The GetMasterVolumeLevel method gets the master volume level, in decibels, of the audio stream that enters or leaves the audio endpoint device.

METHOD	DESCRIPTION
IAudioEndpointVolume::GetMasterVolumeLevelScalar	The GetMasterVolumeLevelScalar method gets the master volume level of the audio stream that enters or leaves the audio endpoint device. The volume level is expressed as a normalized, audio-tapered value in the range from 0.0 to 1.0.
IAudioEndpointVolume::GetMute	The GetMute method gets the muting state of the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::GetVolumeRange	The GetVolumeRange method gets the volume range, in decibels, of the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::GetVolumeStepInfo	The GetVolumeStepInfo method gets information about the current step in the volume range.
IAudioEndpointVolume::QueryHardwareSupport	The QueryHardwareSupport method queries the audio endpoint device for its hardware-supported functions.
IAudioEndpointVolume::RegisterControlChangeNotify	The RegisterControlChangeNotify method registers a client's notification callback interface.
IAudioEndpointVolume::SetChannelVolumeLevel	The SetChannelVolumeLevel method sets the volume level, in decibels, of the specified channel of the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::SetChannelVolumeLevelScalar	The SetChannelVolumeLevelScalar method sets the normalized, audio-tapered volume level of the specified channel in the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::SetMasterVolumeLevel	The SetMasterVolumeLevel method sets the master volume level, in decibels, of the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::SetMasterVolumeLevelScalar	The SetMasterVolumeLevelScalar method sets the master volume level of the audio stream that enters or leaves the audio endpoint device. The volume level is expressed as a normalized, audio-tapered value in the range from 0.0 to 1.0.
IAudioEndpointVolume::SetMute	The SetMute method sets the muting state of the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::UnregisterControlChangeNotify	The UnregisterControlChangeNotify method deletes the registration of a client's notification callback interface that the client registered in a previous call to the IAudioEndpointVolume::RegisterControlChangeNotify method.
IAudioEndpointVolume::VolumeStepDown	The VolumeStepDown method decrements, by one step, the volume level of the audio stream that enters or leaves the audio endpoint device.
IAudioEndpointVolume::VolumeStepUp	The VolumeStepUp method increments, by one step, the volume level of the audio stream that enters or leaves the audio endpoint device.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

Core Audio Interfaces

EndpointVolume API

IMMDevice::Activate

ISimpleAudioVolume Interface

IAudioEndpointVolume::GetChannelCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelCount** method gets a count of the channels in the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT GetChannelCount(
  UINT *pnChannelCount
);
```

Parameters

`pnChannelCount`

Pointer to a **UINT** variable into which the method writes the channel count.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pnChannelCount* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetChannelVolumeLevel method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelVolumeLevel** method gets the volume level, in decibels, of the specified channel in the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT GetChannelVolumeLevel(
  UINT  nChannel,
  float *pfLevelDB
);
```

Parameters

`nChannel`

The channel number. If the audio stream has *n* channels, the channels are numbered from 0 to *n*– 1. To obtain the number of channels in the stream, call the IAudioEndpointVolume::GetChannelCount method.

`pfLevelDB`

Pointer to a **float** variable into which the method writes the volume level in decibels. To get the range of volume levels obtained from this method, call the IAudioEndpointVolume::GetVolumeRange method.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *nChannel* is greater than or equal to the number of channels in the stream.
E_P OIN TER	Parameter *pfLevelDB* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]

Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetChannelCount

IAudioEndpointVolume::GetVolumeRange

IAudioEndpointVolume::GetChannelVolumeLevelScalar method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelVolumeLevelScalar** method gets the normalized, audio-tapered volume level of the specified channel of the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT GetChannelVolumeLevelScalar(
  UINT  nChannel,
  float *pfLevel
);
```

Parameters

`nChannel`

The channel number. If the audio stream contains *n* channels, the channels are numbered from 0 to *n*– 1. To obtain the number of channels, call the IAudioEndpointVolume::GetChannelCount method.

`pfLevel`

Pointer to a **float** variable into which the method writes the volume level. The level is expressed as a normalized value in the range from 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *nChannel* is greater than or equal to the number of channels in the stream.
E_P OIN TER	Parameter *pfLevel* is **NULL**.

Remarks

The volume level is normalized to the range from 0.0 to 1.0, where 0.0 is the minimum volume level and 1.0 is the maximum level. Within this range, the relationship of the normalized volume level to the attenuation of signal amplitude is described by a nonlinear, audio-tapered curve. Note that the shape of the curve might change in future versions of Windows. For more information about audio-tapered curves, see Audio-Tapered Volume

Controls.

The normalized volume levels that are retrieved by this method are suitable to represent the positions of volume controls in application windows and on-screen displays.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetChannelCount

IAudioEndpointVolume::GetMasterVolumeLevel method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMasterVolumeLevel** method gets the master volume level, in decibels, of the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT GetMasterVolumeLevel(
  float *pfLevelDB
);
```

Parameters

`pfLevelDB`

Pointer to the master volume level. This parameter points to a **float** variable into which the method writes the volume level in decibels. To get the range of volume levels obtained from this method, call the IAudioEndpointVolume::GetVolumeRange method.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P OIN TER	Parameter *pfLevelDB* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetMasterVolumeLevelScalar method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMasterVolumeLevelScalar** method gets the master volume level of the audio stream that enters or leaves the audio endpoint device. The volume level is expressed as a normalized, audio-tapered value in the range from 0.0 to 1.0.

Syntax

```
HRESULT GetMasterVolumeLevelScalar(
  float *pfLevel
);
```

Parameters

`pfLevel`

Pointer to the master volume level. This parameter points to a **float** variable into which the method writes the volume level. The level is expressed as a normalized value in the range from 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pfLevel* is **NULL**.

Remarks

The volume level is normalized to the range from 0.0 to 1.0, where 0.0 is the minimum volume level and 1.0 is the maximum level. Within this range, the relationship of the normalized volume level to the attenuation of signal amplitude is described by a nonlinear, audio-tapered curve. Note that the shape of the curve might change in future versions of Windows. For more information about audio-tapered curves, see Audio-Tapered Volume Controls.

The normalized volume levels that are retrieved by this method are suitable to represent the positions of volume controls in application windows and on-screen displays.

For a code example that calls **GetMasterVolumeLevelScalar**, see Endpoint Volume Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetMute method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMute** method gets the muting state of the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT GetMute(
  BOOL *pbMute
);
```

Parameters

`pbMute`

Pointer to a **BOOL** variable into which the method writes the muting state. If **pbMute* is **TRUE**, the stream is muted. If **FALSE**, the stream is not muted.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_P **OIN** **TER**	Parameter *pbMute* is **NULL**.

Remarks

For a code example that calls **GetMute**, see Endpoint Volume Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetVolumeRange method

1/11/2020 • 2 minutes to read • Edit Online

The **GetVolumeRange** method gets the volume range, in decibels, of the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT GetVolumeRange(
  float *pflVolumeMindB,
  float *pflVolumeMaxdB,
  float *pflVolumeIncrementdB
);
```

Parameters

`pflVolumeMindB`

Pointer to the minimum volume level. This parameter points to a **float** variable into which the method writes the minimum volume level in decibels. This value remains constant for the lifetime of the IAudioEndpointVolume interface instance.

`pflVolumeMaxdB`

Pointer to the maximum volume level. This parameter points to a **float** variable into which the method writes the maximum volume level in decibels. This value remains constant for the lifetime of the **IAudioEndpointVolume** interface instance.

`pflVolumeIncrementdB`

Pointer to the volume increment. This parameter points to a **float** variable into which the method writes the volume increment in decibels. This increment remains constant for the lifetime of the **IAudioEndpointVolume** interface instance.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pfLevelMinDB*, *pfLevelMaxDB*, or *pfVolumeIncrementDB* is **NULL**.

Remarks

The volume range from vmin = **pfLevelMinDB* to vmax = **pfLevelMaxDB* is divided into *n* uniform intervals of size vinc = **pfVolumeIncrementDB*, where

n = (vmax − vmin) / vinc.

The values vmin, vmax, and vinc are measured in decibels. The client can set the volume level to one of n + 1 discrete values in the range from vmin to vmax.

The IAudioEndpointVolume::SetChannelVolumeLevel and IAudioEndpointVolume::SetMasterVolumeLevel methods accept only volume levels in the range from vmin to vmax. If the caller specifies a volume level outside of this range, the method fails and returns E_INVALIDARG. If the caller specifies a volume level that falls between two steps in the volume range, the method sets the endpoint volume level to the step that lies closest to the requested volume level and returns S_OK. However, a subsequent call to IAudioEndpointVolume::GetChannelVolumeLevel or IAudioEndpointVolume::GetMasterVolumeLevel retrieves the volume level requested by the previous call to **SetChannelVolumeLevel** or **SetMasterVolumeLevel**, not the step value.

If the volume control is implemented in hardware, **GetVolumeRange** describes the range and granularity of the hardware volume settings. In contrast, the steps that are reported by the IEndpointVolume::GetVolumeStepInfo method correspond to points on an audio-tapered curve that are calculated in software by the IEndpointVolume::VolumeStepDown and IEndpointVolume::VolumeStepUp methods. Either method first calculates the idealized volume level that corresponds to the next point on the curve. Next, the method selects the hardware volume setting that is the best approximation to the idealized level. For more information about audio-tapered curves, see Audio-Tapered Volume Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetChannelVolumeLevel

IAudioEndpointVolume::GetMasterVolumeLevel

IAudioEndpointVolume::SetChannelVolumeLevel

IAudioEndpointVolume::SetMasterVolumeLevel

IEndpointVolume::GetVolumeStepInfo

IEndpointVolume::VolumeStepDown

IEndpointVolume::VolumeStepUp

IAudioEndpointVolume::GetVolumeStepInfo method

1/11/2020 • 2 minutes to read • Edit Online

The **GetVolumeStepInfo** method gets information about the current step in the volume range.

Syntax

```
HRESULT GetVolumeStepInfo(
  UINT *pnStep,
  UINT *pnStepCount
);
```

Parameters

`pnStep`

Pointer to a **UINT** variable into which the method writes the current step index. This index is a value in the range from 0 to **pStepCount*– 1, where 0 represents the minimum volume level and **pStepCount*– 1 represents the maximum level.

`pnStepCount`

Pointer to a **UINT** variable into which the method writes the number of steps in the volume range. This number remains constant for the lifetime of the IAudioEndpointVolume interface instance.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pnStep* and *pnStepCount* are both **NULL**.

Remarks

This method represents the volume level of the audio stream that enters or leaves the audio endpoint device as an index or "step" in a range of discrete volume levels. Output value **pnStepCount* is the number of steps in the range. Output value **pnStep* is the step index of the current volume level. If the number of steps is n = **pnStepCount*, then step index **pnStep* can assume values from 0 (minimum volume) to n – 1 (maximum volume).

Over the range from 0 to n – 1, successive intervals between adjacent steps do not necessarily represent uniform volume increments in either linear signal amplitude or decibels. In Windows Vista, **GetVolumeStepInfo** defines the relationship of index to volume level (signal amplitude) to be an audio-tapered curve. Note that the shape of the curve might change in future versions of Windows. For more information about audio-tapered curves, see Audio-Tapered Volume Controls.

Audio applications can call the IAudioEndpointVolume::VolumeStepUp and

IAudioEndpointVolume::VolumeStepDown methods to increase or decrease the volume level by one interval. Either method first calculates the idealized volume level that corresponds to the next point on the audio-tapered curve. Next, the method selects the endpoint volume setting that is the best approximation to the idealized level. To obtain the range and granularity of the endpoint volume settings, call the IEndpointVolume::GetVolumeRange method. If the audio endpoint device implements a hardware volume control, **GetVolumeRange** describes the hardware volume settings. Otherwise, the EndpointVolume API implements the endpoint volume control in software, and **GetVolumeRange** describes the volume settings of the software-implemented control.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::VolumeStepDown

IAudioEndpointVolume::VolumeStepUp

IAudioEndpointVolume::QueryHardwareSupport method

1/11/2020 • 2 minutes to read • Edit Online

The *QueryHardwareSupport* method queries the audio endpoint device for its hardware-supported functions.

Syntax

```
HRESULT QueryHardwareSupport(
  DWORD *pdwHardwareSupportMask
);
```

Parameters

`pdwHardwareSupportMask`

Pointer to a **DWORD** variable into which the method writes a hardware support mask that indicates the hardware capabilities of the audio endpoint device. The method can set the mask to 0 or to the bitwise-OR combination of one or more ENDPOINT_HARDWARE_SUPPORT_XXX constants.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pdwHardwareSupportMask* is **NULL**.

Remarks

This method indicates whether the audio endpoint device implements the following functions in hardware:

- Volume control
- Mute control
- Peak meter

The system automatically substitutes a software implementation for any function in the preceding list that the endpoint device does not implement in hardware.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]

Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::RegisterControlChangeNotify method

1/11/2020 • 2 minutes to read • Edit Online

The **RegisterControlChangeNotify** method registers a client's notification callback interface.

Syntax

```
HRESULT RegisterControlChangeNotify(
  IAudioEndpointVolumeCallback *pNotify
);
```

Parameters

`pNotify`

Pointer to the IAudioEndpointVolumeCallback interface that the client is registering for notification callbacks. If the **RegisterControlChangeNotify** method succeeds, it calls the AddRef method on the client's **IAudioEndpointVolumeCallback** interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pNotify* is **NULL**.

Remarks

This method registers an IAudioEndpointVolumeCallback interface to be called by the system when the volume level or muting state of an endpoint changes. The caller implements the **IAudioEndpointVolumeCallback** interface.

When notifications are no longer needed, the client can call the IAudioEndpointVolume::UnregisterControlChangeNotify method to terminate the notifications.

Before the client releases its final reference to the IAudioEndpointVolumeCallback interface, it should call UnregisterControlChangeNotify to unregister the interface. Otherwise, the application leaks the resources held by the **IAudioEndpointVolumeCallback** and IAudioEndpointVolume objects. Note that **RegisterControlChangeNotify** calls the client's IAudioEndpointVolumeCallback::AddRef method, and **UnregisterControlChangeNotify** calls the IAudioEndpointVolumeCallback::Release method. If the client errs by releasing its reference to the **IAudioEndpointVolumeCallback** interface before calling **UnregisterControlChangeNotify**, the **IAudioEndpointVolume** object never releases its reference to the **IAudioEndpointVolumeCallback** interface. For example, a poorly designed **IAudioEndpointVolumeCallback**

implementation might call **UnregisterControlChangeNotify** from the destructor for the **IAudioEndpointVolumeCallback** object. In this case, the client will not call **UnregisterControlChangeNotify** until the **IAudioEndpointVolume** object releases its reference to the **IAudioEndpointVolumeCallback** interface, and the **IAudioEndpointVolume** object will not release its reference to the **IAudioEndpointVolumeCallback** interface until the client calls **UnregisterControlChangeNotify**. For more information about the **AddRef** and **Release** methods, see the discussion of the IUnknown interface in the Windows SDK documentation.

In addition, the client should call UnregisterControlChangeNotify before releasing the final reference to the IAudioEndpointVolume object. Otherwise, the object leaks the storage that it allocated to hold the registration information. After registering a notification interface, the client continues to receive notifications for only as long as the **IAudioEndpointVolume** object exists.

For a code example that calls **RegisterControlChangeNotify**, see Endpoint Volume Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::UnregisterControlChangeNotify

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolume::SetChannelVolumeLevel method

1/11/2020 • 2 minutes to read • Edit Online

The **SetChannelVolumeLevel** method sets the volume level, in decibels, of the specified channel of the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT SetChannelVolumeLevel(
  UINT    nChannel,
  float   fLevelDB,
  LPCGUID pguidEventContext
);
```

Parameters

`nChannel`

The channel number. If the audio stream contains *n* channels, the channels are numbered from 0 to *n*– 1. To obtain the number of channels, call the IAudioEndpointVolume::GetChannelCount method.

`fLevelDB`

The new volume level in decibels. To obtain the range and granularity of the volume levels that can be set by this method, call the IAudioEndpointVolume::GetVolumeRange method.

`pguidEventContext`

Context value for the IAudioEndpointVolumeCallback::OnNotify method. This parameter points to an event-context GUID. If the **SetChannelVolumeLevel** call changes the volume level of the endpoint, all clients that have registered IAudioEndpointVolumeCallback interfaces with that endpoint will receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the volume-change event. If the caller supplies a **NULL** pointer for this parameter, the notification routine receives the context GUID value GUID_NULL.

Return value

If the method succeeds, it returns S_OK. If the method fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *nChannel* is greater than or equal to the number of channels in the stream; or parameter *fLevelDB* lies outside of the volume range supported by the device.

E_OUTOFMEMORY	Out of memory.

Remarks

If volume level *fLevelDB* falls outside of the volume range reported by the **IAudioEndpointVolume::GetVolumeRange** method, the **SetChannelVolumeLevel** call fails and returns error code E_INVALIDARG.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetChannelCount

IAudioEndpointVolume::GetVolumeRange

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolumeCallback::OnNotify

IAudioEndpointVolume::SetChannelVolumeLevelScalar method

1/11/2020 • 2 minutes to read • Edit Online

The **SetChannelVolumeLevelScalar** method sets the normalized, audio-tapered volume level of the specified channel in the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT SetChannelVolumeLevelScalar(
  UINT    nChannel,
  float   fLevel,
  LPCGUID pguidEventContext
);
```

Parameters

`nChannel`

The channel number. If the audio stream contains *n* channels, the channels are numbered from 0 to *n*– 1. To obtain the number of channels, call the IAudioEndpointVolume::GetChannelCount method.

`fLevel`

The volume level. The volume level is expressed as a normalized value in the range from 0.0 to 1.0.

`pguidEventContext`

Context value for the IAudioEndpointVolumeCallback::OnNotify method. This parameter points to an event-context GUID. If the **SetChannelVolumeLevelScalar** call changes the volume level of the endpoint, all clients that have registered IAudioEndpointVolumeCallback interfaces with that endpoint will receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the volume-change event. If the caller supplies a **NULL** pointer for this parameter, the notification routine receives the context GUID value GUID_NULL.

Return value

If the method succeeds, it returns S_OK. If the method fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *nChannel* is greater than or equal to the number of channels in the stream; or parameter *fLevel* is outside the range from 0.0 to 1.0.

E_OUTOFMEMORY	Out of memory.

Remarks

The volume level is normalized to the range from 0.0 to 1.0, where 0.0 is the minimum volume level and 1.0 is the maximum level. Within this range, the relationship of the normalized volume level to the attenuation of signal amplitude is described by a nonlinear, audio-tapered curve. Note that the shape of the curve might change in future versions of Windows. For more information about audio-tapered curves, see Audio-Tapered Volume Controls.

The normalized volume levels that are passed to this method are suitable to represent the positions of volume controls in application windows and on-screen displays.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetChannelCount

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolumeCallback::OnNotify

IAudioEndpointVolume::SetMasterVolumeLevel method

1/11/2020 • 2 minutes to read • Edit Online

The **SetMasterVolumeLevel** method sets the master volume level, in decibels, of the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT SetMasterVolumeLevel(
  float   fLevelDB,
  LPCGUID pguidEventContext
);
```

Parameters

`fLevelDB`

The new master volume level in decibels. To obtain the range and granularity of the volume levels that can be set by this method, call the IAudioEndpointVolume::GetVolumeRange method.

`pguidEventContext`

Context value for the IAudioEndpointVolumeCallback::OnNotify method. This parameter points to an event-context GUID. If the **SetMasterVolumeLevel** call changes the volume level of the endpoint, all clients that have registered IAudioEndpointVolumeCallback interfaces with that endpoint will receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the volume-change event. If the caller supplies a **NULL** pointer for this parameter, the notification routine receives the context GUID value GUID_NULL.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *fLevelDB* lies outside of the volume range supported by the device.
E_OUTOFMEMORY	Out of memory.

Remarks

If volume level *fLevelDB* falls outside of the volume range reported by the **IAudioEndpointVolume::GetVolumeRange** method, the **SetMasterVolumeLevel** call fails and returns error code E_INVALIDARG.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetVolumeRange

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolumeCallback::OnNotify

IAudioEndpointVolume::SetMasterVolumeLevelScalar method

1/11/2020 • 2 minutes to read • Edit Online

The **SetMasterVolumeLevelScalar** method sets the master volume level of the audio stream that enters or leaves the audio endpoint device. The volume level is expressed as a normalized, audio-tapered value in the range from 0.0 to 1.0.

Syntax

```
HRESULT SetMasterVolumeLevelScalar(
  float   fLevel,
  LPCGUID pguidEventContext
);
```

Parameters

`fLevel`

The new master volume level. The level is expressed as a normalized value in the range from 0.0 to 1.0.

`pguidEventContext`

Context value for the IAudioEndpointVolumeCallback::OnNotify method. This parameter points to an event-context GUID. If the **SetMasterVolumeLevelScalar** call changes the volume level of the endpoint, all clients that have registered IAudioEndpointVolumeCallback interfaces with that endpoint will receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the volume-change event. If the caller supplies a **NULL** pointer for this parameter, the notification routine receives the context GUID value GUID_NULL.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *fLevel* is outside the range from 0.0 to 1.0.
E_OUTOFMEMORY	Out of memory.

Remarks

The volume level is normalized to the range from 0.0 to 1.0, where 0.0 is the minimum volume level and 1.0 is the maximum level. Within this range, the relationship of the normalized volume level to the attenuation of signal amplitude is described by a nonlinear, audio-tapered curve. Note that the shape of the curve might change in future versions of Windows. For more information about audio-tapered curves, see Audio-Tapered Volume Controls.

The normalized volume levels that are passed to this method are suitable to represent the positions of volume controls in application windows and on-screen displays.

For a code example that calls **SetMasterVolumeLevelScalar**, see Endpoint Volume Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolumeCallback::OnNotify

IAudioEndpointVolume::SetMute method

1/11/2020 • 2 minutes to read • Edit Online

The **SetMute** method sets the muting state of the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT SetMute(
  BOOL    bMute,
  LPCGUID pguidEventContext
);
```

Parameters

`bMute`

The new muting state. If *bMute* is **TRUE**, the method mutes the stream. If **FALSE**, the method turns off muting.

`pguidEventContext`

Context value for the IAudioEndpointVolumeCallback::OnNotify method. This parameter points to an event-context GUID. If the **SetMute** call changes the muting state of the endpoint, all clients that have registered IAudioEndpointVolumeCallback interfaces with that endpoint will receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the control-change event. If the caller supplies a **NULL** pointer for this parameter, the notification routine receives the context GUID value GUID_NULL.

Return value

If the method succeeds and the muting state changes, the method returns S_OK. If the method succeeds and the new muting state is the same as the previous muting state, the method returns S_FALSE. If the method fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_OUTOFMEMORY	Out of memory.

Remarks

For a code example that calls **SetMute**, see Endpoint Volume Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolumeCallback::OnNotify

IAudioEndpointVolume::UnregisterControlChangeNotify method

1/11/2020 • 2 minutes to read • Edit Online

The **UnregisterControlChangeNotify** method deletes the registration of a client's notification callback interface that the client registered in a previous call to the IAudioEndpointVolume::RegisterControlChangeNotify method.

Syntax

```
HRESULT UnregisterControlChangeNotify(
  IAudioEndpointVolumeCallback *pNotify
);
```

Parameters

`pNotify`

Pointer to the client's IAudioEndpointVolumeCallback interface. The client passed this same interface pointer to the endpoint volume object in a previous call to the IAudioEndpointVolume::RegisterControlChangeNotify method. If the **UnregisterControlChangeNotify** method succeeds, it calls the Release method on the client's **IAudioEndpointVolumeCallback** interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pNotify* is **NULL**.

Remarks

Before the client releases its final reference to the IAudioEndpointVolumeCallback interface, it should call **UnregisterControlChangeNotify** to unregister the interface. Otherwise, the application leaks the resources held by the **IAudioEndpointVolumeCallback** and IAudioEndpointVolume objects. Note that the IAudioEndpointVolume::RegisterControlChangeNotify method calls the client's IAudioEndpointVolumeCallback::AddRef method, and **UnregisterControlChangeNotify** calls the IAudioEndpointVolumeCallback::Release method. If the client errs by releasing its reference to the **IAudioEndpointVolumeCallback** interface before calling **UnregisterControlChangeNotify**, the **IAudioEndpointVolume** object never releases its reference to the **IAudioEndpointVolumeCallback** interface. For example, a poorly designed **IAudioEndpointVolumeCallback** implementation might call **UnregisterControlChangeNotify** from the destructor for the **IAudioEndpointVolumeCallback** object. In this case, the client will not call **UnregisterControlChangeNotify** until the **IAudioEndpointVolume** object releases its reference to the **IAudioEndpointVolumeCallback** interface, and the **IAudioEndpointVolume** object will not

release its reference to the **IAudioEndpointVolumeCallback** interface until the client calls **UnregisterControlChangeNotify**. For more information about the **AddRef** and **Release** methods, see the discussion of the IUnknown interface in the Windows SDK documentation.

For a code example that calls **UnregisterControlChangeNotify**, see Endpoint Volume Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::RegisterControlChangeNotify

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolume::VolumeStepDown method

1/11/2020 • 2 minutes to read • Edit Online

The **VolumeStepDown** method decrements, by one step, the volume level of the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT VolumeStepDown(
  LPCGUID pguidEventContext
);
```

Parameters

`pguidEventContext`

Context value for the IAudioEndpointVolumeCallback::OnNotify method. This parameter points to an event-context GUID. If the **VolumeStepDown** call changes the volume level of the endpoint, all clients that have registered IAudioEndpointVolumeCallback interfaces with that endpoint will receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the volume-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_OUTOFMEMORY	Out of memory.

Remarks

To obtain the current volume step and the total number of steps in the volume range, call the IAudioEndpointVolume::GetVolumeStepInfo method.

If the volume level is already at the lowest step in the volume range, the call to **VolumeStepDown** has no effect and returns status code S_OK.

Successive intervals between adjacent steps do not necessarily represent uniform volume increments in either linear signal amplitude or decibels. In Windows Vista, **VolumeStepDown** defines the relationship of step index to volume level (signal amplitude) to be an audio-tapered curve. Note that the shape of the curve might change in future versions of Windows. For more information about audio-tapered curves, see Audio-Tapered Volume

Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetVolumeStepInfo

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolumeCallback::OnNotify

IAudioEndpointVolume::VolumeStepUp method

1/11/2020 • 2 minutes to read • Edit Online

The **VolumeStepUp** method increments, by one step, the volume level of the audio stream that enters or leaves the audio endpoint device.

Syntax

```
HRESULT VolumeStepUp(
  LPCGUID pguidEventContext
);
```

Parameters

`pguidEventContext`

Context value for the IAudioEndpointVolumeCallback::OnNotify method. This parameter points to an event-context GUID. If the **VolumeStepUp** call changes the volume level of the endpoint, all clients that have registered IAudioEndpointVolumeCallback interfaces with that endpoint will receive notifications. In its implementation of the **OnNotify** method, a client can inspect the event-context GUID to discover whether it or another client is the source of the volume-change event. If the caller supplies a **NULL** pointer for this parameter, the client's notification method receives a **NULL** context pointer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_OUTOFMEMORY	Out of memory.

Remarks

To obtain the current volume step and the total number of steps in the volume range, call the IAudioEndpointVolume::GetVolumeStepInfo method.

If the volume level is already at the highest step in the volume range, the call to **VolumeStepUp** has no effect and returns status code S_OK.

Successive intervals between adjacent steps do not necessarily represent uniform volume increments in either linear signal amplitude or decibels. In Windows Vista, **VolumeStepUp** defines the relationship of step index to volume level (signal amplitude) to be an audio-tapered curve. Note that the shape of the curve might change in future versions of Windows. For more information about audio-tapered curves, see Audio-Tapered Volume

Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolume Interface

IAudioEndpointVolume::GetVolumeStepInfo

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolumeCallback::OnNotify

IAudioEndpointVolumeCallback interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioEndpointVolumeCallback** interface provides notifications of changes in the volume level and muting state of an audio endpoint device. Unlike the other interfaces in this section, which are implemented by the WASAPI system component, an EndpointVolume API client implements the **IAudioEndpointVolumeCallback** interface. To receive event notifications, the client passes a pointer to its **IAudioEndpointVolumeCallback** interface to the IAudioEndpointVolume::RegisterControlChangeNotify method.

After registering its **IAudioEndpointVolumeCallback** interface, the client receives event notifications in the form of callbacks through the **OnNotify** method in the interface. These event notifications occur when one of the following methods causes a change in the volume level or muting state of an endpoint device:

- IAudioEndpointVolume::SetChannelVolumeLevel
- IAudioEndpointVolume::SetChannelVolumeLevelScalar
- IAudioEndpointVolume::SetMasterVolumeLevel
- IAudioEndpointVolume::SetMasterVolumeLevelScalar
- IAudioEndpointVolume::SetMute
- IAudioEndpointVolume::VolumeStepDown
- IAudioEndpointVolume::VolumeStepUp

If an audio endpoint device implements hardware volume and mute controls, the **IAudioEndpointVolume** interface uses the hardware controls to manage the device's volume. Otherwise, the **IAudioEndpointVolume** interface implements volume and mute controls in software, transparently to the client.

If a device has hardware volume and mute controls, changes made to the volume and mute settings through the methods in the preceding list affect the device's volume in both shared mode and exclusive mode. If a device lacks hardware volume and mute controls, changes made to the software volume and mute controls through these methods affect the device's volume in shared mode, but not in exclusive mode. In exclusive mode, the client and the device exchange audio data directly, bypassing the software controls. However, changes made to the software controls through these methods generate event notifications regardless of whether the device is operating in shared mode or in exclusive mode. Changes made to the software volume and mute controls while the device operates in exclusive mode take effect when the device switches to shared mode.

To determine whether a device has hardware volume and mute controls, call the IAudioEndpointVolume::QueryHardwareSupport method.

In implementing the **IAudioEndpointVolumeCallback** interface, the client should observe these rules to avoid deadlocks:

- The methods in the interface must be nonblocking. The client should never wait on a synchronization object during an event callback.
- The client should never call the IAudioEndpointVolume::UnregisterControlChangeNotify method during an event callback.
- The client should never release the final reference on an EndpointVolume API object during an event callback.

For a code example that implements the **IAudioEndpointVolumeCallback** interface, see Endpoint Volume Controls.

Inheritance

The **IAudioEndpointVolumeCallback** interface inherits from the IUnknown interface.

IAudioEndpointVolumeCallback also has these types of members:

- Methods

Methods

The **IAudioEndpointVolumeCallback** interface has these methods.

METHOD	DESCRIPTION
IAudioEndpointVolumeCallback::OnNotify	The OnNotify method notifies the client that the volume level or muting state of the audio endpoint device has changed.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

Core Audio Interfaces

EndpointVolume API

IAudioEndpointVolume::RegisterControlChangeNotify

IAudioEndpointVolume::UnregisterControlChangeNotify

IAudioEndpointVolumeCallback::OnNotify method

1/11/2020 • 2 minutes to read • Edit Online

The **OnNotify** method notifies the client that the volume level or muting state of the audio endpoint device has changed.

Syntax

```
HRESULT OnNotify(
  PAUDIO_VOLUME_NOTIFICATION_DATA pNotify
);
```

Parameters

`pNotify`

Pointer to the volume-notification data. This parameter points to a structure of type AUDIO_VOLUME_NOTIFICATION_DATA.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

The *pNotify* parameter points to a structure that describes the volume change event that initiated the call to **OnNotify**. This structure contains an event-context GUID. This GUID enables a client to distinguish between a volume (or muting) change that it initiated and one that some other client initiated. When calling an IAudioEndpointVolume method that changes the volume level of the stream, a client passes in a pointer to an event-context GUID that its implementation of the **OnNotify** method can recognize. The structure pointed to by *pNotify* contains this context GUID. If the client that changes the volume level supplies a **NULL** pointer value for the pointer to the event-context GUID, the value of the event-context GUID in the structure pointed to by *pNotify* is GUID_NULL.

The Windows 7, the system's volume user interface does not specify GUID_NULL when it changes the volume in the system. A third-party OSD application can differentiate between master volume control changes that result from the system's volume user interface, and other volume changes such as changes from the built-in volume control handler.

For a code example that implements the **OnNotify** method, see Endpoint Volume Controls.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]

Target Platform	Windows
Header	endpointvolume.h

See also

AUDIO_VOLUME_NOTIFICATION_DATA

IAudioEndpointVolume Interface

IAudioEndpointVolumeCallback Interface

IAudioEndpointVolumeEx interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioEndpointVolumeEx** interface provides volume controls on the audio stream to or from a device endpoint.

A client obtains a reference to the **IAudioEndpointVolumeEx** interface of an endpoint device by calling the IMMDevice::Activate method with parameter *iid* set to REFIID IID_IAudioEndpointVolumeEx.

Inheritance

The **IAudioEndpointVolumeEx** interface inherits from IAudioEndpointVolume. **IAudioEndpointVolumeEx** also has these types of members:

- Methods

Methods

The **IAudioEndpointVolumeEx** interface has these methods.

METHOD	DESCRIPTION
IAudioEndpointVolumeEx::GetVolumeRangeChannel	The GetVolumeRangeChannel method gets the volume range for a specified channel.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	endpointvolume.h

See also

Core Audio Interfaces

EndpointVolume API

IAudioEndpointVolume

IMMDevice::Activate

ISimpleAudioVolume Interface

IAudioEndpointVolumeEx::GetVolumeRangeChannel method

1/11/2020 • 2 minutes to read • Edit Online

The **GetVolumeRangeChannel** method gets the volume range for a specified channel.

Syntax

```
HRESULT GetVolumeRangeChannel(
  UINT  iChannel,
  float *pflVolumeMindB,
  float *pflVolumeMaxdB,
  float *pflVolumeIncrementdB
);
```

Parameters

`iChannel`

The channel number for which to get the volume range. If the audio stream has *n* channels, the channels are numbered from 0 to *n*– 1. To obtain the number of channels in the stream, call the IAudioEndpointVolume::GetChannelCount method.

`pflVolumeMindB`

Receives the minimum volume level for the channel, in decibels.

`pflVolumeMaxdB`

Receives the maximum volume level for the channel, in decibels.

`pflVolumeIncrementdB`

Receives the volume increment for the channel, in decibels.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pfLevelMinDB*, *pfLevelMaxDB*, or *pfVolumeIncrementDB* is **NULL**.

Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioEndpointVolumeEx

IAudioMeterInformation interface

1/23/2020 • 2 minutes to read • Edit Online

The **IAudioMeterInformation** interface represents a peak meter on an audio stream to or from an audio endpoint device. The client obtains a reference to the **IAudioMeterInformation** interface on an endpoint object by calling the IMMDevice::Activate method with parameter *iid* set to REFIID IID_IAudioMeterInformation.

If the adapter device that streams audio data to or from the endpoint device implements a hardware peak meter, the **IAudioMeterInformation** interface uses that meter to monitor the peak levels in the audio stream. If the audio device lacks a hardware peak meter, the audio engine automatically implements the peak meter in software, transparently to the client.

If a device has a hardware peak meter, a client can use the methods in the **IAudioMeterInformation** interface to monitor the device's peak levels in both shared mode and exclusive mode. If a device lacks a hardware peak meter, a client can use those methods to monitor the device's peak levels in shared mode, but not in exclusive mode. In exclusive mode, the client and the device exchange audio data directly, bypassing the software peak meter. In exclusive mode, a software peak meter always reports a peak value of 0.0.

To determine whether a device has a hardware peak meter, call the IAudioMeterInformation::QueryHardwareSupport method.

For a rendering endpoint device, the **IAudioMeterInformation** interface monitors the peak levels in the output stream before the stream is attenuated by the endpoint volume controls. Similarly, for a capture endpoint device, the interface monitors the peak levels in the input stream before the stream is attenuated by the endpoint volume controls.

The peak values reported by the methods in the **IAudioMeterInformation** interface are normalized to the range from 0.0 to 1.0. For example, if a PCM stream contains 16-bit samples, and the peak sample value during a particular metering period is –8914, then the absolute value recorded by the peak meter is 8914, and the normalized peak value reported by the **IAudioMeterInformation** interface is 8914/32768 = 0.272.

For a code example that uses the **IAudioMeterInformation** interface, see Peak Meters.

Inheritance

The **IAudioMeterInformation** interface inherits from the IUnknown interface. **IAudioMeterInformation** also has these types of members:

- Methods

Methods

The **IAudioMeterInformation** interface has these methods.

METHOD	DESCRIPTION
IAudioMeterInformation::GetChannelsPeakValues	The GetChannelsPeakValues method gets the peak sample values for all the channels in the audio stream.
IAudioMeterInformation::GetMeteringChannelCount	The GetMeteringChannelCount method gets the number of channels in the audio stream that are monitored by peak meters.

METHOD	DESCRIPTION
IAudioMeterInformation::GetPeakValue	The GetPeakValue method gets the peak sample value for the channels in the audio stream.
IAudioMeterInformation::QueryHardwareSupport	The QueryHardwareSupport method queries the audio endpoint device for its hardware-supported functions.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

Core Audio Interfaces

EndpointVolume API

IMMDevice::Activate

IAudioMeterInformation::GetChannelsPeakValues method

1/11/2020 • 2 minutes to read • Edit Online

The **GetChannelsPeakValues** method gets the peak sample values for all the channels in the audio stream.

Syntax

```
HRESULT GetChannelsPeakValues(
  UINT32 u32ChannelCount,
  float  *afPeakValues
);
```

Parameters

`u32ChannelCount`

The channel count. This parameter also specifies the number of elements in the *afPeakValues* array. If the specified count does not match the number of channels in the stream, the method returns error code E_INVALIDARG.

`afPeakValues`

Pointer to an array of peak sample values. The method writes the peak values for the channels into the array. The array contains one element for each channel in the stream. The peak values are numbers in the normalized range from 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	Parameter *u32ChannelCount* does not equal the number of channels in the audio stream.
E_POINTER	Parameter *afPeakValues* is **NULL**.

Remarks

This method retrieves the peak sample values for the channels in the stream. The peak value for each channel is recorded over one device period and made available during the subsequent device period. Thus, this method always retrieves the peak values recorded during the previous device period. To obtain the device period, call the IAudioClient::GetDevicePeriod method.

Parameter *afPeakValues* points to a caller-allocated **float** array. If the stream contains *n* channels, the channels are numbered 0 to *n*– 1. The method stores the peak value for each channel in the array element whose array index matches the channel number. To get the number of channels in the audio stream that are monitored by peak meters, call the IAudioMeterInformation::GetMeteringChannelCount method.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioClient::GetDevicePeriod

IAudioMeterInformation Interface

IAudioMeterInformation::GetMeteringChannelCount

IAudioMeterInformation::GetMeteringChannelCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetMeteringChannelCount** method gets the number of channels in the audio stream that are monitored by peak meters.

Syntax

```
HRESULT GetMeteringChannelCount(
  UINT *pnChannelCount
);
```

Parameters

`pnChannelCount`

Pointer to a **UINT** variable into which the method writes the number of channels.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pnChannelCount* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioMeterInformation Interface

IAudioMeterInformation::GetPeakValue method

1/11/2020 • 2 minutes to read • Edit Online

The **GetPeakValue** method gets the peak sample value for the channels in the audio stream.

Syntax

```
HRESULT GetPeakValue(
  float *pfPeak
);
```

Parameters

`pfPeak`

Pointer to a **float** variable into which the method writes the peak sample value for the audio stream. The peak value is a number in the normalized range from 0.0 to 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pfPeak* is **NULL**.

Remarks

This method retrieves the peak sample value recorded across all of the channels in the stream. The peak value for each channel is recorded over one device period and made available during the subsequent device period. Thus, this method always retrieves the peak value recorded during the previous device period. To obtain the device period, call the IAudioClient::GetDevicePeriod method.

For a code example that uses the **GetPeakValue** method, see Peak Meters.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows

Header	endpointvolume.h

See also

IAudioClient::GetDevicePeriod

IAudioMeterInformation Interface

IAudioMeterInformation::QueryHardwareSupport method

1/11/2020 • 2 minutes to read • Edit Online

The **QueryHardwareSupport** method queries the audio endpoint device for its hardware-supported functions.

Syntax

```
HRESULT QueryHardwareSupport(
  DWORD *pdwHardwareSupportMask
);
```

Parameters

`pdwHardwareSupportMask`

Pointer to a **DWORD** variable into which the method writes a hardware support mask that indicates the hardware capabilities of the audio endpoint device. The method can set the mask to 0 or to the bitwise-OR combination of one or more ENDPOINT_HARDWARE_SUPPORT_XXX constants.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pdwHardwareSupportMask* is **NULL**.

Remarks

This method indicates whether the audio endpoint device implements the following functions in hardware:

- Volume control
- Mute control
- Peak meter

The system automatically substitutes a software implementation for any function in the preceding list that the endpoint devices does not implement in hardware.

Requirements

Minimum supported client	Windows Vista [desktop apps | UWP apps]

Minimum supported server	Windows Server 2008 [desktop apps | UWP apps]
Target Platform	Windows
Header	endpointvolume.h

See also

IAudioMeterInformation Interface

mmdeviceapi.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs mmdeviceapi.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IActivateAudioInterfaceAsyncOperation	Represents an asynchronous operation activating a WASAPI interface and provides a method to retrieve the results of the activation.
IActivateAudioInterfaceCompletionHandler	Provides a callback to indicate that activation of a WASAPI interface is complete.
IMMDevice	The IMMDevice interface encapsulates the generic features of a multimedia device resource.
IMMDeviceCollection	The IMMDeviceCollection interface represents a collection of multimedia device resources.
IMMDeviceEnumerator	The IMMDeviceEnumerator interface provides methods for enumerating multimedia device resources.
IMMEndpoint	The IMMEndpoint interface represents an audio endpoint device.
IMMNotificationClient	The IMMNotificationClient interface provides notifications when an audio endpoint device is added or removed, when the state or properties of an endpoint device change, or when there is a change in the default role assigned to an endpoint device.

Functions

TITLE	DESCRIPTION
ActivateAudioInterfaceAsync	Enables Windows Store apps to access preexisting Component Object Model (COM) interfaces in the WASAPI family.

Structures

TITLE	DESCRIPTION

TITLE	DESCRIPTION
AudioExtensionParams	This structure is passed to the Control Panel Endpoint Extension property page through IShellPropSheetExt::AddPages and is used to create endpoint PropertyPages.
DIRECTX_AUDIO_ACTIVATION_PARAMS	The DIRECTX_AUDIO_ACTIVATION_PARAMS structure specifies the initialization parameters for a DirectSound stream.

Enumerations

TITLE	DESCRIPTION
EDataFlow	The EDataFlow enumeration defines constants that indicate the direction in which audio data flows between an audio endpoint device and an application.
EndpointFormFactor	The EndpointFormFactor enumeration defines constants that indicate the general physical attributes of an audio endpoint device.
ERole	The ERole enumeration defines constants that indicate the role that the system has assigned to an audio endpoint device.

ActivateAudioInterfaceAsync function

1/11/2020 • 3 minutes to read • Edit Online

Enables Windows Store apps to access preexisting Component Object Model (COM) interfaces in the WASAPI family.

Syntax

```
HRESULT ActivateAudioInterfaceAsync(
  LPCWSTR                                deviceInterfacePath,
  REFIID                                 riid,
  PROPVARIANT                            *activationParams,
  IActivateAudioInterfaceCompletionHandler *completionHandler,
  IActivateAudioInterfaceAsyncOperation    **activationOperation
);
```

Parameters

`deviceInterfacePath`

A device interface ID for an audio device. This is normally retrieved from a DeviceInformation object or one of the methods of the MediaDevice class.

The GUIDs DEVINTERFACE_AUDIO_CAPTURE and **DEVINTERFACE_AUDIO_RENDER** represent the default audio capture and render device respectively. Call StringFromIID to convert either of these GUIDs to an **LPCWSTR** to use for this argument.

`riid`

The IID of a COM interface in the WASAPI family, such as IAudioClient.

`activationParams`

Interface-specific activation parameters. For more information, see the *pActivationParams* parameter in IMMDevice::Activate.

`completionHandler`

An interface implemented by the caller that is called by Windows when the result of the activation procedure is available.

`activationOperation`

Returns an IActivateAudioInterfaceAsyncOperation interface that represents the asynchronous operation of activating the requested **WASAPI** interface.

Return value

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

RETURN CODE	DESCRIPTION

S_OK	The underlying object and asynchronous operation were created successfully.
E_ILLEGAL_METHOD_CALL	This error may result if the function is called from an incorrect COM apartment, or if the passed IActivateAudioInterfaceCompletionHandler is not implemented on an agile object (aggregating a free-threaded marshaler).

Remarks

This function enables Windows Store apps to activate certain WASAPI COM interfaces after using Windows Runtime APIs in the **Windows.Devices** and Windows.Media.Devices namespaces to select an audio device.

For many implementations, an application must call this function from the main UI thread to activate a COM interface in the WASAPI family so that the system can show a dialog to the user. The application passes an IActivateAudioInterfaceCompletionHandler callback COM interface through *completionHandler*. Windows calls a method in the application's **IActivateAudioInterfaceCompletionHandler** interface from a worker thread in the COM Multi-threaded Apartment (MTA) when the activation results are available. The application can then call a method in the IActivateAudioInterfaceAsyncOperation interface to retrieve the result code and the requested **WASAPI** interface. There are some activations that are explicitly safe and therefore don't require that this function be called from the main UI thread. These explicitly safe activations include:

- Calling **ActivateAudioInterfaceAsync** with a *deviceInterfacePath* that specifies an audio render device and an *riid* that specifies the IAudioClient interface.
- Calling **ActivateAudioInterfaceAsync** with a *deviceInterfacePath* that specifies an audio render device and an *riid* that specifies the IAudioEndpointVolume interface.
- Calling **ActivateAudioInterfaceAsync** from a session 0 service. For more information, see Services.

Windows holds a reference to the application's IActivateAudioInterfaceCompletionHandler interface until the operation is complete and the application releases the IActivateAudioInterfaceAsyncOperation interface.

> **Important**
> Applications must not free the object implementing the **IActivateAudioInterfaceCompletionHandler** until the completion handler callback has executed.

Depending on which WASAPI interface is activated, this function may display a consent prompt the first time it is called. For example, when the application calls this function to activate IAudioClient to access a microphone, the purpose of the consent prompt is to get the user's permission for the app to access the microphone. For more information about the consent prompt, see Guidelines for devices that access personal data.

ActivateAudioInterfaceAsync must be called on the main UI thread so that the consent prompt can be shown. If the consent prompt can't be shown, the user can't grant device access to the app.

ActivateAudioInterfaceAsync must be called on a thread in a COM Single-Threaded Apartment (STA). The *completionHandler* that is passed into **ActivateAudioInterfaceAsync** needs to implement IAgileObject to ensure that there is no deadlock when the *completionHandler* is called from the MTA. Otherwise, an **E_ILLEGAL_METHOD_CALL** will occur.

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Target Platform	Windows
Header	mmdeviceapi.h (include Mmdevapi.idl)
Library	Mmdevapi.lib
DLL	Mmdevapi.dll
IRQL	No

See also

Core Audio Functions

IActivateAudioInterfaceAsyncOperation

IActivateAudioInterfaceCompletionHandler

AudioExtensionParams structure

1/23/2020 • 2 minutes to read • Edit Online

This structure is passed to the Control Panel Endpoint Extension property page through IShellPropSheetExt::AddPages and is used to create endpoint PropertyPages.

Syntax

```
typedef struct __MIDL___MIDL_itf_mmdeviceapi_0000_0008_0001 {
  LPARAM    AddPageParam;
  IMMDevice *pEndpoint;
  IMMDevice *pPnpInterface;
  IMMDevice *pPnpDevnode;
} AudioExtensionParams;
```

Members

`AddPageParam`

The add page param.

`pEndpoint`

Pointer to the end point.

`pPnpInterface`

Pointer to the Pnp interface.

`pPnpDevnode`

Pointer to the Pnp devnode.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	mmdeviceapi.h (include Mmdevapi.idl)

See also

Core Audio Structures

IShellPropSheetExt::AddPages

DIRECTX_AUDIO_ACTIVATION_PARAMS structure

1/23/2020 • 2 minutes to read • Edit Online

The **DIRECTX_AUDIO_ACTIVATION_PARAMS** structure specifies the initialization parameters for a DirectSound stream.

Syntax

```
typedef struct tagDIRECTX_AUDIO_ACTIVATION_PARAMS {
  DWORD cbDirectXAudioActivationParams;
  GUID  guidAudioSession;
  DWORD dwAudioStreamFlags;
} DIRECTX_AUDIO_ACTIVATION_PARAMS, *PDIRECTX_AUDIO_ACTIVATION_PARAMS;
```

Members

`cbDirectXAudioActivationParams`

The size, in bytes, of the **DIRECTX_AUDIO_ACTIVATION_PARAMS** structure. Set this member to sizeof(DIRECTX_AUDIO_ACTIVATION_PARAMS).

`guidAudioSession`

Session GUID. This member is a GUID value that identifies the audio session that the stream belongs to. If the GUID identifies a session that has been previously opened, the method adds the stream to that session. If the GUID does not identify an existing session, the method opens a new session and adds the stream to that session. The stream remains a member of the same session for its lifetime.

`dwAudioStreamFlags`

Stream-initialization flags. This member specifies whether the stream belongs to a cross-process session or to a session that is specific to the current process. Set this member to 0 or to the following AUDCLNT_STREAMFLAGS_XXX constant:

AUDCLNT_STREAMFLAGS_CROSSPROCESS

Remarks

This structure is used by the IMMDevice::Activate method. When activating an **IDirectSound**, **IDirectSoundCapture**, or **IBaseFilter** interface on an audio endpoint device, the **DIRECTX_AUDIO_ACTIVATION_PARAMS** structure specifies the session GUID and stream-initialization flags for the audio stream that the DirectSound module creates and encapsulates in the interface instance. During the **Activate** call, DirectSound calls the IAudioClient::Initialize method and specifies the session GUID and stream-initialization flags from the **DIRECTX_AUDIO_ACTIVATION_PARAMS** structure as input parameters.

For more information about **IDirectSound**, **IDirectSoundCapture**, and **IBaseFilter**, see the Windows SDK documentation.

For a code example that uses the **DIRECTX_AUDIO_ACTIVATION_PARAMS** structure, see Device Roles for DirectShow Applications.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	mmdeviceapi.h

See also

Core Audio Structures

IAudioClient::Initialize

IMMDevice::Activate

EDataFlow enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **EDataFlow** enumeration defines constants that indicate the direction in which audio data flows between an audio endpoint device and an application.

Syntax

```
typedef enum __MIDL___MIDL_itf_mmdeviceapi_0000_0000_0001 {
  eRender,
  eCapture,
  eAll,
  EDataFlow_enum_count
} EDataFlow;
```

Constants

eRender	Audio rendering stream. Audio data flows from the application to the audio endpoint device, which renders the stream.
eCapture	Audio capture stream. Audio data flows from the audio endpoint device that captures the stream, to the application.
eAll	Audio rendering or capture stream. Audio data can flow either from the application to the audio endpoint device, or from the audio endpoint device to the application.
EDataFlow_enum_count	The number of members in the EDataFlow enumeration (not counting the EDataFlow_enum_count member).

Remarks

The IMMDeviceEnumerator::GetDefaultAudioEndpoint, IMMDeviceEnumerator::EnumAudioEndpoints, IMMEndpoint::GetDataFlow, and IMMNotificationClient::OnDefaultDeviceChanged methods use the constants defined in the **EDataFlow** enumeration.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	mmdeviceapi.h

See also

Core Audio Enumerations

IMMDeviceEnumerator::EnumAudioEndpoints

IMMDeviceEnumerator::GetDefaultAudioEndpoint

IMMEndpoint::GetDataFlow

IMMNotificationClient::OnDefaultDeviceChanged

EndpointFormFactor enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **EndpointFormFactor** enumeration defines constants that indicate the general physical attributes of an audio endpoint device.

Syntax

```
typedef enum __MIDL___MIDL_itf_mmdeviceapi_0000_0000_0003 {
  RemoteNetworkDevice,
  Speakers,
  LineLevel,
  Headphones,
  Microphone,
  Headset,
  Handset,
  UnknownDigitalPassthrough,
  SPDIF,
  DigitalAudioDisplayDevice,
  UnknownFormFactor,
  EndpointFormFactor_enum_count
} EndpointFormFactor;
```

Constants

RemoteNetworkDevice	An audio endpoint device that the user accesses remotely through a network.
Speakers	A set of speakers.
LineLevel	An audio endpoint device that sends a line-level analog signal to a line-input jack on an audio adapter or that receives a line-level analog signal from a line-output jack on the adapter.
Headphones	A set of headphones.
Microphone	A microphone.
Headset	An earphone or a pair of earphones with an attached mouthpiece for two-way communication.
Handset	The part of a telephone that is held in the hand and that contains a speaker and a microphone for two-way communication.
UnknownDigitalPassthrough	An audio endpoint device that connects to an audio adapter through a connector for a digital interface of unknown type that transmits non-PCM data in digital pass-through mode. For more information, see Remarks.

SPDIF	An audio endpoint device that connects to an audio adapter through a Sony/Philips Digital Interface (S/PDIF) connector.
DigitalAudioDisplayDevice	An audio endpoint device that connects to an audio adapter through a High-Definition Multimedia Interface (HDMI) connector or a display port. In **Windows Vista**, this value was named HDMI.
UnknownFormFactor	An audio endpoint device with unknown physical attributes.
EndpointFormFactor_enum_count	Windows 7: Maximum number of endpoint form factors.

Remarks

The constants in this enumeration are the values that can be assigned to the PKEY_AudioEndpoint_FormFactor property.

In digital pass-through mode, a digital interface transports blocks of non-PCM data through a connection without modifying them and without attempting to interpret their contents. For more information about digital pass-through mode, see the following documentation:

- The descriptions of the WAVE_FORMAT_WMA_SPDIF and WAVE_FORMAT_DOLBY_AC3_SPDIF wave-format tags in the Windows DDK documentation.
- The white paper titled "Audio Driver Support for the WMA Pro-over-S/PDIF Format" at the Audio Device Technologies for Windows website.

For information about obtaining a description of the audio jack or connector through which an audio endpoint device connects to an audio adapter, see IKsJackDescription::GetJackDescription and IKsJackDescription2::GetJackDescription2.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	mmdeviceapi.h

See also

Core Audio Enumerations

IKsJackDescription::GetJackDescription

PKEY_AudioEndpoint_FormFactor Property

ERole enumeration

1/11/2020 • 2 minutes to read • Edit Online

The **ERole** enumeration defines constants that indicate the role that the system has assigned to an audio endpoint device.

Syntax

```
typedef enum __MIDL___MIDL_itf_mmdeviceapi_0000_0000_0002 {
  eConsole,
  eMultimedia,
  eCommunications,
  ERole_enum_count
} ERole;
```

Constants

eConsole	Games, system notification sounds, and voice commands.
eMultimedia	Music, movies, narration, and live music recording.
eCommunications	Voice communications (talking to another person).
ERole_enum_count	The number of members in the ERole enumeration (not counting the ERole_enum_count member).

Remarks

The IMMDeviceEnumerator::GetDefaultAudioEndpoint and IMMNotificationClient::OnDefaultDeviceChanged methods use the constants defined in the **ERole** enumeration.

For more information, see Device Roles.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	mmdeviceapi.h

See also

Core Audio Enumerations

IMMDeviceEnumerator::GetDefaultAudioEndpoint

IMMNotificationClient::OnDefaultDeviceChanged

IActivateAudioInterfaceAsyncOperation interface

1/23/2020 • 2 minutes to read • Edit Online

Represents an asynchronous operation activating a WASAPI interface and provides a method to retrieve the results of the activation.

Inheritance

The **IActivateAudioInterfaceAsyncOperation** interface inherits from the IUnknown interface. **IActivateAudioInterfaceAsyncOperation** also has these types of members:

- Methods

Methods

The **IActivateAudioInterfaceAsyncOperation** interface has these methods.

METHOD	DESCRIPTION
IActivateAudioInterfaceAsyncOperation::GetActivateResult	Gets the results of an asynchronous activation of a WASAPI interface initiated by an application calling the ActivateAudioInterfaceAsync function.

Remarks

When to implement:
Implemented by Windows and returned from the function ActivateAudioInterfaceAsync.

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Target Platform	Windows
Header	mmdeviceapi.h

See also

ActivateAudioInterfaceAsync

Core Audio Interfaces

IActivateAudioInterfaceCompletionHandler

IActivateAudioInterfaceAsyncOperation::GetActivateResult method

1/11/2020 • 2 minutes to read • Edit Online

Gets the results of an asynchronous activation of a WASAPI interface initiated by an application calling the ActivateAudioInterfaceAsync function.

Syntax

```
HRESULT GetActivateResult(
  HRESULT  *activateResult,
  IUnknown **activatedInterface
);
```

Parameters

`activateResult`

`activatedInterface`

Return value

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

RETURN CODE	DESCRIPTION
E_IL **LEG** **AL_** **MET** **HO** **D_C** **ALL**	The method was called before the asynchronous operation was complete.

Remarks

An application calls this method after Windows calls the ActivateCompleted method of the application's IActivateAudioInterfaceCompletionHandler interface.

The result code returned through *activateResult* may depend on the requested interface. For additional information, see IMMDevice::Activate. A result code of **E_ACCESSDENIED** might indicate that the user has not given consent to access the device in a manner required by the requested WASAPI interface.

The returned *activatedInterface* may be **NULL** if *activateResult* is not a success code.

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]

Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Target Platform	Windows
Header	mmdeviceapi.h

See also

ActivateAudioInterfaceAsync

IActivateAudioInterfaceAsyncOperation

IActivateAudioInterfaceCompletionHandler interface

1/23/2020 • 2 minutes to read • Edit Online

Provides a callback to indicate that activation of a WASAPI interface is complete.

Inheritance

The **IActivateAudioInterfaceCompletionHandler** interface inherits from the IUnknown interface. **IActivateAudioInterfaceCompletionHandler** also has these types of members:

- Methods

Methods

The **IActivateAudioInterfaceCompletionHandler** interface has these methods.

METHOD	DESCRIPTION
IActivateAudioInterfaceCompletionHandler::ActivateCompleted	Indicates that activation of a WASAPI interface is complete and results are available.

Remarks

When to implement:
An application implements this interface if it calls the ActivateAudioInterfaceAsync function.

The implementation must be agile (aggregating a free-threaded marshaler).

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Target Platform	Windows
Header	mmdeviceapi.h

See also

ActivateAudioInterfaceAsync

Core Audio Interfaces

IActivateAudioInterfaceAsyncOperation

IActivateAudioInterfaceCompletionHandler::ActivateCompleted method

1/11/2020 • 2 minutes to read • Edit Online

Indicates that activation of a WASAPI interface is complete and results are available.

Syntax

```
HRESULT ActivateCompleted(
  IActivateAudioInterfaceAsyncOperation *activateOperation
);
```

Parameters

`activateOperation`

An interface representing the asynchronous operation of activating the requested **WASAPI** interface

Return value

The function returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

RETURN CODE	DESCRIPTION
S_OK	The function succeeded.

Remarks

An application implements this method if it calls the ActivateAudioInterfaceAsync function. When Windows calls this method, the results of the activation are available. The application can then retrieve the results by calling the GetActivateResult method of the IActivateAudioInterfaceAsyncOperation interface, passed through the *activateOperation* parameter.

Requirements

Minimum supported client	Windows 8 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps | UWP apps]
Target Platform	Windows
Header	mmdeviceapi.h

See also

ActivateAudioInterfaceAsync

IActivateAudioInterfaceCompletionHandler

IMMDevice interface

1/23/2020 • 2 minutes to read • Edit Online

The **IMMDevice** interface encapsulates the generic features of a multimedia device resource. In the current implementation of the MMDevice API, the only type of device resource that an **IMMDevice** interface can represent is an audio endpoint device.

A client can obtain an **IMMDevice** interface from one of the following methods:

- IMMDeviceCollection::Item
- IMMDeviceEnumerator::GetDefaultAudioEndpoint
- IMMDeviceEnumerator::GetDevice

For more information, see IMMDeviceCollection Interface.

After obtaining the **IMMDevice** interface of an audio endpoint device, a client can obtain an interface that encapsulates the endpoint-specific features of the device by calling the **IMMDevice::QueryInterface** method with parameter *iid* set to **REFIID** IID_IMMEndpoint. For more information, see IMMEndpoint Interface.

For code examples that use the **IMMDevice** interface, see the following topics:

- Device Properties
- Rendering a Stream
- Device Roles for Legacy Windows Multimedia Applications

Inheritance

The **IMMDevice** interface inherits from the IUnknown interface. **IMMDevice** also has these types of members:

- Methods

Methods

The **IMMDevice** interface has these methods.

METHOD	DESCRIPTION
IMMDevice::Activate	The Activate method creates a COM object with the specified interface.
IMMDevice::GetId	The GetId method retrieves an endpoint ID string that identifies the audio endpoint device.
IMMDevice::GetState	The GetState method retrieves the current device state.
IMMDevice::OpenPropertyStore	The OpenPropertyStore method retrieves an interface to the device's property store.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

Core Audio Interfaces

IMMDeviceCollection Interface

IMMDeviceCollection::Item

IMMDeviceEnumerator::GetDefaultAudioEndpoint

IMMDeviceEnumerator::GetDevice

IMMEndpoint Interface

MMDevice API

IMMDevice::Activate method

1/11/2020 • 5 minutes to read • Edit Online

The **Activate** method creates a COM object with the specified interface.

Syntax

```
HRESULT Activate(
  REFIID      iid,
  DWORD       dwClsCtx,
  PROPVARIANT *pActivationParams,
  void        **ppInterface
);
```

Parameters

`iid`

The interface identifier. This parameter is a reference to a GUID that identifies the interface that the caller requests be activated. The caller will use this interface to communicate with the COM object. Set this parameter to one of the following interface identifiers:

IID_IAudioClient

IID_IAudioEndpointVolume

IID_IAudioMeterInformation

IID_IAudioSessionManager

IID_IAudioSessionManager2

IID_IBaseFilter

IID_IDeviceTopology

IID_IDirectSound

IID_IDirectSound8

IID_IDirectSoundCapture

IID_IDirectSoundCapture8

IID_IMFTrustedOutput

IID_ISpatialAudioClient

IID_ISpatialAudioMetadataClient

For more information, see Remarks.

`dwClsCtx`

The execution context in which the code that manages the newly created object will run. The caller can restrict the context by setting this parameter to the bitwise **OR** of one or more **CLSCTX** enumeration values. Alternatively, the

client can avoid imposing any context restrictions by specifying CLSCTX_ALL. For more information about **CLSCTX**, see the Windows SDK documentation.

`pActivationParams`

Set to **NULL** to activate an IAudioClient, IAudioEndpointVolume, IAudioMeterInformation, IAudioSessionManager, or IDeviceTopology interface on an audio endpoint device. When activating an **IBaseFilter**, **IDirectSound**, **IDirectSound8**, **IDirectSoundCapture**, or **IDirectSoundCapture8** interface on the device, the caller can specify a pointer to a **PROPVARIANT** structure that contains stream-initialization information. For more information, see Remarks.

`ppInterface`

Pointer to a pointer variable into which the method writes the address of the interface specified by parameter *iid*. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **Activate** call fails, **ppInterface* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_NOINTERFACE	The object does not support the requested interface type.
E_POINTER	Parameter *ppInterface* is **NULL**.
E_INVALIDARG	The *pActivationParams* parameter must be **NULL** for the specified interface; or *pActivationParams* points to invalid data.
E_OUTOFMEMORY	Out of memory.

AU **DCL** **NT_** **E_D** **EVI** **CE_I** **NV** **ALI** **DAT** **ED**	The user has removed either the audio endpoint device or the adapter device that the endpoint device connects to.

Remarks

This method creates a COM object with an interface that is specified by the *iid* parameter. The method is similar to the Windows **CoCreateInstance** function, except that the caller does not supply a CLSID as a parameter. For more information about **CoCreateInstance**, see the Windows SDK documentation.

A client can call the **Activate** method of the **IMMDevice** interface for a particular audio endpoint device to obtain a counted reference to an interface on that device. The method can activate the following interfaces:

- IAudioClient
- IAudioEndpointVolume
- IAudioMeterInformation
- IAudioSessionManager
- IAudioSessionManager2
- IBaseFilter
- IDeviceTopology
- IDirectSound
- IDirectSound8
- IDirectSoundCapture
- IDirectSoundCapture8
- IMFTrustedOutput

To obtain the interface ID for an interface, use the **__uuidof** operator. For example, the interface ID of **IAudioCaptureClient** is defined as follows:

```
const IID IID_IAudioClient  __uuidof(IAudioCaptureClient)
```

For information about the **__uuidof** operator, see the Windows SDK documentation. For information about **IBaseFilter**, **IDirectSound**, **IDirectSound8**, **IDirectSoundCapture**, **IDirectSoundCapture8**, and **IMFTrustedOutput** see the Windows SDK documentation.

The *pActivationParams* parameter should be **NULL** for an **Activate** call to create an **IAudioClient**, **IAudioEndpointVolume**, **IAudioMeterInformation**, **IAudioSessionManager**, or **IDeviceTopology** interface for an audio endpoint device.

For an **Activate** call to create an **IBaseFilter**, **IDirectSound**, **IDirectSound8**, **IDirectSoundCapture**, or **IDirectSoundCapture8** interface, the caller can, as an option, specify a non-**NULL** value for *pActivationParams*. In this case, *pActivationParams* points to a **PROPVARIANT** structure that contains stream-initialization information. Set the **vt** member of the structure to VT_BLOB. Set the **blob.pBlobData** member to point to a DIRECTX_AUDIO_ACTIVATION_PARAMS structure that contains an audio session GUID and stream-initialization flags. Set the **blob.cbSize** member to **sizeof**(**DIRECTX_AUDIO_ACTIVATION_PARAMS**). For a code example,

see Device Roles for DirectShow Applications. For more information about **PROPVARIANT**, see the Windows SDK documentation.

An **IBaseFilter**, **IDirectSound**, **IDirectSound8**, **IDirectSoundCapture**, or **IDirectSoundCapture8** interface instance that is created by the **Activate** method encapsulates a stream on the audio endpoint device. During the **Activate** call, the DirectSound system module creates the stream by calling the IAudioClient::Initialize method. If *pActivationParams* is non-**NULL**, DirectSound supplies the audio session GUID and stream-initialization flags from the **DIRECTX_AUDIO_ACTIVATION_PARAMS** structure as input parameters to the **Initialize** call. If *pActivationParams* is **NULL**, DirectSound sets the **Initialize** method's *AudioSessionGuid* and *StreamFlags* parameters to their respective default values, **NULL** and 0. These values instruct the method to assign the stream to the process-specific session that is identified by the session GUID value GUID_NULL.

Activate can activate an **IDirectSound** or **IDirectSound8** interface only on a rendering endpoint device. It can activate an **IDirectSoundCapture** or **IDirectSoundCapture8** interface only on a capture endpoint device. An **Activate** call to activate an **IDirectSound** or **IDirectSoundCapture8** interface on a capture device or an **IDirectSoundCapture** or **IDirectSoundCapture8** interface on a rendering device fails and returns error code E_NOINTERFACE.

In Windows 7, a client can call **IMMDevice::Activate** and specify, **IID_IMFTrustedOutput**, to create an output trust authorities (OTA) object and retrieve a pointer to the object's IMFTrustedOutput interface. OTAs can operate inside or outside the Media Foundation's protected media path (PMP) and send content outside the Media Foundation pipeline. If the caller is outside PMP, then the OTA may not operate in the PMP, and the protection settings are less robust. For information about using protected objects for audio and example code, see Protected User Mode Audio (PUMA).

For general information about protected objects and IMFTrustedOutput, see "Protected Media Path" in Media Foundation documentation.

> **Note** When using the ISpatialAudioClient interfaces on an Xbox One Development Kit (XDK) title, you must first call **EnableSpatialAudio** before calling IMMDeviceEnumerator::EnumAudioEndpoints or IMMDeviceEnumerator::GetDefaultAudioEndpoint. Failure to do so will result in an E_NOINTERFACE error being returned from the call to Activate. **EnableSpatialAudio** is only available for XDK titles, and does not need to be called for Universal Windows Platform apps running on Xbox One, nor for any non-Xbox One devices.

For code examples that call the **Activate** method, see the following topics:

- Rendering a Stream
- Device Topologies
- Using the IKsControl Interface to Access Audio Properties
- Audio Events for Legacy Audio Applications
- Render Spatial Sound Using Spatial Audio Objects

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IAudioClient Interface

IAudioEndpointVolume Interface

IAudioMeterInformation Interface

IAudioSessionManager Interface

IDeviceTopology Interface

IMMDevice Interface

IMMDevice::GetId method

1/11/2020 • 2 minutes to read • Edit Online

The **GetId** method retrieves an endpoint ID string that identifies the audio endpoint device.

Syntax

```
HRESULT GetId(
  LPWSTR *ppstrId
);
```

Parameters

`ppstrId`

Pointer to a pointer variable into which the method writes the address of a null-terminated, wide-character string containing the endpoint device ID. The method allocates the storage for the string. The caller is responsible for freeing the storage, when it is no longer needed, by calling the **CoTaskMemFree** function. If the **GetId** call fails, **ppstrId is NULL*. For information about **CoTaskMemFree**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_OUTOFMEMORY	Out of memory.
E_POINTER	Parameter *pwstrId* is **NULL**.

Remarks

The endpoint ID string obtained from this method identifies the audio endpoint device that is represented by the **IMMDevice** interface instance. A client can use the endpoint ID string to create an instance of the audio endpoint device at a later time or in a different process by calling the IMMDeviceEnumerator::GetDevice method. Clients should treat the contents of the endpoint ID string as opaque. That is, clients should *not* attempt to parse the contents of the string to obtain information about the device. The reason is that the string format is undefined and might change from one implementation of the MMDevice API system module to the next.

For code examples that call the **GetId** method, see the following topics:

- Device Properties
- Device Roles for Legacy Windows Multimedia Applications

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDevice Interface

IMMDeviceEnumerator::GetDevice

IMMDevice::GetState method

1/11/2020 • 2 minutes to read • Edit Online

The **GetState** method retrieves the current device state.

Syntax

```
HRESULT GetState(
  DWORD *pdwState
);
```

Parameters

`pdwState`

Pointer to a **DWORD** variable into which the method writes the current state of the device. The device-state value is one of the following DEVICE_STATE_XXX constants:

DEVICE_STATE_ACTIVE

DEVICE_STATE_DISABLED

DEVICE_STATE_NOTPRESENT

DEVICE_STATE_UNPLUGGED

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pdwState* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDevice Interface

IMMDevice::OpenPropertyStore method

1/11/2020 • 2 minutes to read • Edit Online

The **OpenPropertyStore** method retrieves an interface to the device's property store.

Syntax

```
HRESULT OpenPropertyStore(
  DWORD          stgmAccess,
  IPropertyStore **ppProperties
);
```

Parameters

`stgmAccess`

The storage-access mode. This parameter specifies whether to open the property store in read mode, write mode, or read/write mode. Set this parameter to one of the following STGM constants:

STGM_READ

STGM_WRITE

STGM_READWRITE

The method permits a client running as an administrator to open a store for read-only, write-only, or read/write access. A client that is not running as an administrator is restricted to read-only access. For more information about STGM constants, see the Windows SDK documentation.

`ppProperties`

Pointer to a pointer variable into which the method writes the address of the **IPropertyStore** interface of the device's property store. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **OpenPropertyStore** call fails, **ppProperties* is **NULL**. For more information about **IPropertyStore**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	Parameter *stgmAccess* is not a valid access mode.

E_POINTER	Parameter *ppProperties* is **NULL**.
E_OUTOFMEMORY	Out of memory.

Remarks

In general, the properties in the device's property store are read-only for clients that do not perform administrative, system, or service functions.

For code examples that call the **OpenPropertyStore** method, see the following topics:

- Device Properties
- Device Roles for DirectSound Applications

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDevice Interface

IMMDeviceCollection interface

1/23/2020 • 2 minutes to read • Edit Online

The **IMMDeviceCollection** interface represents a collection of multimedia device resources. In the current implementation, the only device resources that the MMDevice API can create collections of are audio endpoint devices.

A client can obtain a reference to an **IMMDeviceCollection** interface instance by calling the IMMDeviceEnumerator::EnumAudioEndpoints method. This method creates a collection of endpoint objects, each of which represents an audio endpoint device in the system. Each endpoint object in the collection supports the IMMDevice and IMMEndpoint interfaces. For more information, see IMMDeviceEnumerator Interface.

For a code example that uses the **IMMDeviceCollection** interface, see Device Properties.

Inheritance

The **IMMDeviceCollection** interface inherits from the IUnknown interface. **IMMDeviceCollection** also has these types of members:

- Methods

Methods

The **IMMDeviceCollection** interface has these methods.

METHOD	DESCRIPTION
IMMDeviceCollection::GetCount	The GetCount method retrieves a count of the devices in the device collection.
IMMDeviceCollection::Item	The Item method retrieves a pointer to the specified item in the device collection.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

Core Audio Interfaces

IMMDevice Interface

IMMDeviceEnumerator Interface

IMMDeviceEnumerator::EnumAudioEndpoints

IMMEndpoint Interface

MMDevice API

IMMDeviceCollection::GetCount method

1/11/2020 • 2 minutes to read • Edit Online

The **GetCount** method retrieves a count of the devices in the device collection.

Syntax

```
HRESULT GetCount(
  UINT *pcDevices
);
```

Parameters

`pcDevices`

Pointer to a **UINT** variable into which the method writes the number of devices in the device collection.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pcDevices* is **NULL**.

Remarks

For a code example that calls the **GetCount** method, see Device Properties.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDeviceCollection Interface

IMMDeviceCollection::Item method

1/11/2020 • 2 minutes to read • Edit Online

The **Item** method retrieves a pointer to the specified item in the device collection.

Syntax

```
HRESULT Item(
  UINT      nDevice,
  IMMDevice **ppDevice
);
```

Parameters

`nDevice`

The device number. If the collection contains *n* devices, the devices are numbered 0 to *n*– 1.

`ppDevice`

Pointer to a pointer variable into which the method writes the address of the IMMDevice interface of the specified item in the device collection. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **Item** call fails, **ppDevice* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *ppDevice* is **NULL**.
E_INVALIDARG	Parameter *nDevice* is not a valid device number.

Remarks

This method retrieves a pointer to the **IMMDevice** interface of the specified item in the device collection. Each item in the collection is an endpoint object that represents an audio endpoint device. The caller selects a device from the device collection by specifying the device number. For a collection of *n* devices, valid device numbers range from 0 to *n*– 1. To obtain a count of the devices in a collection, call the IMMDeviceCollection::GetCount method.

For a code example that calls the **Item** method, see Device Properties.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDevice Interface

IMMDeviceCollection Interface

IMMDeviceCollection::GetCount

IMMDeviceEnumerator interface

1/23/2020 • 2 minutes to read • Edit Online

The **IMMDeviceEnumerator** interface provides methods for enumerating multimedia device resources. In the current implementation of the MMDevice API, the only device resources that this interface can enumerate are audio endpoint devices. A client obtains a reference to an **IMMDeviceEnumerator** interface by calling the **CoCreateInstance** function, as described previously (see MMDevice API).

The device resources enumerated by the methods in the **IMMDeviceEnumerator** interface are represented as collections of objects with IMMDevice interfaces. A collection has an IMMDeviceCollection interface. The IMMDeviceEnumerator::EnumAudioEndpoints method creates a device collection.

To obtain a pointer to the **IMMDevice** interface of an item in a device collection, the client calls the IMMDeviceCollection::Item method.

For code examples that use the **IMMDeviceEnumerator** interface, see the following topics:

- Device Properties
- Rendering a Stream

Inheritance

The **IMMDeviceEnumerator** interface inherits from the IUnknown interface. **IMMDeviceEnumerator** also has these types of members:

- Methods

Methods

The **IMMDeviceEnumerator** interface has these methods.

METHOD	DESCRIPTION
IMMDeviceEnumerator::EnumAudioEndpoints	The EnumAudioEndpoints method generates a collection of audio endpoint devices that meet the specified criteria.
IMMDeviceEnumerator::GetDefaultAudioEndpoint	The GetDefaultAudioEndpoint method retrieves the default audio endpoint for the specified data-flow direction and role.
IMMDeviceEnumerator::GetDevice	The GetDevice method retrieves an audio endpoint device that is identified by an endpoint ID string.
IMMDeviceEnumerator::RegisterEndpointNotificationCallback	The RegisterEndpointNotificationCallback method registers a client's notification callback interface.
IMMDeviceEnumerator::UnregisterEndpointNotificationCallback	The UnregisterEndpointNotificationCallback method deletes the registration of a notification interface that the client registered in a previous call to the IMMDeviceEnumerator::RegisterEndpointNotificationCallback method.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

Core Audio Interfaces

IMMDevice Interface

IMMDeviceCollection Interface

IMMDeviceCollection::Item

IMMDeviceEnumerator::EnumAudioEndpoints

MMDevice API

IMMDeviceEnumerator::EnumAudioEndpoints method

1/11/2020 • 2 minutes to read • Edit Online

The **EnumAudioEndpoints** method generates a collection of audio endpoint devices that meet the specified criteria.

Syntax

```
HRESULT EnumAudioEndpoints(
  EDataFlow           dataFlow,
  DWORD               dwStateMask,
  IMMDeviceCollection **ppDevices
);
```

Parameters

`dataFlow`

The data-flow direction for the endpoint devices in the collection. The caller should set this parameter to one of the following EDataFlow enumeration values:

eRender

eCapture

eAll

If the caller specifies eAll, the method includes both rendering and capture endpoints in the collection.

`dwStateMask`

The state or states of the endpoints that are to be included in the collection. The caller should set this parameter to the bitwise OR of one or more of the following DEVICE_STATE_XXX constants:

DEVICE_STATE_ACTIVE

DEVICE_STATE_DISABLED

DEVICE_STATE_NOTPRESENT

DEVICE_STATE_UNPLUGGED

For example, if the caller sets the *dwStateMask* parameter to DEVICE_STATE_ACTIVE | DEVICE_STATE_UNPLUGGED, the method includes endpoints that are either active or unplugged from their jacks, but excludes endpoints that are on audio adapters that have been disabled or are not present. To include all endpoints, regardless of state, set *dwStateMask* = DEVICE_STATEMASK_ALL.

`ppDevices`

Pointer to a pointer variable into which the method writes the address of the IMMDeviceCollection interface of the device-collection object. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **EnumAudioEndpoints** call fails, **ppDevices* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *ppDevices* is **NULL**.
E_INVALIDARG	Parameter *dataFlow* or *dwStateMask* is out of range.
E_OUTOFMEMORY	Out of memory.

Remarks

For example, the following call enumerates all audio-rendering endpoint devices that are currently active (present and not disabled):

```
hr = pDevEnum->EnumAudioEndpoints(
                 eRender, DEVICE_STATE_ACTIVE,
                 &pEndpoints);
```

In the preceding code fragment, variable *hr* is of type **HRESULT**, *pDevEnum* is a pointer to an **IMMDeviceEnumerator** interface, and *pEndpoints* is a pointer to an **IMMDeviceCollection** interface.

Examples

For a code example that calls the **EnumAudioEndpoints** method, see Device Properties.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDeviceCollection Interface

IMMDeviceEnumerator Interface

IMMDeviceEnumerator::GetDefaultAudioEndpoint method

1/11/2020 • 3 minutes to read • Edit Online

The **GetDefaultAudioEndpoint** method retrieves the default audio endpoint for the specified data-flow direction and role.

Syntax

```
HRESULT GetDefaultAudioEndpoint(
  EDataFlow dataFlow,
  ERole     role,
  IMMDevice **ppEndpoint
);
```

Parameters

`dataFlow`

The data-flow direction for the endpoint device. The caller should set this parameter to one of the following two EDataFlow enumeration values:

eRender

eCapture

The data-flow direction for a rendering device is eRender. The data-flow direction for a capture device is eCapture.

`role`

The role of the endpoint device. The caller should set this parameter to one of the following ERole enumeration values:

eConsole

eMultimedia

eCommunications

For more information, see Remarks.

`ppEndpoint`

Pointer to a pointer variable into which the method writes the address of the IMMDevice interface of the endpoint object for the default audio endpoint device. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetDefaultAudioEndpoint** call fails, **ppDevice* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *ppDevice* is **NULL**.
E_INVALIDARG	Parameter *dataFlow* or *role* is out of range.
E_NOTFOUND	No device is available.
E_OUTOFMEMORY	Out of memory.

Remarks

Note

In Windows Vista, the MMDevice API supports device roles but the system-supplied user interface programs do not. The user interface in Windows Vista enables the user to select a default audio device for rendering and a default audio device for capture. When the user changes the default rendering or capture device, the system assigns all three device roles (eConsole, eMultimedia, and eCommunications) to that device. Thus, **GetDefaultAudioEndpoint** always selects the default rendering or capture device, regardless of which role is indicated by the *role* parameter. In a future version of Windows, the user interface might enable the user to assign individual roles to different devices. In that case, the selection of a rendering or capture device by **GetDefaultAudioEndpoint** might depend on the *role* parameter. Thus, the behavior of an audio application developed to run in Windows Vista might change when run in a future version of Windows. For more information, see Device Roles in Windows Vista.

This method retrieves the default endpoint device for the specified data-flow direction (rendering or capture) and role. For example, a client can get the default console playback device by making the following call:

```
hr = pDevEnum->GetDefaultAudioEndpoint(
                  eRender, eConsole, &pDeviceOut);
```

In the preceding code fragment, variable *hr* is of type **HRESULT**, *pDevEnum* is a pointer to an **IMMDeviceEnumerator** interface, and *pDeviceOut* is a pointer to an **IMMDevice** interface.

A Windows system might contain some combination of audio endpoint devices such as desktop speakers, high-fidelity headphones, desktop microphones, headsets with speaker and microphones, and high-fidelity multichannel

speakers. The user can assign appropriate roles to the devices. For example, an application that manages voice communications streams can call **GetDefaultAudioEndpoint** to identify the designated rendering and capture devices for that role.

If only a single rendering or capture device is available, the system always assigns all three rendering or capture roles to that device. If the method fails to find a rendering or capture device for the specified role, this means that no rendering or capture device is available at all. If no device is available, the method sets **ppEndpoint* = **NULL** and returns ERROR_NOT_FOUND.

For code examples that call the **GetDefaultAudioEndpoint** method, see the following topics:

- Rendering a Stream
- Audio Events for Legacy Audio Applications

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDevice Interface

IMMDeviceEnumerator Interface

IMMDeviceEnumerator::GetDevice method

1/11/2020 • 2 minutes to read • Edit Online

The **GetDevice** method retrieves an audio endpoint device that is identified by an endpoint ID string.

Syntax

```
HRESULT GetDevice(
  LPCWSTR   pwstrId,
  IMMDevice **ppDevice
);
```

Parameters

`pwstrId`

Pointer to a string containing the endpoint ID. The caller typically obtains this string from the IMMDevice::GetId method or from one of the methods in the IMMNotificationClient interface.

`ppDevice`

Pointer to a pointer variable into which the method writes the address of the IMMDevice interface for the specified device. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's **Release** method. If the **GetDevice** call fails, **ppDevice* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pwstrId* or *ppDevice* is **NULL**.
E_NOTFOUND	The device ID does not identify an audio device that is in this system.
E_OUTOFMEMORY	Out of memory.

Remarks

If two programs are running in two different processes and both need to access the same audio endpoint device, one program cannot simply pass the device's **IMMDevice** interface to the other program. However, the programs can access the same device by following these steps:

1. The first program calls the **IMMDevice::GetId** method in the first process to obtain the endpoint ID string that identifies the device.
2. The first program passes the endpoint ID string across the process boundary to the second program.
3. To obtain a reference to the device's **IMMDevice** interface in the second process, the second program calls **GetDevice** with the endpoint ID string.

For more information about the **GetDevice** method, see the following topics:

- Endpoint ID Strings
- Audio Events for Legacy Audio Applications

For code examples that use the **GetDevice** method, see the following topics:

- Device Properties
- Device Events
- Using the IKsControl Interface to Access Audio Properties

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDevice Interface

IMMDevice::GetId

IMMDeviceEnumerator Interface

IMMNotificationClient Interface

IMMDeviceEnumerator::RegisterEndpointNotificationCallback method

1/11/2020 • 2 minutes to read • Edit Online

The **RegisterEndpointNotificationCallback** method registers a client's notification callback interface.

Syntax

```
HRESULT RegisterEndpointNotificationCallback(
  IMMNotificationClient *pClient
);
```

Parameters

`pClient`

Pointer to the IMMNotificationClient interface that the client is registering for notification callbacks.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pNotify* is **NULL**.
E_OUTOFMEMORY	Out of memory.

Remarks

This method registers an IMMNotificationClient interface to be called by the system when the roles, state, existence, or properties of an endpoint device change. The caller implements the IMMNotificationClient interface.

When notifications are no longer needed, the client can call the IMMDeviceEnumerator::UnregisterEndpointNotificationCallback method to terminate the notifications.

The client must ensure that the IMMNotificationClient object is not released after the **RegisterEndpointNotificationCallback** call and before calling UnregisterEndpointNotificationCallback. These methods do not call the client's **IMMNotificationClient::AddRef** and **IMMNotificationClient::Release** implementations. The client is responsible for maintaining the reference count of the **IMMNotificationClient** object. The client must increment the count if the **RegisterEndpointNotificationCallback** call succeeds and release the final reference only after calling **UnregisterEndpointNotificationCallback** or implement some other mechanism to ensure that the object is not deleted before **UnregisterEndpointNotificationCallback** is called. Otherwise, the application leaks the resources held by the **IMMNotificationClient** and any other object that is implemented in the same container.

For more information about the **AddRef** and **Release** methods, see the discussion of the **IUnknown** interface in the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDeviceEnumerator Interface

IMMDeviceEnumerator::UnregisterEndpointNotificationCallback

IMMNotificationClient Interface

IMMDeviceEnumerator::UnregisterEndpointNotificationCallback method

1/11/2020 • 2 minutes to read • Edit Online

The **UnregisterEndpointNotificationCallback** method deletes the registration of a notification interface that the client registered in a previous call to the IMMDeviceEnumerator::RegisterEndpointNotificationCallback method.

Syntax

```
HRESULT UnregisterEndpointNotificationCallback(
  IMMNotificationClient *pClient
);
```

Parameters

`pClient`

Pointer to the client's IMMNotificationClient interface. The client passed this same interface pointer to the device enumerator in a previous call to the IMMDeviceEnumerator::RegisterEndpointNotificationCallback method. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *pNotify* is **NULL**.
E_NOTFOUND	The specified notification interface was not found.

Remarks

The client must ensure that the IMMNotificationClient object is not released after the RegisterEndpointNotificationCallback call and before calling **UnregisterEndpointNotificationCallback**. These methods do not call the client's **IMMNotificationClient::AddRef** and **IMMNotificationClient::Release** implementations. The client is responsible for maintaining the reference count of the **IMMNotificationClient** object. The client must increment the count if the **RegisterEndpointNotificationCallback** call succeeds and release the final reference only after calling **UnregisterEndpointNotificationCallback** or implement some other mechanism to ensure that the object is not deleted before **UnregisterEndpointNotificationCallback** is called. Otherwise, the application leaks the resources held by the **IMMNotificationClient** and any other object that is implemented in the same container.

For more information about the **AddRef** and **Release** methods, see the discussion of the **IUnknown** interface in the Windows SDK documentation.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMDeviceEnumerator Interface

IMMDeviceEnumerator::RegisterEndpointNotificationCallback

IMMNotificationClient Interface

IMMEndpoint interface

1/23/2020 • 2 minutes to read • Edit Online

The **IMMEndpoint** interface represents an audio endpoint device. A client obtains a reference to an **IMMEndpoint** interface instance by following these steps:

1. By using one of the techniques described in IMMDevice Interface, obtain a reference to the **IMMDevice** interface of an audio endpoint device.
2. Call the **IMMDevice::QueryInterface** method with parameter *iid* set to **REFIID** IID_IMMEndpoint.

Inheritance

The **IMMEndpoint** interface inherits from the IUnknown interface. **IMMEndpoint** also has these types of members:

- Methods

Methods

The **IMMEndpoint** interface has these methods.

METHOD	DESCRIPTION
IMMEndpoint::GetDataFlow	The GetDataFlow method indicates whether the audio endpoint device is a rendering device or a capture device.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

Core Audio Interfaces

IMMDevice Interface

MMDevice API

IMMEndpoint::GetDataFlow method

1/11/2020 • 2 minutes to read • Edit Online

The **GetDataFlow** method indicates whether the audio endpoint device is a rendering device or a capture device.

Syntax

```
HRESULT GetDataFlow(
  EDataFlow *pDataFlow
);
```

Parameters

`pDataFlow`

Pointer to a variable into which the method writes the data-flow direction of the endpoint device. The direction is indicated by one of the following EDataFlow enumeration constants:

eRender

eCapture

The data-flow direction for a rendering device is eRender. The data-flow direction for a capture device is eCapture.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *ppDataFlow* is **NULL**.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMNotificationClient interface

1/23/2020 • 2 minutes to read • Edit Online

The **IMMNotificationClient** interface provides notifications when an audio endpoint device is added or removed, when the state or properties of an endpoint device change, or when there is a change in the default role assigned to an endpoint device. Unlike the other interfaces in this section, which are implemented by the MMDevice API system component, an MMDevice API client implements the **IMMNotificationClient** interface. To receive notifications, the client passes a pointer to its **IMMNotificationClient** interface instance as a parameter to the IMMDeviceEnumerator::RegisterEndpointNotificationCallback method.

After registering its **IMMNotificationClient** interface, the client receives event notifications in the form of callbacks through the methods of the interface.

Each method in the **IMMNotificationClient** interface receives, as one of its input parameters, an endpoint ID string that identifies the audio endpoint device that is the subject of the notification. The string uniquely identifies the device with respect to all of the other audio endpoint devices in the system. The methods in the **IMMNotificationClient** interface implementation should treat this string as opaque. That is, none of the methods should attempt to parse the contents of the string to obtain information about the device. The reason is that the string format is undefined and might change from one implementation of the MMDevice API system module to the next.

A client can use the endpoint ID string that it receives as an input parameter in a call to an **IMMNotificationClient** method in two ways:

- The client can create an instance of the device that the endpoint ID string identifies. The client does this by calling the IMMDeviceEnumerator::GetDevice method and supplying the endpoint ID string as an input parameter.
- The client can compare the endpoint ID string with the endpoint ID string of an existing device instance. To obtain the second endpoint ID string, the client calls the IMMDevice::GetId method of the device instance. If the two strings match, they identify the same device.

In implementing the **IMMNotificationClient** interface, the client should observe these rules to avoid deadlocks and undefined behavior:

- The methods of the interface must be nonblocking. The client should never wait on a synchronization object during an event callback.
- To avoid dead locks, the client should never call IMMDeviceEnumerator::RegisterEndpointNotificationCallback or IMMDeviceEnumerator::UnregisterEndpointNotificationCallback in its implementation of **IMMNotificationClient** methods.
- The client should never release the final reference on an MMDevice API object during an event callback.

For a code example that implements the **IMMNotificationClient** interface, see Device Events.

Inheritance

The **IMMNotificationClient** interface inherits from the IUnknown interface. **IMMNotificationClient** also has these types of members:

- Methods

Methods

The **IMMNotificationClient** interface has these methods.

METHOD	DESCRIPTION
IMMNotificationClient::OnDefaultDeviceChanged	The OnDefaultDeviceChanged method notifies the client that the default audio endpoint device for a particular device role has changed.
IMMNotificationClient::OnDeviceAdded	The OnDeviceAdded method indicates that a new audio endpoint device has been added.
IMMNotificationClient::OnDeviceRemoved	The OnDeviceRemoved method indicates that an audio endpoint device has been removed.
IMMNotificationClient::OnDeviceStateChanged	The OnDeviceStateChanged method indicates that the state of an audio endpoint device has changed.
IMMNotificationClient::OnPropertyValueChanged	The OnPropertyValueChanged method indicates that the value of a property belonging to an audio endpoint device has changed.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

Core Audio Interfaces

IMMDevice::GetId

IMMDeviceEnumerator::GetDevice

IMMDeviceEnumerator::RegisterEndpointNotificationCallback

IMMDeviceEnumerator::UnregisterEndpointNotificationCallback

MMDevice API

IMMNotificationClient::OnDefaultDeviceChanged method

1/11/2020 • 2 minutes to read • Edit Online

The **OnDefaultDeviceChanged** method notifies the client that the default audio endpoint device for a particular device role has changed.

Syntax

```
HRESULT OnDefaultDeviceChanged(
  EDataFlow flow,
  ERole     role,
  LPCWSTR   pwstrDefaultDeviceId
);
```

Parameters

`flow`

The data-flow direction of the endpoint device. This parameter is set to one of the following EDataFlow enumeration values:

eRender

eCapture

The data-flow direction for a rendering device is eRender. The data-flow direction for a capture device is eCapture.

`role`

The device role of the audio endpoint device. This parameter is set to one of the following ERole enumeration values:

eConsole

eMultimedia

eCommunications

`pwstrDefaultDeviceId`

Pointer to the endpoint ID string that identifies the audio endpoint device. This parameter points to a null-terminated, wide-character string containing the endpoint ID. The string remains valid for the duration of the call. If the user has removed or disabled the default device for a particular role, and no other device is available to assume that role, then *pwstrDefaultDevice* is **NULL**.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

The three input parameters specify the data-flow direction, device role, and endpoint ID string of the new default

audio endpoint device.

In Windows Vista, the MMDevice API supports device roles but the system-supplied user interface programs do not. The user interface in Windows Vista enables the user to select a default audio device for rendering and a default audio device for capture. When the user changes the default rendering or capture device, the system assigns all three device roles (eConsole, eMultimedia, and eCommunications) to the new device. Thus, when the user changes the default rendering or capture device, the system calls the client's **OnDefaultDeviceChanged** method three times—once for each of the three device roles.

In a future version of Windows, the user interface might enable the user to assign individual roles to different devices. In that case, if the user changes the assignment of only one or two device roles to a new rendering or capture device, the system will call the client's **OnDefaultDeviceChanged** method only once or twice (that is, one call per changed role). Depending on how the **OnDefaultDeviceChanged** method responds to role changes, the behavior of an audio application developed to run in Windows Vista might change when run in a future version of Windows. For more information, see Device Roles in Windows Vista.

For a code example that implements the **OnDefaultDeviceChanged** method, see Device Events.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMNotificationClient Interface

IMMNotificationClient::OnDeviceAdded method

1/11/2020 • 2 minutes to read • Edit Online

The **OnDeviceAdded** method indicates that a new audio endpoint device has been added.

Syntax

```
HRESULT OnDeviceAdded(
  LPCWSTR pwstrDeviceId
);
```

Parameters

`pwstrDeviceId`

Pointer to the endpoint ID string that identifies the audio endpoint device. This parameter points to a null-terminated, wide-character string containing the endpoint ID. The string remains valid for the duration of the call.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

For a code example that implements the **OnDeviceAdded** method, see Device Events.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMNotificationClient Interface

IMMNotificationClient::OnDeviceRemoved method

1/11/2020 • 2 minutes to read • Edit Online

The **OnDeviceRemoved** method indicates that an audio endpoint device has been removed.

Syntax

```
HRESULT OnDeviceRemoved(
  LPCWSTR pwstrDeviceId
);
```

Parameters

`pwstrDeviceId`

Pointer to the endpoint ID string that identifies the audio endpoint device. This parameter points to a null-terminated, wide-character string containing the endpoint ID. The string remains valid for the duration of the call.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

For a code example that implements the **OnDeviceRemoved** method, see Device Events.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMNotificationClient Interface

IMMNotificationClient::OnDeviceStateChanged method

1/11/2020 • 2 minutes to read • Edit Online

The **OnDeviceStateChanged** method indicates that the state of an audio endpoint device has changed.

Syntax

```
HRESULT OnDeviceStateChanged(
  LPCWSTR pwstrDeviceId,
  DWORD   dwNewState
);
```

Parameters

`pwstrDeviceId`

Pointer to the endpoint ID string that identifies the audio endpoint device. This parameter points to a null-terminated, wide-character string containing the endpoint ID. The string remains valid for the duration of the call.

`dwNewState`

Specifies the new state of the endpoint device. The value of this parameter is one of the following DEVICE_STATE_XXX constants:

DEVICE_STATE_ACTIVE

DEVICE_STATE_DISABLED

DEVICE_STATE_NOTPRESENT

DEVICE_STATE_UNPLUGGED

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

For a code example that implements the **OnDeviceStateChanged** method, see Device Events.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Header	mmdeviceapi.h

See also

IMMNotificationClient Interface

IMMNotificationClient::OnPropertyValueChanged method

1/11/2020 • 2 minutes to read • Edit Online

The **OnPropertyValueChanged** method indicates that the value of a property belonging to an audio endpoint device has changed.

Syntax

```
HRESULT OnPropertyValueChanged(
  LPCWSTR           pwstrDeviceId,
  const PROPERTYKEY key
);
```

Parameters

`pwstrDeviceId`

Pointer to the endpoint ID string that identifies the audio endpoint device. This parameter points to a null-terminated, wide-character string that contains the endpoint ID. The string remains valid for the duration of the call.

`key`

A PROPERTYKEY structure that specifies the property. The structure contains the property-set GUID and an index identifying a property within the set. The structure is passed by value. It remains valid for the duration of the call. For more information about **PROPERTYKEY**, see the Windows SDK documentation.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

A call to the IPropertyStore::SetValue method that successfully changes the value of a property of an audio endpoint device generates a call to **OnPropertyValueChanged**. For more information about **IPropertyStore::SetValue**, see the Windows SDK documentation.

A client can use the *key* parameter to retrieve the new property value. For a code example that uses a property key to retrieve a property value from the property store of an endpoint device, see Device Properties.

For a code example that implements the **OnPropertyValueChanged** method, see Device Events.

Requirements

Minimum supported client	Windows Vista [desktop apps only]

Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	mmdeviceapi.h

See also

IMMNotificationClient Interface

spatialaudioclient.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs spatialaudioclient.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IAudioFormatEnumerator	Provides a list of supported audio formats. The most preferred format is first in the list. Get a reference to this interface by calling ISpatialAudioClient::GetSupportedAudioObjectFormatEnumerator.
ISpatialAudioClient	The ISpatialAudioClient interface enables a client to create audio streams that emit audio from a position in 3D space.
ISpatialAudioObject	Represents an object that provides audio data to be rendered from a position in 3D space, relative to the user.
ISpatialAudioObjectBase	Base interface that represents an object that provides audio data to be rendered from a position in 3D space, relative to the user.
ISpatialAudioObjectRenderStream	Provides methods for controlling a spatial audio object render stream, including starting, stopping, and resetting the stream.
ISpatialAudioObjectRenderStreamBase	Base interface that provides methods for controlling a spatial audio object render stream, including starting, stopping, and resetting the stream.
ISpatialAudioObjectRenderStreamNotify	Provides notifications for spatial audio clients to respond to changes in the state of an ISpatialAudioObjectRenderStream.

Structures

TITLE	DESCRIPTION
SpatialAudioClientActivationParams	Represents optional activation parameters for a spatial audio render stream. Pass this structure to ActivateAudioInterfaceAsync when activating an ISpatialAudioClient interface.
SpatialAudioObjectRenderStreamActivationParams	Represents activation parameters for a spatial audio render stream. Pass this structure to ISpatialAudioClient::ActivateSpatialAudioStream when activating a stream.

Enumerations

TITLE	DESCRIPTION
AudioObjectType	Specifies the type of an ISpatialAudioObject.

AudioObjectType enumeration

1/11/2020 • 2 minutes to read • Edit Online

Specifies the type of an ISpatialAudioObject. A spatial audio object can be dynamic, meaning that it's spatial properties can change over time, or static, which means that its spatial properties are fixed. There are 17 audio channels to which a static spatial audio object can be assigned, each representing a real or virtualized speaker. The static channel values of the enumeration can be combined as a mask to assign a spatial audio object to multiple channels. All of the enumeration values except for **AudioObjectType_None** and **AudioObjectType_Dynamic** represent static channels.

Syntax

```
typedef enum AudioObjectType {
  AudioObjectType_None,
  AudioObjectType_Dynamic,
  AudioObjectType_FrontLeft,
  AudioObjectType_FrontRight,
  AudioObjectType_FrontCenter,
  AudioObjectType_LowFrequency,
  AudioObjectType_SideLeft,
  AudioObjectType_SideRight,
  AudioObjectType_BackLeft,
  AudioObjectType_BackRight,
  AudioObjectType_TopFrontLeft,
  AudioObjectType_TopFrontRight,
  AudioObjectType_TopBackLeft,
  AudioObjectType_TopBackRight,
  AudioObjectType_BottomFrontLeft,
  AudioObjectType_BottomFrontRight,
  AudioObjectType_BottomBackLeft,
  AudioObjectType_BottomBackRight,
  AudioObjectType_BackCenter
} ;
```

Constants

AudioObjectType_None	The spatial audio object is not spatialized.
AudioObjectType_Dynamic	The spatial audio object is dynamic. It's spatial properties can be changed over time.
AudioObjectType_FrontLeft	The spatial audio object is assigned the front left channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_FRONT_LEFT.
AudioObjectType_FrontRight	The spatial audio object is assigned the front right channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_FRONT_RIGHT.

AudioObjectType_FrontCenter	The spatial audio object is assigned the front center channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_FRONT_CENTER.
AudioObjectType_LowFrequency	The spatial audio object is assigned the low frequency channel. Because this channel is not spatialized, it does not count toward the system resource limits for spatialized audio objects. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_LOW_FREQUENCY.
AudioObjectType_SideLeft	The spatial audio object is assigned the side left channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_SIDE_LEFT.
AudioObjectType_SideRight	The spatial audio object is assigned the side right channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_SIDE_RIGHT.
AudioObjectType_BackLeft	The spatial audio object is assigned the back left channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_BACK_LEFT.
AudioObjectType_BackRight	The spatial audio object is assigned the back right channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_BACK_RIGHT.
AudioObjectType_TopFrontLeft	The spatial audio object is assigned the top front left channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_TOP_FRONT_LEFT.
AudioObjectType_TopFrontRight	The spatial audio object is assigned the top front right channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_TOP_FRONT_RIGHT.
AudioObjectType_TopBackLeft	The spatial audio object is assigned the top back left channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_TOP_BACK_LEFT.
AudioObjectType_TopBackRight	The spatial audio object is assigned the top back right channel. The equivalent channel mask of DirectShow's WAVEFORMATEXTENSIBLE enumeration is SPEAKER_TOP_BACK_RIGHT.
AudioObjectType_BottomFrontLeft	The spatial audio object is assigned the bottom front left channel.
AudioObjectType_BottomFrontRight	The spatial audio object is assigned the bottom front right channel.

AudioObjectType_BottomBackLeft	The spatial audio object is assigned the bottom back left channel.
AudioObjectType_BottomBackRight	The spatial audio object is assigned the bottom back right channel.
AudioObjectType_BackCenter	The spatial audio object is assigned the back center channel.

Requirements

Header	spatialaudioclient.h

IAudioFormatEnumerator interface

1/23/2020 • 2 minutes to read • Edit Online

Provides a list of supported audio formats. The most preferred format is first in the list. Get a reference to this interface by calling ISpatialAudioClient::GetSupportedAudioObjectFormatEnumerator.

Inheritance

The **IAudioFormatEnumerator** interface inherits from the IUnknown interface. **IAudioFormatEnumerator** also has these types of members:

- Methods

Methods

The **IAudioFormatEnumerator** interface has these methods.

METHOD	DESCRIPTION
IAudioFormatEnumerator::GetCount	Gets the number of supported audio formats in the list.
IAudioFormatEnumerator::GetFormat	Gets the format with the specified index in the list. The formats are listed in order of importance. The most preferable format is first in the list.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

IAudioFormatEnumerator::GetCount method

1/11/2020 • 2 minutes to read • Edit Online

Gets the number of supported audio formats in the list

Syntax

```
HRESULT GetCount(
  UINT32 *count
);
```

Parameters

`count`

The number of supported audio formats in the list.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

IAudioFormatEnumerator

IAudioFormatEnumerator::GetFormat method

1/11/2020 • 2 minutes to read • Edit Online

Gets the format with the specified index in the list. The formats are listed in order of importance. The most preferable format is first in the list.

Syntax

```
HRESULT GetFormat(
  UINT32       index,
  WAVEFORMATEX **format
);
```

Parameters

`index`

The index of the item in the list to retrieve.

`format`

Pointer to a pointer to a **WAVEFORMATEX** structure describing a supported audio format.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

IAudioFormatEnumerator

ISpatialAudioClient interface

1/23/2020 • 2 minutes to read • Edit Online

The **ISpatialAudioClient** interface enables a client to create audio streams that emit audio from a position in 3D space. This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioClient** interface inherits from the IUnknown interface. **ISpatialAudioClient** also has these types of members:

- Methods

Methods

The **ISpatialAudioClient** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioClient::ActivateSpatialAudioStream	Activates and initializes spatial audio stream using one of the spatial audio stream activation structures.
ISpatialAudioClient::GetMaxDynamicObjectCount	Gets the maximum number of dynamic audio objects for the spatial audio client.
ISpatialAudioClient::GetMaxFrameCount	Gets the maximum possible frame count per processing pass. This method can be used to determine the size of the source buffer that should be allocated to convey audio data for each processing pass.
ISpatialAudioClient::GetNativeStaticObjectTypeMask	Gets a channel mask which represents the subset of static speaker bed channels native to current rendering engine.
ISpatialAudioClient::GetStaticObjectPosition	Gets the position in 3D space of the specified static spatial audio channel.
ISpatialAudioClient::GetSupportedAudioObjectFormatEnumerator	Gets an IAudioFormatEnumerator that contains all supported audio formats for spatial audio objects, the first item in the list represents the most preferable format.
ISpatialAudioClient::IsAudioObjectFormatSupported	Gets a value indicating whether ISpatialAudioObjectRenderStream supports a the specified format.
ISpatialAudioClient::IsSpatialAudioStreamAvailable	When successful, gets a value indicating whether the currently active spatial rendering engine supports the specified spatial audio render stream.

Remarks

Get an instance of this interface by calling ActivateAudioInterfaceAsync, using the __uuidof operator to get the

class ID of the **ISpatialAudioClient** interface. The following example code shows how to initialize this interface.

```
PROPVARIANT var;
PropVariantInit(&var);
auto p = reinterpret_cast<SpatialAudioClientActivationParams *>
(CoTaskMemAlloc(sizeof(SpatialAudioClientActivationParams)));
if (nullptr == p) { ... }
p->tracingContextId = /* context identifier */;
p->appId = /* app identifier */;
p->majorVersion = /* app version info */;
p->majorVersionN = /* app version info */;
var.vt = VT_BLOB;
var.blob.cbSize = sizeof(*p);
var.blob.pBlobData = reinterpret_cast<BYTE *>(p);
hr = ActivateAudioInterfaceAsync(device, __uuidof(ISpatialAudioClient), &var, ...);
// ...
ropVariantClear(&var);
```

Note When using the **ISpatialAudioClient** interfaces on an Xbox One Development Kit (XDK) title, you must first call **EnableSpatialAudio** before calling IMMDeviceEnumerator::EnumAudioEndpoints or IMMDeviceEnumerator::GetDefaultAudioEndpoint. Failure to do so will result in an E_NOINTERFACE error being returned from the call to Activate. **EnableSpatialAudio** is only available for XDK titles, and does not need to be called for Universal Windows Platform apps running on Xbox One, nor for any non-Xbox One devices.

To access the **ActivateAudioIntefaceAsync**, you will need to link to mmdevapi.lib.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudioclient.h

ISpatialAudioClient::ActivateSpatialAudioStream method

1/11/2020 • 2 minutes to read • Edit Online

Activates and initializes spatial audio stream using one of the spatial audio stream activation structures.

Syntax

```
HRESULT ActivateSpatialAudioStream(
  const PROPVARIANT *activationParams,
  REFIID            riid,
  void              **stream
);
```

Parameters

`activationParams`

The structure defining the activation parameters for the spatial audio stream. The **vt** field should be set to VT_BLOB and the **blob** field should be populated with a SpatialAudioObjectRenderStreamActivationParams or a SpatialAudioObjectRenderStreamForMetadataActivationParams.

`riid`

The UUID of the spatial audio stream interface to activate.

`stream`

A pointer to the pointer which receives the activated spatial audio interface.

Return value

If the method succeeds, it returns S_OK.

Remarks

This method supports activation of the following spatial audio stream interfaces:

ISpatialAudioObjectRenderStream

ISpatialAudioObjectRenderStreamForMetadata

Examples

```
Microsoft::WRL::ComPtr<ISpatialAudioClient> spatialAudioClient;

// Activate ISpatialAudioClient on the desired audio-device
hr = defaultDevice->Activate(__uuidof(ISpatialAudioClient), CLSCTX_INPROC_SERVER, nullptr,
(void**)&spatialAudioClient);

hr = spatialAudioClient->IsAudioObjectFormatSupported(&format);

// Create the event that will be used to signal the client for more data
HANDLE bufferCompletionEvent = CreateEvent(nullptr, FALSE, FALSE, nullptr);

SpatialAudioObjectRenderStreamActivationParams streamParams;
streamParams.ObjectFormat = &format;
streamParams.StaticObjectTypeMask = ChannelMask_Stereo;
streamParams.MinDynamicObjectCount = 0;
streamParams.MaxDynamicObjectCount = 0;
streamParams.Category = AudioCategory_SoundEffects;
streamParams.EventHandle = bufferCompletionEvent;
streamParams.NotifyObject = nullptr;

PROPVARIANT activationParams;
PropVariantInit(&activationParams);
activationParams.vt = VT_BLOB;
activationParams.blob.cbSize = sizeof(streamParams);
activationParams.blob.pBlobData = reinterpret_cast<BYTE *>(&streamParams);

Microsoft::WRL::ComPtr<ISpatialAudioObjectRenderStream> spatialAudioStream;
hr = spatialAudioClient->ActivateSpatialAudioStream(&activationParams, __uuidof(spatialAudioStream),
(void**)&spatialAudioStream);
```

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioClient

SpatialAudioObjectRenderStreamActivationParams

SpatialAudioObjectRenderStreamForMetadataActivationParams

ISpatialAudioClient::GetMaxDynamicObjectCount method

1/11/2020 • 2 minutes to read • Edit Online

Gets the maximum number of dynamic audio objects for the spatial audio client.

Syntax

```
HRESULT GetMaxDynamicObjectCount(
  UINT32 *value
);
```

Parameters

`value`

Gets the maximum dynamic object count for this client.

Return value

If the method succeeds, it returns S_OK.

Remarks

A dynamic ISpatialAudioObject is one that was activated by setting the *type* parameter to the ISpatialAudioObjectRenderStream::ActivateSpatialAudioObject method to **AudioObjectType_Dynamic**. The client has a limit of the maximum number of dynamic spatial audio objects that can be activated at one time. When the capacity of the audio rendering pipeline changes, the system will dynamically adjust the maximum number of concurrent dynamic spatial audio objects. Before doing so, the system will call OnAvailableDynamicObjectCountChange to notify clients of the resource limit change.

Call Release on an **ISpatialAudioObject** when it is no longer being used to free up the resource to create new dynamic spatial audio objects.

When Windows Sonic is not available (for instance, when playing to embedded laptop stereo speakers, or if the user has not explicitly enabled Windows Sonic on the device), the number of available dynamic objects returned by **GetMaxDynamicObjectCount** to an application will be 0.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioClient::GetMaxFrameCount method

1/11/2020 • 2 minutes to read • Edit Online

Gets the maximum possible frame count per processing pass. This method can be used to determine the size of the source buffer that should be allocated to convey audio data for each processing pass.

Syntax

```
HRESULT GetMaxFrameCount(
  const WAVEFORMATEX *objectFormat,
  UINT32             *frameCountPerBuffer
);
```

Parameters

`objectFormat`

The audio format used to calculate the maximum frame count. This should be the same format specified in the **ObjectFormat** field of the SpatialAudioObjectRenderStreamActivationParams passed to ActivateSpatialAudioStream.

`frameCountPerBuffer`

The maximum number of audio frames that will be processed in one pass.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioClient

ISpatialAudioClient::GetNativeStaticObjectTypeMask method

1/11/2020 • 2 minutes to read • Edit Online

Gets a channel mask which represents the subset of static speaker bed channels native to current rendering engine.

Syntax

```
HRESULT GetNativeStaticObjectTypeMask(
  AudioObjectType *mask
);
```

Parameters

`mask`

A bitwise combination of values from the AudioObjectType enumeration indicating a subset of static speaker channels. The values returned will only include the static channel values and will not include **AudioObjectType_Dynamic**.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioClient

ISpatialAudioClient::GetStaticObjectPosition method

1/11/2020 • 2 minutes to read • Edit Online

Gets the position in 3D space of the specified static spatial audio channel.

Syntax

```
HRESULT GetStaticObjectPosition(
  AudioObjectType type,
  float           *x,
  float           *y,
  float           *z
);
```

Parameters

`type`

A value indicating the static spatial audio channel for which the position is being queried. This method will return E_INVALIDARG if the value does not represent a static channel, including **AudioObjectType_Dynamic** and **AudioObjectType_None**.

`x`

The x coordinate of the static audio channel, in meters, relative to the listener. Positive values are to the right of the listener and negative values are to the left.

`y`

The y coordinate of the static audio channel, in meters, relative to the listener. Positive values are above the listener and negative values are below.

`z`

The z coordinate of the static audio channel, in meters, relative to the listener. Positive values are behind the listener and negative values are in front.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	The supplied AudioObjectType value does not represent a static channel.

Remarks

Position values use a right-handed Cartesian coordinate system, where each unit represents 1 meter. The coordinate system is relative to the listener where the origin (x=0.0, y=0.0, z=0.0) represents the center point between the listener's ears.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioClient

ISpatialAudioClient::GetSupportedAudioObjectFormatEnumerator method

1/11/2020 • 2 minutes to read • Edit Online

Gets an IAudioFormatEnumerator that contains all supported audio formats for spatial audio objects, the first item in the list represents the most preferable format.

Syntax

```
HRESULT GetSupportedAudioObjectFormatEnumerator(
  IAudioFormatEnumerator **enumerator
);
```

Parameters

`enumerator`

Pointer to the pointer that receives the IAudioFormatEnumerator interface.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioClient

ISpatialAudioClient::IsAudioObjectFormatSupported method

1/11/2020 • 2 minutes to read • Edit Online

Gets a value indicating whether ISpatialAudioObjectRenderStream supports a the specified format.

Syntax

```
HRESULT IsAudioObjectFormatSupported(
  const WAVEFORMATEX *objectFormat
);
```

Parameters

`objectFormat`

The format for which support is queried.

Return value

If the specified format is supported, it returns S_OK. If specified format is unsupported, this method returns AUDCLNT_E_UNSUPPORTED_FORMAT.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioClient

ISpatialAudioClient::IsSpatialAudioStreamAvailable method

1/11/2020 • 2 minutes to read • Edit Online

When successful, gets a value indicating whether the currently active spatial rendering engine supports the specified spatial audio render stream.

Syntax

```
HRESULT IsSpatialAudioStreamAvailable(
  REFIID            streamUuid,
  const PROPVARIANT *auxiliaryInfo
);
```

Parameters

`streamUuid`

The interface ID of the interface for which availability is queried.

`auxiliaryInfo`

A structure containing additional information to be used when support is queried. For more information, see Remarks.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_S TRE AM _IS_ NO T_A VAI LAB LE	The specified stream interface can't be activated by the currently active rendering engine.

SPT LAU DCL NT_ E_M ETA DAT A_F OR MA T_IS _NO T_S UPP ORT ED	The metadata format supplied in the *auxiliaryInfo* parameter is not supported by the current rendering engine. For more information, see Remarks..

Remarks

When querying to see if the ISpatialAudioObjectRenderStreamForMetadata you can use the auxilaryInfo parameter to query if a particular metadata format is supported. The following code example demonstrates how to initialize the PROPVARIANT structure to check for support for an example metadata format.

```
PROPVARIANT auxiliaryInfo;
auxiliaryInfo.vt = VT_CLSID;
auxiliaryInfo.puuid = const_cast<CLSID*>(&CONTOSO_SPATIAL_METADATA_V1_0);
```

If the specified metadata format is unsupported, **IsSpatialAudioStreamAvailable** returns SPTLAUDCLNT_E_METADATA_FORMAT_IS_NOT_SUPPORTED.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioClient

ISpatialAudioObject interface

1/23/2020 • 2 minutes to read • Edit Online

Represents an object that provides audio data to be rendered from a position in 3D space, relative to the user. Spatial audio objects can be static or dynamic, which you specify with the *type* parameter to the ISpatialAudioObjectRenderStream::ActivateSpatialAudioObject method. Dynamic audio objects can be placed in an arbitrary position in space and can be moved over time. Static audio objects are assigned to one or more channels, defined in the AudioObjectType enumeration, that each correlate to a fixed speaker location that may be a physical or a virtualized speaker.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioObject** interface inherits from ISpatialAudioObjectBase. **ISpatialAudioObject** also has these types of members:

- Methods

Methods

The **ISpatialAudioObject** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObject::SetPosition	Sets the position in 3D space, relative to the listener, from which the ISpatialAudioObject audio data will be rendered.
ISpatialAudioObject::SetVolume	Sets an audio amplitude multiplier that will be applied to the audio data provided by the ISpatialAudioObject before it is submitted to the audio rendering engine.

Remarks

> **Note** Many of the methods provided by this interface are implemented in the inherited ISpatialAudioObjectBase interface.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows

Header	spatialaudioclient.h

See also

ISpatialAudioObjectBase

ISpatialAudioObject::SetPosition method

1/11/2020 • 2 minutes to read • Edit Online

Sets the position in 3D space, relative to the listener, from which the ISpatialAudioObject audio data will be rendered.

Syntax

```
HRESULT SetPosition(
  float x,
  float y,
  float z
);
```

Parameters

`x`

The x position of the audio object, in meters, relative to the listener. Positive values are to the right of the listener and negative values are to the left.

`y`

The y position of the audio object, in meters, relative to the listener. Positive values are above the listener and negative values are below.

`z`

The z position of the audio object, in meters, relative to the listener. Positive values are behind the listener and negative values are in front.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_O UT_ OF_ OR DER	ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects was not called before the call to **SetPosition**.

SPT LAU DCL NT_ E_R ESO URC ES_I NV ALI DAT ED	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects).
SPT LAU DCL NT_ E_P RO PER TY_ NO T_S UPP ORT ED	The ISpatialAudioObject is not of type **AudioObjectType_Dynamic**. Set the type of the audio object with the *type* parameter to the ISpatialAudioObjectRenderStreamBase::ActivateSpatialAudioObject method.

Remarks

This method can only be called on a ISpatialAudioObject that is of type **AudioObjectType_Dynamic**. Set the type of the audio object with the *type* parameter to the ISpatialAudioObjectRenderStreamBase::ActivateSpatialAudioObject method.

Position values use a right-handed Cartesian coordinate system, where each unit represents 1 meter. The coordinate system is relative to the listener where the origin (x=0.0, y=0.0, z=0.0) represents the center point between the listener's ears.

If **SetPosition** is never called, the origin (x=0.0, y=0.0, z=0.0) is used as the default position. After **SetPosition** is called, the position that is set will be used for the audio object until the position is changed with another call to **SetPosition**.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObject

ISpatialAudioObject::SetVolume method

1/11/2020 • 2 minutes to read • Edit Online

Sets an audio amplitude multiplier that will be applied to the audio data provided by the ISpatialAudioObject before it is submitted to the audio rendering engine.

Syntax

```
HRESULT SetVolume(
  float volume
);
```

Parameters

`volume`

The amplitude multiplier for audio data. This must be a value between 0.0 and 1.0.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUDCLNT_E_OUTOFORDER	ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects was not called before the call to **SetVolume**.
SPTLAUDCLNT_E_RESOURCES_INVALIDATED	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects).

Remarks

If **SetVolume** is never called, the default value of 1.0 is used. After **SetVolume** is called, the volume that is set will be used for the audio object until the volume is changed with another call to **SetVolume**.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObject

ISpatialAudioObjectBase interface

1/23/2020 • 2 minutes to read • Edit Online

Base interface that represents an object that provides audio data to be rendered from a position in 3D space, relative to the user. Spatial audio objects can be static or dynamic, which you specify with the *type* parameter to the ISpatialAudioObjectRenderStream::ActivateSpatialAudioObject method. Dynamic audio objects can be placed in an arbitrary position in space and can be moved over time. Static audio objects are assigned to one or more channels, defined in the AudioObjectType enumeration, that each correlate to a fixed speaker location that may be a physical or a virtualized speaker.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioObjectBase** interface inherits from the IUnknown interface. **ISpatialAudioObjectBase** also has these types of members:

- Methods

Methods

The **ISpatialAudioObjectBase** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObjectBase::GetAudioObjectType	Gets a value specifying the type of audio object that is represented by the ISpatialAudioObject.
ISpatialAudioObjectBase::GetBuffer	Gets a buffer that is used to supply the audio data for the ISpatialAudioObject.
ISpatialAudioObjectBase::IsActive	Gets a boolean value indicating whether the ISpatialAudioObject is valid.
ISpatialAudioObjectBase::SetEndOfStream	Instructs the system that the final block of audio data has been submitted for the ISpatialAudioObject so that the object can be deactivated and it's resources reused.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudioclient.h

ISpatialAudioObjectBase::GetAudioObjectType method

1/11/2020 • 2 minutes to read • Edit Online

Gets a value specifying the type of audio object that is represented by the ISpatialAudioObject. This value indicates if the object is dynamic or static. If the object is static, one and only one of the static audio channel values to which the object is assigned is returned.

Syntax

```
HRESULT GetAudioObjectType(
  AudioObjectType *audioObjectType
);
```

Parameters

`audioObjectType`

A value specifying the type of audio object that is represented

Return value

If the method succeeds, it returns S_OK.

Remarks

Set the type of the audio object with the *type* parameter to the ISpatialAudioObjectRenderStream::ActivateSpatialAudioObject method.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObject

ISpatialAudioObjectBase

ISpatialAudioObjectBase::GetBuffer method

1/11/2020 • 2 minutes to read • Edit Online

Gets a buffer that is used to supply the audio data for the ISpatialAudioObject.

Syntax

```
HRESULT GetBuffer(
  BYTE   **buffer,
  UINT32 *bufferLength
);
```

Parameters

`buffer`

The buffer into which audio data is written.

`bufferLength`

The length of the buffer in bytes. This length will be the value returned in the *frameCountPerBuffer* parameter to ISpatialAudioObjectRenderStream::BeginUpdatingAudioObjects multiplied by the value of the **nBlockAlign** field of the WAVEFORMATEX structure passed in the SpatialAudioObjectRenderStreamActivationParams parameter to ISpatialAudioClient::ActivateSpatialAudioStream.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_O UT_ OF_ OR DER	ISpatialAudioObjectRenderStream::BeginUpdatingAudioObjects was not called before the call to **GetBuffer**. This method must be called before the first time **GetBuffer** is called and after every subsequent call to ISpatialAudioObjectRenderStream::EndUpdatingAudioObjects.

SPT **LAU** **DCL** **NT_** **E_R** **ESO** **URC** **ES_I** **NV** **ALI** **DAT** **ED**	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStream::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStream::EndUpdatingAudioObjects).

Remarks

The first time **GetBuffer** is called after the ISpatialAudioObject is activated with a call ISpatialAudioObjectRenderStream::ActivateSpatialAudioObject,
lifetime of the spatial audio object starts.
To keep the spatial audio object alive after that, this **GetBuffer** must be called on every processing pass (between calls to ISpatialAudioObjectRenderStream::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStream::EndUpdatingAudioObjects). If **GetBuffer** is not called within an audio processing pass, SetEndOfStream is called implicitly on the audio object to deactivate, and the audio object can only be reused after calling Release on the object and then reactivating the object by calling **ActivateSpatialAudioObject** again.

The pointers retrieved by **GetBuffer** should not be used after ISpatialAudioObjectRenderStream::EndUpdatingAudioObjects has been called.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObject

ISpatialAudioObjectBase

ISpatialAudioObjectBase::IsActive method

1/11/2020 • 2 minutes to read • Edit Online

Gets a boolean value indicating whether the ISpatialAudioObject is valid.

Syntax

```
HRESULT IsActive(
  BOOL *isActive
);
```

Parameters

`isActive`

TRUE if the audio object is currently valid; otherwise, **FALSE**.

Return value

If the method succeeds, it returns S_OK.

Remarks

If this value is false, you should call Release to make the audio object resource available in the future.

IsActive will be set to false after SetEndOfStream is called implicitly or explicitly. **SetEndOfStream** is called implicitly by the system if GetBuffer is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStream::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStream::EndUpdatingAudioObjects).

The rendering engine will also deactivate the audio object, setting **IsActive** to false, when audio object resources become unavailable. In this case, a notification is sent via ISpatialAudioObjectRenderStreamNotify before the object is deactivated. The value returned in the *availableDynamicObjectCount* parameter to ISpatialAudioObjectRenderStream::BeginUpdatingAudioObjects indicates how many objects will be processed for each pass.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObject

ISpatialAudioObjectBase

ISpatialAudioObjectBase::SetEndOfStream method

1/11/2020 • 2 minutes to read • Edit Online

Instructs the system that the final block of audio data has been submitted for the ISpatialAudioObject so that the object can be deactivated and it's resources reused.

Syntax

```
HRESULT SetEndOfStream(
  UINT32 frameCount
);
```

Parameters

`frameCount`

The number of audio frames in the audio buffer that should be included in the final processing pass. This number may be smaller than or equal to the value returned in the *frameCountPerBuffer* parameter to ISpatialAudioObjectRenderStream::BeginUpdatingAudioObjects.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUDCLNT_E_OUT_OF_ORDER	ISpatialAudioObjectRenderStream::BeginUpdatingAudioObjects was not called before the call to **SetEndOfStream**.
SPTLAUDCLNT_E_RESOURCES_INVALIDATED	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStream::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStream::EndUpdatingAudioObjects).

Remarks

Call Release after calling **SetEndOfStream** to make free the audio object resources for future use.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObject

ISpatialAudioObjectBase

ISpatialAudioObjectRenderStream interface

1/23/2020 • 2 minutes to read • Edit Online

Provides methods for controlling a spatial audio object render stream, including starting, stopping, and resetting the stream. Also provides methods for activating new ISpatialAudioObject instances and notifying the system when you are beginning and ending the process of updating activated spatial audio objects and data.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioObjectRenderStream** interface inherits from ISpatialAudioObjectRenderStreamBase. **ISpatialAudioObjectRenderStream** also has these types of members:

- Methods

Methods

The **ISpatialAudioObjectRenderStream** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObjectRenderStream::ActivateSpatialAudioObject	Activates an ISpatialAudioObject for audio rendering.

Remarks

> **Note** Many of the methods provided by this interface are implemented in the inherited ISpatialAudioObjectRenderStreamBase interface.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStream::ActivateSpatialAudioObject method

1/11/2020 • 2 minutes to read • Edit Online

Activates an ISpatialAudioObject for audio rendering.

Syntax

```
HRESULT ActivateSpatialAudioObject(
  AudioObjectType     type,
  ISpatialAudioObject **audioObject
);
```

Parameters

`type`

The type of audio object to activate. For dynamic audio objects, this value must be **AudioObjectType_Dynamic**. For static audio objects, specify one of the static audio channel values from the enumeration. Specifying **AudioObjectType_None** will produce an audio object that is not spatialized.

`audioObject`

Receives a pointer to the activated interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_N O_ MO RE_ OBJ ECT S	The system has reached the maximum number of simultaneous audio objects.
SPT LAU DCL NT_ E_D EST RO YED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
SPT LAU DCL NT_ E_IN TER NAL	An internal error has occurred.
AU DCL NT_ E_U NS UPP ORT ED_ FOR MA T	The media associated with the spatial audio stream uses an unsupported format.

Remarks

A dynamic ISpatialAudioObject is one that was activated by setting the *type* parameter to the **ActivateSpatialAudioObject** method to **AudioObjectType_Dynamic**. The client has a limit of the maximum number of dynamic spatial audio objects that can be activated at one time. After the limit has been reached, attempting to activate additional audio objects will result in this method returning an SPTLAUDCLNT_E_NO_MORE_OBJECTS error. To avoid this, call Release on each dynamic **ISpatialAudioObject** after it is no longer being used to free up the resource so that it can be reallocated. See ISpatialAudioObject::IsActive and ISpatialAudioObject::SetEndOfStream for more information on the managing the lifetime of spatial audio objects.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStream

ISpatialAudioObjectRenderStreamBase interface

1/23/2020 • 2 minutes to read • Edit Online

Base interface that provides methods for controlling a spatial audio object render stream, including starting, stopping, and resetting the stream. Also provides methods for activating new ISpatialAudioObject instances and notifying the system when you are beginning and ending the process of updating activated spatial audio objects and data.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioObjectRenderStreamBase** interface inherits from the IUnknown interface. **ISpatialAudioObjectRenderStreamBase** also has these types of members:

- Methods

Methods

The **ISpatialAudioObjectRenderStreamBase** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects	Puts the system into the state where audio object data can be submitted for processing and the ISpatialAudioObject state can be modified.
ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects	Notifies the system that the app has finished supplying audio data for the spatial audio objects activated with ActivateSpatialAudioObject.
ISpatialAudioObjectRenderStreamBase::GetAvailableDynamicObjectCount	Gets the number of dynamic spatial audio objects that are currently available.
ISpatialAudioObjectRenderStreamBase::GetService	Gets additional services from the ISpatialAudioObjectRenderStream.
ISpatialAudioObjectRenderStreamBase::Reset	Reset a stopped audio stream.
ISpatialAudioObjectRenderStreamBase::Start	Starts the spatial audio stream.
ISpatialAudioObjectRenderStreamBase::Stop	Stops a running audio stream.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]

Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudioclient.h

ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects method

1/11/2020 • 2 minutes to read • Edit Online

Puts the system into the state where audio object data can be submitted for processing and the ISpatialAudioObject state can be modified.

Syntax

```
HRESULT BeginUpdatingAudioObjects(
  UINT32 *availableDynamicObjectCount,
  UINT32 *frameCountPerBuffer
);
```

Parameters

`availableDynamicObjectCount`

The number of dynamic audio objects that are available to be rendered for the current processing pass. All allocated static audio objects can be rendered in every pass. For information on audio object types, see AudioObjectType.

`frameCountPerBuffer`

The size, in audio frames, of the buffer returned by GetBuffer.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUDCLNT_E_OUTOFORDER	**BeginUpdatingAudioObjects** was called twice without a matching call to EndUpdatingAudioObjects between the two calls.
SPTLAUDCLNT_DESTROYED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
AU DCL NT_ E_R ESO URC ES_I NV ALI DAT ED	A resource associated with the spatial audio stream is no longer valid.
SPT LAU DCL NT_ E_IN TER NAL	An internal error has occurred.
AU DCL NT_ E_U NS UPP ORT ED_ FOR MA T	The media associated with the spatial audio stream uses an unsupported format.

Remarks

This method must be called each time the event passed in the SpatialAudioObjectRenderStreamActivationParams to ISpatialAudioClient::ActivateSpatialAudioStream is signaled,
even if there no audio object data to submit.

For each **BeginUpdatingAudioObjects** call, there should be a corresponding call to EndUpdatingAudioObjects call.
If **BeginUpdatingAudioObjects** is called twice without a call **EndUpdatingAudioObjects** between them, the second call to **BeginUpdatingAudioObjects** will return SPTLAUDCLNT_E_OUT_OF_ORDER.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStream

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects method

1/11/2020 • 2 minutes to read • Edit Online

Notifies the system that the app has finished supplying audio data for the spatial audio objects activated with ActivateSpatialAudioObject.

Syntax

```
HRESULT EndUpdatingAudioObjects();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_O UT_ OF_ OR DER	**EndUpdatingAudioObjects** was called before BeginUpdatingAudioObjects.
SPT LAU DCL NT_ E_D EST RO YED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

AUDCLNT_E_RESOURCES_INVALIDATED	A resource associated with the spatial audio stream is no longer valid.
AUDCLNT_E_SPTLAUDCLNT_E_INTERNAL	An internal error has occurred.
AUDCLNT_E_UNSUPPORTED_FORMAT	The media associated with the spatial audio stream uses an unsupported format.

Remarks

The pointers retrieved with ISpatialAudioObjectBase::GetBuffer can no longer be used after this method is called.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStream

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStreamBase::GetAvailableDynamicObjectCount method

1/11/2020 • 2 minutes to read • Edit Online

Gets the number of dynamic spatial audio objects that are currently available.

Syntax

```
HRESULT GetAvailableDynamicObjectCount(
  UINT32 *value
);
```

Parameters

`value`

The number of dynamic spatial audio objects that are currently available.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

Remarks

A dynamic ISpatialAudioObject is one that was activated by setting the *type* parameter to the ActivateSpatialAudioObject method to **AudioObjectType_Dynamic**. The system has a limit of the maximum number of dynamic spatial audio objects that can be activated at one time. Call Release on an **ISpatialAudioObject** when it is no longer being used to free up the resource to create new dynamic spatial audio objects.

Requirements

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_D EST RO YED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio device associated with the spatial audio stream is no longer valid.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

SPT LAU DCL NT_ E_IN TER NAL	An internal error has occurred.
AU DCL NT_ E_U NS UPP ORT ED_ FOR MA T	**The media associated with the spatial audio stream uses an unsupported format.**
Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStream

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStreamBase::GetService method

1/11/2020 • 2 minutes to read • Edit Online

Gets additional services from the **ISpatialAudioObjectRenderStream**.

Syntax

```
HRESULT GetService(
  REFIID riid,
  void   **service
);
```

Parameters

`riid`

The interface ID for the requested service. The client should set this parameter to one of the following REFIID values:

IID_IAudioClock

IID_IAudioClock2

IID_IAudioStreamVolume

`service`

Pointer to a pointer variable into which the method writes the address of an instance of the requested interface. Through this method, the caller obtains a counted reference to the interface. The caller is responsible for releasing the interface, when it is no longer needed, by calling the interface's Release method. If the **GetService** call fails, **ppv*is NULL.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	Parameter *ppv* is NULL.

SPT LAU DCL NT_ E_D EST RO YED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
SPT LAU DCL NT_ E_IN TER NAL	An internal error has occurred.
AU DCL NT_ E_U NS UPP ORT ED_ FOR MA T	The media associated with the spatial audio stream uses an unsupported format.

Remarks

The **GetService** method supports the following service interfaces:

- IAudioClock
- IAudioClock2
- IAudioStreamVolume

Requirements

Target Platform	Windows

Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStream

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStreamBase::Reset method

1/11/2020 • 2 minutes to read • Edit Online

Reset a stopped audio stream.

Syntax

```
HRESULT Reset();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_S TRE AM _NO T_S TOP PED	The audio stream has not been stopped. Stop the stream by calling Stop.
SPT LAU DCL NT_ E_D EST RO YED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
SPT LAU DCL NT_ E_IN TER NAL	An internal error has occurred.
AU DCL NT_ E_U NS UPP ORT ED_ FOR MA T	The media associated with the spatial audio stream uses an unsupported format.

Remarks

Resetting the audio stream flushes all pending data and resets the audio clock stream position to 0. Resetting the stream also causes all active ISpatialAudioObject instances to be revoked.
A subsequent call to Start causes the stream to start from 0 position.

The stream must have been previously stopped with a call to Stop or the method will fail and return SPTLAUDCLNT_E_STREAM_NOT_STOPPED.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStream

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStreamBase::Start method

1/11/2020 • 2 minutes to read • Edit Online

Starts the spatial audio stream.

Syntax

```
HRESULT Start();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_S TRE AM _NO T_S TOP PED	The audio stream has not been stopped. Stop the stream by calling Stop.
SPT LAU DCL NT_ E_D EST RO YED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.

AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
SPT LAU DCL NT_ E_IN TER NAL	An internal error has occurred.
AU DCL NT_ E_U NS UPP ORT ED_ FOR MA T	The media associated with the spatial audio stream uses an unsupported format.

Remarks

Starting the stream causes data flow between the endpoint buffer and the audio engine.
The first time this method is called, the stream's audio clock position will be at 0.
Otherwise, the clock resumes from its position at the time that the stream was last paused with a call to Stop.
Call Reset to reset the clock position to 0 and cause all active ISpatialAudioObject instances to be revoked.

The stream must have been previously stopped with a call to Stop or the method will fail and return SPTLAUDCLNT_E_STREAM_NOT_STOPPED.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStream

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStreamBase::Stop method

1/11/2020 • 2 minutes to read • Edit Online

Stops a running audio stream.

Syntax

```
HRESULT Stop();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_D EST RO YED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.
AU DCL NT_ E_D EVI CE_I NV ALI DAT ED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
SPT LAU DCL NT_ E_IN TER NAL	An internal error has occurred.

AU DCL NT_ E_U NS UPP ORT ED_ FOR MA T	The media associated with the spatial audio stream uses an unsupported format.

Remarks

Stopping stream causes data to stop flowing between the endpoint buffer and the audio engine.
You can consider this operation to pause the stream because it leaves the stream's audio clock at its current stream position and does not reset it to 0. A subsequent call to Start causes the stream to resume running from the current position.
Call Reset to reset the clock position to 0 and cause all active ISpatialAudioObject instances to be revoked.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStream

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStreamNotify interface

1/23/2020 • 2 minutes to read • Edit Online

Provides notifications for spatial audio clients to respond to changes in the state of an ISpatialAudioObjectRenderStream.

You register the object that implements this interface by assigning it to the *NotifyObject* parameter of the SpatialAudioClientActivationParams structure passed into the ISpatialAudioClient::ActivateSpatialAudioStream method. After registering its **ISpatialAudioObjectRenderStreamNotify** interface, the client receives event notifications in the form of callbacks through the OnAvailableDynamicObjectCountChange method in the interface.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioObjectRenderStreamNotify** interface inherits from the IUnknown interface. **ISpatialAudioObjectRenderStreamNotify** also has these types of members:

- Methods

Methods

The **ISpatialAudioObjectRenderStreamNotify** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObjectRenderStreamNotify::OnAvailableDynamicObjectCountChange	Notifies the spatial audio client when the rendering capacity for an ISpatialAudioObjectRenderStream is about to change, specifies the time after which the change will occur, and specifies the number of dynamic audio objects that will be available after the change.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudioclient.h

ISpatialAudioObjectRenderStreamNotify::OnAvailableDynamicObjectCountChange method

1/11/2020 • 2 minutes to read • Edit Online

Notifies the spatial audio client when the rendering capacity for an ISpatialAudioObjectRenderStream is about to change, specifies the time after which the change will occur, and specifies the number of dynamic audio objects that will be available after the change.

Syntax

```
HRESULT OnAvailableDynamicObjectCountChange(
  ISpatialAudioObjectRenderStreamBase *sender,
  LONGLONG                            hnsComplianceDeadlineTime,
  UINT32                              availableDynamicObjectCountChange
);
```

Parameters

`sender`

The spatial audio render stream for which the available dynamic object count is changing.

`hnsComplianceDeadlineTime`

The time after which the spatial resource limit will change, in 100-nanosecond units. A value of 0 means that the change will occur immediately.

`availableDynamicObjectCountChange`

The number of dynamic spatial audio objects that will be available to the stream after *hnsComplianceDeadlineTime*.

Return value

If the method succeeds, it returns S_OK. If it fails, it returns an error code.

Remarks

A dynamic ISpatialAudioObject is one that was activated by setting the *type* parameter to the ISpatialAudioObjectRenderStream::ActivateSpatialAudioObject method to **AudioObjectType_Dynamic**. The client has a limit of the maximum number of dynamic spatial audio objects that can be activated at one time. When the capacity of the audio rendering pipeline changes, the system will dynamically adjust the maximum number of concurrent dynamic spatial audio objects. Before doing so, the system will call **OnAvailableDynamicObjectCountChange** to notify clients of the resource limit change.

Call Release on an **ISpatialAudioObject** when it is no longer being used to free up the resource to create new dynamic spatial audio objects.

Requirements

Target Platform	Windows
Header	spatialaudioclient.h

See also

ISpatialAudioObjectRenderStreamNotify

SpatialAudioClientActivationParams structure

1/23/2020 • 2 minutes to read • Edit Online

Represents optional activation parameters for a spatial audio render stream. Pass this structure to ActivateAudioInterfaceAsync when activating an ISpatialAudioClient interface.

Syntax

```
typedef struct SpatialAudioClientActivationParams {
  GUID tracingContextId;
  GUID appId;
  int  majorVersion;
  int  minorVersion1;
  int  minorVersion2;
  int  minorVersion3;
} SpatialAudioClientActivationParams;
```

Members

`tracingContextId`

An app-defined context identifier, used for event logging.

`appId`

An identifier for the client app, used for event logging.

majorVersion

The major version number of the client app, used for event logging.

minorVersion1

The first minor version number of the client app, used for event logging.

minorVersion2

The second minor version number of the client app, used for event logging.

####### minorVersion3

The third minor version number of the client app, used for event logging.

`majorVersion`

`minorVersion1`

`minorVersion2`

`minorVersion3`

Remarks

The following example code shows how to initialize this structure.

```
PROPVARIANT var;
PropVariantInit(&var);
auto p = reinterpret_cast<SpatialAudioClientActivationParams *>
(CoTaskMemAlloc(sizeof(SpatialAudioClientActivationParams)));
if (nullptr == p) { ... }
p->tracingContextId = /* context identifier */;
p->appId = /* app identifier */;
p->majorVersion = /* app version info */;
p->majorVersionN = /* app version info */;
var.vt = VT_BLOB;
var.blob.cbSize = sizeof(*p);
var.blob.pBlobData = reinterpret_cast<BYTE *>(p);
hr = ActivateAudioInterfaceAsync(device, __uuidof(ISpatialAudioClient), &var, ...);
// ...
ropVariantClear(&var);
```

To access the **ActivateAudioIntefaceAsync**, you will need to link to mmdevapi.lib.

Requirements

Header	spatialaudioclient.h

SpatialAudioObjectRenderStreamActivationParams structure

1/23/2020 • 2 minutes to read • Edit Online

Represents activation parameters for a spatial audio render stream. Pass this structure to ISpatialAudioClient::ActivateSpatialAudioStream when activating a stream.

Syntax

```
typedef struct SpatialAudioObjectRenderStreamActivationParams {
  const WAVEFORMATEX                    *ObjectFormat;
  AudioObjectType                       StaticObjectTypeMask;
  UINT32                                MinDynamicObjectCount;
  UINT32                                MaxDynamicObjectCount;
  AUDIO_STREAM_CATEGORY                 Category;
  HANDLE                                EventHandle;
  ISpatialAudioObjectRenderStreamNotify *NotifyObject;
} SpatialAudioObjectRenderStreamActivationParams;
```

Members

`ObjectFormat`

Format descriptor for a single spatial audio object. All objects used by the stream must have the same format and the format must be of type WAVEFORMATEX or WAVEFORMATEXTENSIBLE.

`StaticObjectTypeMask`

A bitwise combination of **AudioObjectType** values indicating the set of static spatial audio channels that will be allowed by the activated stream.

`MinDynamicObjectCount`

The minimum number of concurrent dynamic objects. If this number of dynamic audio objects can't be activated simultaneously, ISpatialAudioClient::ActivateSpatialAudioStream will fail with this error **SPTLAUDCLNT_E_NO_MORE_OBJECTS**.

`MaxDynamicObjectCount`

The maximum number of concurrent dynamic objects that can be activated with ISpatialAudioObjectRenderStream.

`Category`

The category of the audio stream and its spatial audio objects.

`EventHandle`

The event that will signal the client to provide more audio data. This handle will be duplicated internally before it is used.

`NotifyObject`

The object that provides notifications for spatial audio clients to respond to changes in the state of an

ISpatialAudioObjectRenderStream. This object is used to notify clients that the number of dynamic spatial audio objects that can be activated concurrently is about to change.

Requirements

Header	spatialaudioclient.h

spatialaudiohrtf.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs spatialaudiohrtf.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
ISpatialAudioObjectForHrtf	Represents an object that provides audio data to be rendered from a position in 3D space, relative to the user, a head-relative transfer function (HRTF).
ISpatialAudioObjectRenderStreamForHrtf	Provides methods for controlling an Hrtf spatial audio object render stream, including starting, stopping, and resetting the stream.

Structures

TITLE	DESCRIPTION
SpatialAudioHrtfActivationParams	Specifies the activation parameters for an ISpatialAudioRenderStreamForHrtf.
SpatialAudioHrtfDirectivity	Represents an omnidirectional model for an ISpatialAudioObjectForHrtf. The omnidirectional emission is interpolated linearly with the directivity model specified in the Type field based on the value of the Scaling field.
SpatialAudioHrtfDirectivityCardioid	Represents a cardioid-shaped directivity model for an ISpatialAudioObjectForHrtf.
SpatialAudioHrtfDirectivityCone	Represents a cone-shaped directivity model for an ISpatialAudioObjectForHrtf.
SpatialAudioHrtfDirectivityUnion	Defines a spatial audio directivity model for an ISpatialAudioObjectForHrtf.
SpatialAudioHrtfDistanceDecay	Represents the decay model that is applied over distance from the position of an ISpatialAudioObjectForHrtf to the position of the listener.

Enumerations

TITLE	DESCRIPTION
SpatialAudioHrtfDirectivityType	Specifies the shape in which sound is emitted by an ISpatialAudioObjectForHrtf.

TITLE	DESCRIPTION
SpatialAudioHrtfDistanceDecayType	Specifies the type of decay applied over distance from the position of an ISpatialAudioObjectForHrtf to the position of the listener.
SpatialAudioHrtfEnvironmentType	Specifies the type of acoustic environment that is simulated when audio is processed for an ISpatialAudioObjectForHrtf.

ISpatialAudioObjectForHrtf interface

1/23/2020 • 2 minutes to read • Edit Online

Represents an object that provides audio data to be rendered from a position in 3D space, relative to the user, a head-relative transfer function (HRTF). Spatial audio objects can be static or dynamic, which you specify with the *type* parameter to the ISpatialAudioObjectRenderStreamForHrtf::ActivateSpatialAudioObjectForHrtf method. Dynamic audio objects can be placed in an arbitrary position in space and can be moved over time. Static audio objects are assigned to one or more channels, defined in the AudioObjectType enumeration, that each correlate to a fixed speaker location that may be a physical or a virtualized speaker

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioObjectForHrtf** interface inherits from ISpatialAudioObjectBase.
ISpatialAudioObjectForHrtf also has these types of members:

- Methods

Methods

The **ISpatialAudioObjectForHrtf** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObjectForHrtf::SetDirectivity	Sets the spatial audio directivity model for the ISpatialAudioObjectForHrtf.
ISpatialAudioObjectForHrtf::SetDistanceDecay	Sets the decay model that is applied over distance from the position of an ISpatialAudioObjectForHrtf to the position of the listener.
ISpatialAudioObjectForHrtf::SetEnvironment	Sets the type of acoustic environment that is simulated when audio is processed for the ISpatialAudioObjectForHrtf.
ISpatialAudioObjectForHrtf::SetGain	Sets the gain for the ISpatialAudioObjectForHrtf.
ISpatialAudioObjectForHrtf::SetOrientation	Sets the orientation in 3D space, relative to the listener's frame of reference, from which the ISpatialAudioObjectForHrtf audio data will be rendered.
ISpatialAudioObjectForHrtf::SetPosition	Sets the position in 3D space, relative to the listener, from which the ISpatialAudioObjectForHrtf audio data will be rendered.

Remarks

Note Many of the methods provided by this interface are implemented in the inherited ISpatialAudioObjectBase interface.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudiohrtf.h

See also

ISpatialAudioObjectBase

ISpatialAudioObjectForHrtf::SetDirectivity method

1/11/2020 • 2 minutes to read • Edit Online

Sets the spatial audio directivity model for the ISpatialAudioObjectForHrtf.

Syntax

```
HRESULT SetDirectivity(
  SpatialAudioHrtfDirectivityUnion *directivity
);
```

Parameters

`directivity`

The spatial audio directivity model. This value can be one of the following structures:

- SpatialAudioHrtfDirectivity
- SpatialAudioHrtfDirectivityCardioid
- SpatialAudioHrtfDirectivityCone

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_O UT_ OF_ OR DER	ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects was not called before the call to **SetDirectivity**.
SPT LAU DCL NT_ E_R ESO URC ES_I NV ALI DAT ED	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects).

Remarks

The SpatialAudioHrtfDirectivity structure represents an omnidirectional model that can be linearly interpolated with a cardioid or cone model.

If **SetDirectivity** is not called, the default type of SpatialAudioHrtfDirectivity_OmniDirectional is used with no interpolation.

Requirements

Target Platform	Windows
Header	spatialaudiohrtf.h

See also

ISpatialAudioObjectForHrtf

ISpatialAudioObjectForHrtf::SetDistanceDecay method

1/11/2020 • 2 minutes to read • Edit Online

Sets the decay model that is applied over distance from the position of an ISpatialAudioObjectForHrtf to the position of the listener.

Syntax

```
HRESULT SetDistanceDecay(
  SpatialAudioHrtfDistanceDecay *distanceDecay
);
```

Parameters

`distanceDecay`

The decay model.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUDCLNT_E_OUTOFORDER	ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects was not called before the call to **SetDistanceDecay**.
SPTLAUDCLNT_E_RESOURCES_INVALIDATED	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects).

Remarks

If **SetEnvironment** is not called, the default values are used.

Requirements

Target Platform	Windows
Header	spatialaudiohrtf.h

See also

ISpatialAudioObjectForHrtf

ISpatialAudioObjectForHrtf::SetEnvironment method

1/11/2020 • 2 minutes to read • Edit Online

Sets the type of acoustic environment that is simulated when audio is processed for the ISpatialAudioObjectForHrtf.

Syntax

```
HRESULT SetEnvironment(
  SpatialAudioHrtfEnvironmentType environment
);
```

Parameters

`environment`

A value specifying the type of acoustic environment that is simulated when audio is processed for the ISpatialAudioObjectForHrtf.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_O UT_ OF_ OR DER	ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects was not called before the call to **SetEnvironment**.
SPT LAU DCL NT_ E_R ESO URC ES_I NV ALI DAT ED	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects).

Remarks

If **SetEnvironment** is not called, the default value of SpatialAudioHrtfEnvironment_Small is used.

Requirements

Target Platform	Windows
Header	spatialaudiohrtf.h

See also

ISpatialAudioObjectForHrtf

ISpatialAudioObjectForHrtf::SetGain method

1/11/2020 • 2 minutes to read • Edit Online

Sets the gain for the ISpatialAudioObjectForHrtf.

Syntax

```
HRESULT SetGain(
  float gain
);
```

Parameters

`gain`

The gain for the ISpatialAudioObjectForHrtf.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_O UT_ OF_ OR DER	ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects was not called before the call to **SetGain**.
SPT LAU DCL NT_ E_R ESO URC ES_I NV ALI DAT ED	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects).

Remarks

This is valid only for spatial audio objects configured to use the SpatialAudioHrtfDistanceDecay_CustomDecay

decay type. Set the decay type of an ISpatialAudioObjectForHrtf object by calling SetDistanceDecay. Set the default decay type for an all objects in an HRTF render stream by setting the **DistanceDecay** field of the SpatialAudioHrtfActivationParams passed into ISpatialAudioClient::ActivateSpatialAudioStream.

If **SetGain** is never called, the default value of 0.0 is used. After **SetGain** is called, the gain that is set will be used for the audio object until the gain is changed with another call to **SetGain**.

Requirements

Target Platform	Windows
Header	spatialaudiohrtf.h

See also

ISpatialAudioObjectForHrtf

ISpatialAudioObjectForHrtf::SetOrientation method

1/11/2020 • 2 minutes to read • Edit Online

Sets the orientation in 3D space, relative to the listener's frame of reference, from which the ISpatialAudioObjectForHrtf audio data will be rendered.

Syntax

```
HRESULT SetOrientation(
  const SpatialAudioHrtfOrientation *orientation
);
```

Parameters

`orientation`

An array of floats defining row-major 3x3 rotation matrix.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_O UT_ OF_ OR DER	ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects was not called before the call to **SetOrientation**.
SPT LAU DCL NT_ E_R ESO URC ES_I NV ALI DAT ED	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects).

Remarks

If **SetOrientation** is never called, the default value of an identity matrix is used. After **SetOrientation** is called, the orientation that is set will be used for the audio object until the orientation is changed with another call to **SetOrientation**.

Requirements

Target Platform	Windows
Header	spatialaudiohrtf.h

See also

ISpatialAudioObjectForHrtf

ISpatialAudioObjectForHrtf::SetPosition method

1/11/2020 • 2 minutes to read • Edit Online

Sets the position in 3D space, relative to the listener, from which the ISpatialAudioObjectForHrtf audio data will be rendered.

Syntax

```
HRESULT SetPosition(
  float x,
  float y,
  float z
);
```

Parameters

`x`

The x position of the audio object, in meters, relative to the listener. Positive values are to the right of the listener and negative values are to the left.

`y`

The y position of the audio object, in meters, relative to the listener. Positive values are above the listener and negative values are below.

`z`

The z position of the audio object, in meters, relative to the listener. Positive values are behind the listener and negative values are in front.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU DCL NT_ E_O UT_ OF_ OR DER	ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects was not called before the call to **SetPosition**.

SPT LAU DCL NT_ E_R ESO URC ES_I NV ALI DAT ED	SetEndOfStream was called either explicitly or implicitly in a previous audio processing pass. **SetEndOfStream** is called implicitly by the system if **GetBuffer** is not called within an audio processing pass (between calls to ISpatialAudioObjectRenderStreamBase::BeginUpdatingAudioObjects and ISpatialAudioObjectRenderStreamBase::EndUpdatingAudioObjects).
SPT LAU DCL NT_ E_P RO PER TY_ NO T_S UPP ORT ED	The ISpatialAudioObjectForHrtf is not of type **AudioObjectType_Dynamic**. Set the type of the audio object with the *type* parameter to the ISpatialAudioObjectRenderStreamBase::ActivateSpatialAudioObjectForHrtf method.

Remarks

This method can only be called on a ISpatialAudioObjectForHrtf that is of type **AudioObjectType_Dynamic**. Set the type of the audio object with the *type* parameter to the ISpatialAudioObjectRenderStreamForHrtf::ActivateSpatialAudioObjectForHrtf method.

Position values use a right-handed Cartesian coordinate system, where each unit represents 1 meter. The coordinate system is relative to the listener where the origin (x=0.0, y=0.0, z=0.0) represents the center point between the listener's ears.

If **SetPosition** is never called, the origin (x=0.0, y=0.0, z=0.0) is used as the default position. After **SetPosition** is called, the position that is set will be used for the audio object until the position is changed with another call to **SetPosition**.

Requirements

Target Platform	Windows
Header	spatialaudiohrtf.h

See also

ISpatialAudioObjectForHrtf

ISpatialAudioObjectRenderStreamForHrtf interface

1/23/2020 • 2 minutes to read • Edit Online

Provides methods for controlling an Hrtf spatial audio object render stream, including starting, stopping, and resetting the stream. Also provides methods for activating new ISpatialAudioObjectForHrtf instances and notifying the system when you are beginning and ending the process of updating activated spatial audio objects and data.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioObjectRenderStreamForHrtf** interface inherits from ISpatialAudioObjectRenderStreamBase. **ISpatialAudioObjectRenderStreamForHrtf** also has these types of members:

- Methods

Methods

The **ISpatialAudioObjectRenderStreamForHrtf** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObjectRenderStreamForHrtf::ActivateSpatialAudioObjectForHrtf	Activates an ISpatialAudioObjectForHrtf for audio rendering.

Remarks

> **Note** Many of the methods provided by this interface are implemented in the inherited ISpatialAudioObjectRenderStreamBase interface.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudiohrtf.h

See also

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStreamForHrtf::ActivateSpatialAudioObjectForHrtf method

1/11/2020 • 2 minutes to read • Edit Online

Activates an ISpatialAudioObjectForHrtf for audio rendering.

Syntax

```
HRESULT ActivateSpatialAudioObjectForHrtf(
  AudioObjectType            type,
  ISpatialAudioObjectForHrtf **audioObject
);
```

Parameters

`type`

The type of audio object to activate. For dynamic audio objects, this value must be **AudioObjectType_Dynamic**. For static audio objects, specify one of the static audio channel values from the enumeration. Specifying **AudioObjectType_None** will produce an audio object that is not spatialized.

`audioObject`

Receives a pointer to the activated interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUDCLNT_E_NO_MORE_OBJECTS	The system has reached the maximum number of simultaneous audio objects.
SPTLAUDCLNT_E_DESTROYED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.

SPT LAU DCL NT_ E_IN TER NAL	An internal error has occurred.
AU DCL NT_ E_U NS UPP ORT ED_ FOR MA T	The media associated with the spatial audio stream uses an unsupported format.

Remarks

A dynamic ISpatialAudioObjectForHrtf is one that was activated by setting the *type* parameter to the **ActivateSpatialAudioObjectForHrtf** method to **AudioObjectType_Dynamic**. The client has a limit of the maximum number of dynamic spatial audio objects that can be activated at one time. After the limit has been reached, attempting to activate additional audio objects will result in this method returning an SPTLAUDCLNT_E_NO_MORE_OBJECTS error. To avoid this, call Release on each dynamic **ISpatialAudioObjectForHrtf** after it is no longer being used to free up the resource so that it can be reallocated. See ISpatialAudioObjectgBase::IsActive and ISpatialAudioObjectgBase::SetEndOfStream for more information on the managing the lifetime of spatial audio objects.

Requirements

Target Platform	Windows
Header	spatialaudiohrtf.h

See also

ISpatialAudioRenderStreamForHrtf

SpatialAudioHrtfActivationParams structure

1/23/2020 • 2 minutes to read • Edit Online

Specifies the activation parameters for an ISpatialAudioRenderStreamForHrtf.

Syntax

```
typedef struct SpatialAudioHrtfActivationParams {
  const WAVEFORMATEX                    *ObjectFormat;
  AudioObjectType                       StaticObjectTypeMask;
  UINT32                                MinDynamicObjectCount;
  UINT32                                MaxDynamicObjectCount;
  AUDIO_STREAM_CATEGORY                 Category;
  HANDLE                                EventHandle;
  ISpatialAudioObjectRenderStreamNotify *NotifyObject;
  SpatialAudioHrtfDistanceDecay         *DistanceDecay;
  SpatialAudioHrtfDirectivityUnion      *Directivity;
  SpatialAudioHrtfEnvironmentType       *Environment;
  SpatialAudioHrtfOrientation           *Orientation;
} SpatialAudioHrtfActivationParams;
```

Members

`ObjectFormat`

Format descriptor for spatial audio objects associated with the stream. All objects must have the same format and must be of type WAVEFORMATEX or WAVEFORMATEXTENSIBLE.

`StaticObjectTypeMask`

A bitwise combination of **AudioObjectType** values indicating the set of static spatial audio channels that will be allowed by the activated stream.

`MinDynamicObjectCount`

The minimum number of concurrent dynamic objects. If this number of dynamic audio objects can't be activated simultaneously, no dynamic audio objects will be activated.

`MaxDynamicObjectCount`

The maximum number of concurrent dynamic objects that can be activated with ISpatialAudioRenderStreamForHrtf.

`Category`

The category of the audio stream and its spatial audio objects.

`EventHandle`

The event that will signal the client to provide more audio data. This handle will be duplicated internally before it is used.

`NotifyObject`

The object that provides notifications for spatial audio clients to respond to changes in the state of an ISpatialAudioRenderStreamForHrtf. This object is used to notify clients that the number of dynamic spatial audio

objects that can be activated concurrently is about to change.

`DistanceDecay`

Optional default value for the decay model used for ISpatialAudioObjectForHrtf objects associated with the stream. **nullptr** if unused.

`Directivity`

Optional default value for the spatial audio directivity model used for ISpatialAudioObjectForHrtf objects associated with the stream. **nullptr** if unused.

`Environment`

Optional default value for the type of environment that is simulated when audio is processed for ISpatialAudioObjectForHrtf objects associated with the stream. **nullptr** if unused.

`Orientation`

Optional default value for the orientation of ISpatialAudioObjectForHrtf objects associated with the stream. **nullptr** if unused.

Requirements

Header	spatialaudiohrtf.h

SpatialAudioHrtfDirectivity structure

1/23/2020 • 2 minutes to read • Edit Online

Represents an omnidirectional model for an ISpatialAudioObjectForHrtf. The omnidirectional emission is interpolated linearly with the directivity model specified in the **Type** field based on the value of the **Scaling** field.

Syntax

```
typedef struct SpatialAudioHrtfDirectivity {
  SpatialAudioHrtfDirectivityType Type;
  float                           Scaling;
} SpatialAudioHrtfDirectivity;
```

Members

`Type`

The type of shape in which sound is emitted by an ISpatialAudioObjectForHrtf.

`Scaling`

The amount of linear interpolation applied between omnidirectional sound and the directivity specified in the **Type** field. This is a normalized value between 0 and 1.0 where 0 is omnidirectional and 1.0 is full directivity using the specified type.

Requirements

Header	spatialaudiohrtf.h

SpatialAudioHrtfDirectivityCardioid structure

1/23/2020 • 2 minutes to read • Edit Online

Represents a cardioid-shaped directivity model for an ISpatialAudioObjectForHrtf.

Syntax

```
typedef struct SpatialAudioHrtfDirectivityCardioid {
  SpatialAudioHrtfDirectivity directivity;
  float                       Order;
} SpatialAudioHrtfDirectivityCardioid;
```

Members

`directivity`

A structure that expresses the direction in which sound is emitted by an ISpatialAudioObjectForHrtf.

`Order`

The order of the cardioid.

Requirements

Header	spatialaudiohrtf.h

SpatialAudioHrtfDirectivityCone structure

1/23/2020 • 2 minutes to read • Edit Online

Represents a cone-shaped directivity model for an ISpatialAudioObjectForHrtf.

Syntax

```
typedef struct SpatialAudioHrtfDirectivityCone {
  SpatialAudioHrtfDirectivity directivity;
  float                       InnerAngle;
  float                       OuterAngle;
} SpatialAudioHrtfDirectivityCone;
```

Members

`directivity`

A structure that expresses the direction in which sound is emitted by an ISpatialAudioObjectForHrtf.

`InnerAngle`

The inner angle of the cone.

`OuterAngle`

The outer angle of the cone.

Requirements

Header	spatialaudiohrtf.h

SpatialAudioHrtfDirectivityType enumeration

1/11/2020 • 2 minutes to read • Edit Online

Specifies the shape in which sound is emitted by an ISpatialAudioObjectForHrtf.

Syntax

```
typedef enum SpatialAudioHrtfDirectivityType {
  SpatialAudioHrtfDirectivity_OmniDirectional,
  SpatialAudioHrtfDirectivity_Cardioid,
  SpatialAudioHrtfDirectivity_Cone
} ;
```

Constants

SpatialAudioHrtfDirectivity_OmniDirectional	The sound is emitted in all directions.
SpatialAudioHrtfDirectivity_Cardioid	The sound is emitted in a cardioid shape.
SpatialAudioHrtfDirectivity_Cone	The sound is emitted in a cone shape.

Requirements

Header	spatialaudiohrtf.h

SpatialAudioHrtfDirectivityUnion union

1/23/2020 • 2 minutes to read • Edit Online

Defines a spatial audio directivity model for an ISpatialAudioObjectForHrtf.

Syntax

```
typedef union SpatialAudioHrtfDirectivityUnion {
  SpatialAudioHrtfDirectivityCone     Cone;
  SpatialAudioHrtfDirectivityCardioid Cardiod;
  SpatialAudioHrtfDirectivity         Omni;
} SpatialAudioHrtfDirectivityUnion;
```

Members

`Cone`

A cone-shaped directivity model

`Cardiod`

`Omni`

And omni-direction directivity model that can be interpolated linearly with one of the other directivity models.

Requirements

Header	spatialaudiohrtf.h

SpatialAudioHrtfDistanceDecay structure

1/23/2020 • 2 minutes to read • Edit Online

Represents the decay model that is applied over distance from the position of an ISpatialAudioObjectForHrtf to the position of the listener.

Syntax

```
typedef struct SpatialAudioHrtfDistanceDecay {
  SpatialAudioHrtfDistanceDecayType Type;
  float                             MaxGain;
  float                             MinGain;
  float                             UnityGainDistance;
  float                             CutoffDistance;
} SpatialAudioHrtfDistanceDecay;
```

Members

`Type`

The type of decay, natural or custom. The default value for this field is **SpatialAudioHrtfDistanceDecay_NaturalDecay**.

`MaxGain`

`MinGain`

`UnityGainDistance`

`CutoffDistance`

Requirements

Header	spatialaudiohrtf.h

SpatialAudioHrtfDistanceDecayType enumeration

1/11/2020 • 2 minutes to read • Edit Online

Specifies the type of decay applied over distance from the position of an ISpatialAudioObjectForHrtf to the position of the listener.

Syntax

```
typedef enum SpatialAudioHrtfDistanceDecayType {
  SpatialAudioHrtfDistanceDecay_NaturalDecay,
  SpatialAudioHrtfDistanceDecay_CustomDecay
} ;
```

Constants

SpatialAudioHrtfDistanceDecay_NaturalDecay	A natural decay over distance, as constrained by minimum and maximum gain distance limits. The output drops to silent at the distance specified by SpatialAudioHrtfDistanceDecay.CutoffDistance.
SpatialAudioHrtfDistanceDecay_CustomDecay	A custom gain curve, within the maximum and minimum gain limit.

Requirements

Header	spatialaudiohrtf.h

SpatialAudioHrtfEnvironmentType enumeration

1/11/2020 • 2 minutes to read • Edit Online

Specifies the type of acoustic environment that is simulated when audio is processed for an ISpatialAudioObjectForHrtf.

Syntax

```
typedef enum SpatialAudioHrtfEnvironmentType {
  SpatialAudioHrtfEnvironment_Small,
  SpatialAudioHrtfEnvironment_Medium,
  SpatialAudioHrtfEnvironment_Large,
  SpatialAudioHrtfEnvironment_Outdoors,
  SpatialAudioHrtfEnvironment_Average
} ;
```

Constants

SpatialAudioHrtfEnvironment_Small	A small room.
SpatialAudioHrtfEnvironment_Medium	A medium-sized room.
SpatialAudioHrtfEnvironment_Large	A large room.
SpatialAudioHrtfEnvironment_Outdoors	An outdoor space.
SpatialAudioHrtfEnvironment_Average	Reserved for Microsoft use. Apps should not use this value.

Requirements

Header	spatialaudiohrtf.h

spatialaudiometadata.h header

1/18/2020 • 2 minutes to read • Edit Online

This header is used by Core Audio APIs. For more information, see:

- Core Audio APIs spatialaudiometadata.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
ISpatialAudioMetadataClient	Provides a class factory for creating ISpatialAudioMetadataItems, ISpatialAudioMetadataWriter, ISpatialAudioMetadataReader, and ISpatialAudioMetadataCopier objects.
ISpatialAudioMetadataCopier	Provides methods for copying all or subsets of metadata items from a source SpatialAudioMetadataItems into a destination SpatialAudioMetadataItems.
ISpatialAudioMetadataItems	Represents a buffer of spatial audio metadata items.
ISpatialAudioMetadataItemsBuffer	Provides methods for attaching buffers to SpatialAudioMetadataItems for in-place storage of data.
ISpatialAudioMetadataReader	Provides methods for extracting spatial audio metadata items and item command value pairs from an ISpatialAudioMetadataItems object.
ISpatialAudioMetadataWriter	Provides methods for storing spatial audio metadata items positioned within a range of corresponding audio frames.
ISpatialAudioObjectForMetadataCommands	Used to write metadata commands for spatial audio.
ISpatialAudioObjectForMetadataItems	Used to write spatial audio metadata for applications that require multiple metadata items per buffer with frame-accurate placement.
ISpatialAudioObjectRenderStreamForMetadata	Provides methods for controlling a spatial audio object render stream for metadata, including starting, stopping, and resetting the stream.

Structures

TITLE	DESCRIPTION
SpatialAudioMetadataItemsInfo	Provides information about an ISpatialAudioMetadataItems object. Get a copy of this structure by calling GetInfo.

TITLE	DESCRIPTION
SpatialAudioObjectRenderStreamForMetadataActivationParams	Represents activation parameters for a spatial audio render stream for metadata. Pass this structure to ISpatialAudioClient::ActivateSpatialAudioStream when activating a stream.

Enumerations

TITLE	DESCRIPTION
SpatialAudioMetadataCopyMode	Specifies the copy mode used when calling ISpatialAudioMetadataCopier::CopyMetadataForFrames.
SpatialAudioMetadataWriterOverflowMode	Specifies the desired behavior when an ISpatialAudioMetadataWriter attempts to write more items into the metadata buffer than was specified when the client was initialized.

ISpatialAudioMetadataClient interface

1/23/2020 • 2 minutes to read • Edit Online

Provides a class factory for creating ISpatialAudioMetadataItems, ISpatialAudioMetadataWriter, ISpatialAudioMetadataReader, and ISpatialAudioMetadataCopier objects. When an **ISpatialAudioMetadataItems** is activated, a metadata format ID is specified, which defines the metadata format enforced for all objects created from this factory. If the specified format is not supported by the current audio render endpoint, the class factory will not successfully activate the interface and will return an error.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioMetadataClient** interface inherits from the IUnknown interface. **ISpatialAudioMetadataClient** also has these types of members:

- Methods

Methods

The **ISpatialAudioMetadataClient** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioMetadataClient::ActivateSpatialAudioMetadataCopier	Creates an ISpatialAudioMetadataWriter object for copying spatial audio metadata items from one ISpatialAudioMetadataItems object to another.
ISpatialAudioMetadataClient::ActivateSpatialAudioMetadataItems	Creates an ISpatialAudioMetadataItems object for storing spatial audio metadata items.
ISpatialAudioMetadataClient::ActivateSpatialAudioMetadataReader	Creates an ISpatialAudioMetadataWriter object for reading spatial audio metadata items from an ISpatialAudioMetadataItems object.
ISpatialAudioMetadataClient::ActivateSpatialAudioMetadataWriter	Creates an ISpatialAudioMetadataWriter object for writing spatial audio metadata items to an ISpatialAudioMetadataItems object.
ISpatialAudioMetadataClient::GetSpatialAudioMetadataItemsBufferLength	Gets the length of the buffer required to store the specified number of spatial audio metadata items.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]

Target Platform	Windows
Header	spatialaudiometadata.h

ISpatialAudioMetadataClient::ActivateSpatialAudioMetadataCopier method

1/11/2020 • 2 minutes to read • Edit Online

Creates an ISpatialAudioMetadataWriter object for copying spatial audio metadata items from one ISpatialAudioMetadataItems object to another.

Syntax

```
HRESULT ActivateSpatialAudioMetadataCopier(
  ISpatialAudioMetadataCopier **metadataCopier
);
```

Parameters

`metadataCopier`

Receives a pointer to an instance of ISpatialAudioMetadataWriter.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	The provided pointer is not valid.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataClient

ISpatialAudioMetadataClient::ActivateSpatialAudioMetadataItems method

1/11/2020 • 2 minutes to read • Edit Online

Creates an ISpatialAudioMetadataItems object for storing spatial audio metadata items.

Syntax

```
HRESULT ActivateSpatialAudioMetadataItems(
  UINT16                          maxItemCount,
  UINT16                          frameCount,
  ISpatialAudioMetadataItemsBuffer **metadataItemsBuffer,
  ISpatialAudioMetadataItems       **metadataItems
);
```

Parameters

`maxItemCount`

The maximum number of metadata items that can be stored in the returned ISpatialAudioMetadataItems.

`frameCount`

The valid range of frame offset positions for metadata items stored in the returned ISpatialAudioMetadataItems.

`metadataItemsBuffer`

If a pointer is supplied, returns an ISpatialAudioMetadataItemsBuffer interface which provides methods for attaching caller-provided memory for storage of metadata items. If this parameter is NULL, the object will allocate internal storage for the items. This interface cannot be obtained via QueryInterface.

`metadataItems`

Receives an instance ISpatialAudioMetadataItems object which can be populated with metadata items using an by ISpatialAudioMetadataWriter or ISpatialAudioMetadataCopier and can be read with an ISpatialAudioMetadataReader.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	The pointer provided in the *metadataItems* parameter is not valid. The value of *maxItemCount* or *frameCount* is 0.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataClient

ISpatialAudioMetadataClient::ActivateSpatialAudioMetadataReader method

1/11/2020 • 2 minutes to read • Edit Online

Creates an ISpatialAudioMetadataWriter object for reading spatial audio metadata items from an ISpatialAudioMetadataItems object.

Syntax

```
HRESULT ActivateSpatialAudioMetadataReader(
  ISpatialAudioMetadataReader **metadataReader
);
```

Parameters

`metadataReader`

Receives a pointer to an instance of ISpatialAudioMetadataReader.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN VAL IDA RG	The provided pointer is not valid.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataClient

ISpatialAudioMetadataClient::ActivateSpatialAudioMetadataWriter method

1/11/2020 • 2 minutes to read • Edit Online

Creates an ISpatialAudioMetadataWriter object for writing spatial audio metadata items to an ISpatialAudioMetadataItems object.

Syntax

```
HRESULT ActivateSpatialAudioMetadataWriter(
  SpatialAudioMetadataWriterOverflowMode overflowMode,
  ISpatialAudioMetadataWriter            **metadataWriter
);
```

Parameters

`overflowMode`

A value that specifies the behavior when attempting to write more metadata items to the ISpatialAudioMetadataItems than the maximum number of items specified when calling ActivateSpatialAudioMetadataItems.

`metadataWriter`

Receives a pointer to an instance of ISpatialAudioMetadataWriter.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_IN **VAL** **IDA** **RG**	The provided pointer is not valid.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataClient

ISpatialAudioMetadataClient::GetSpatialAudioMetadataItemsBufferLength method

1/11/2020 • 2 minutes to read • Edit Online

Gets the length of the buffer required to store the specified number of spatial audio metadata items. Use this method to determine the correct buffer size to use when attaching caller-provided memory through the ISpatialAudioMetadataItemsBuffer interface.

Syntax

```
HRESULT GetSpatialAudioMetadataItemsBufferLength(
  UINT16 maxItemCount,
  UINT32 *bufferLength
);
```

Parameters

`maxItemCount`

The maximum number of metadata items to be stored in an ISpatialAudioMetadataItems object.

`bufferLength`

The length of the buffer required to store the number of spatial audio metadata items specified in the *maxItemCount* parameter.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_INVALIDARG	The provided pointer is not valid. The value of *maxItemCount* or *frameCount* is 0.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataClient

ISpatialAudioMetadataCopier interface

1/23/2020 • 2 minutes to read • Edit Online

Provides methods for copying all or subsets of metadata items from a source SpatialAudioMetadataItems into a destination **SpatialAudioMetadataItems**. The **SpatialAudioMetadataItems** object, which is populated using an ISpatialAudioMetadataWriter or **ISpatialAudioMetadataCopier**, has a frame count, specified with the *frameCount* parameter to ActivateSpatialAudioMetadataItems, that represents the valid range of metadata item offsets. **ISpatialAudioMetadataReader** enables copying groups of items within a subrange of the total frame count. The object maintains an internal read position, which is advanced by the number of frames specified when a copy operation is performed.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioMetadataCopier** interface inherits from the IUnknown interface.
ISpatialAudioMetadataCopier also has these types of members:

- Methods

Methods

The **ISpatialAudioMetadataCopier** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioMetadataCopier::Close	Completes any necessary operations on the SpatialAudioMetadataItems object and releases the object.
ISpatialAudioMetadataCopier::CopyMetadataForFrames	Copies metadata items from the source ISpatialAudioMetadataItems, provided to the Open method, object to the destination ISpatialAudioMetadataItems object, specified with the dstMetadataItems parameter.
ISpatialAudioMetadataCopier::Open	Opens an ISpatialAudioMetadataItems object for copying.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudiometadata.h

ISpatialAudioMetadataCopier::Close method

1/11/2020 • 2 minutes to read • Edit Online

Completes any necessary operations on the SpatialAudioMetadataItems object and releases the object.

Syntax

```
HRESULT Close();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU D_ MD _CL NT_ E_N O_I TEM S_O PEN	The ISpatialAudioMetadataItems has not been opened for reading with a call to Open or the object has been closed for writing with a call to Close.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataCopier

ISpatialAudioMetadataReader

ISpatialAudioMetadataCopier::CopyMetadataForFrames method

1/11/2020 • 2 minutes to read • Edit Online

Copies metadata items from the source ISpatialAudioMetadataItems, provided to the Open method, object to the destination **ISpatialAudioMetadataItems** object, specified with the *dstMetadataItems* parameter. Each call advances the internal copy position by the number of frames in the *copyFrameCount* parameter.

Syntax

```
HRESULT CopyMetadataForFrames(
  UINT16                      copyFrameCount,
  SpatialAudioMetadataCopyMode copyMode,
  ISpatialAudioMetadataItems   *dstMetadataItems,
  UINT16                      *itemsCopied
);
```

Parameters

`copyFrameCount`

The number of frames from the current copy position for which metadata items are copied. After the copy, the internal copy position within the source **SpatialAudioMetadataItems** is advanced the value specified in this parameter. Set this value to 0 to copy the entire frame range contained in the source **SpatialAudioMetadataItems**.

`copyMode`

A value that specifies the copy mode for the operation.

`dstMetadataItems`

A pointer to the destination **SpatialAudioMetadataItems** for the copy operation.

`itemsCopied`

Receives number of metadata items copied in the operation.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION

SPT LAU D_ MD _CL NT_ E_N O_I TEM S_O PEN	The ISpatialAudioMetadataItems has not been opened for copying with a call to Open or the object has been closed for writing with a call to Close.
E_IN VAL IDA RG	One of the provided pointers is not valid.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataCopier

ISpatialAudioMetadataCopier::Open method

1/11/2020 • 2 minutes to read • Edit Online

Opens an ISpatialAudioMetadataItems object for copying.

Syntax

```
HRESULT Open(
  ISpatialAudioMetadataItems *metadataItems
);
```

Parameters

`metadataItems`

A pointer to an ISpatialAudioMetadataItems object to be opened for copying

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUD_MD_CLNT_E_ITEMS_ALREADY_OPEN	**Open** has already been called on the supplied ISpatialAudioMetadataItems since the object was created or since the last call to Close.
E_INVALIDARG	The provided pointer is not valid.

Requirements

Target Platform	Windows

Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataCopier

ISpatialAudioMetadataReader

ISpatialAudioMetadataItems interface

1/23/2020 • 2 minutes to read • Edit Online

Represents a buffer of spatial audio metadata items. Metadata commands and values can be written to, read from, and copied between ISpatialAudioMetadataItems using the ISpatialAudioMetadataWriter, ISpatialAudioMetadataReader, and ISpatialAudioMetadataCopier interfaces. Use caller-allocated memory to store metadata items by creating an ISpatialAudioMetadataItemsBuffer.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioMetadataItems** interface inherits from the IUnknown interface.
ISpatialAudioMetadataItems also has these types of members:

- Methods

Methods

The **ISpatialAudioMetadataItems** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioMetadataItems::GetFrameCount	Gets the total frame count of the ISpatialAudioMetadataItems, which defines valid item offsets.
ISpatialAudioMetadataItems::GetInfo	Gets the total frame count for the ISpatialAudioMetadataItems, which defines valid item offsets.
ISpatialAudioMetadataItems::GetItemCount	The current number of items stored by the ISpatialAudioMetadataItems.
ISpatialAudioMetadataItems::GetMaxItemCount	The maximum number of items allowed by the ISpatialAudioMetadataItems, defined when the object is created.
ISpatialAudioMetadataItems::GetMaxValueBufferLength	The size of the largest command value defined by the metadata format for the ISpatialAudioMetadataItems.

Remarks

Get an instance of this interface by calling ISpatialAudioMetadataClient::ActivateSpatialAudioMetadataItems.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]

Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudiometadata.h

ISpatialAudioMetadataItems::GetFrameCount method

1/11/2020 • 2 minutes to read • Edit Online

Gets the total frame count of the ISpatialAudioMetadataItems, which defines valid item offsets.

Syntax

```
HRESULT GetFrameCount(
  UINT16 *frameCount
);
```

Parameters

`frameCount`

The total frame count.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataItems

ISpatialAudioMetadataItems::GetInfo method

1/11/2020 • 2 minutes to read • Edit Online

Gets the total frame count for the ISpatialAudioMetadataItems, which defines valid item offsets.

Syntax

```
HRESULT GetInfo(
  SpatialAudioMetadataItemsInfo *info
);
```

Parameters

`info`

The total frame count, which defines valid item offsets.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataItems

ISpatialAudioMetadataItems::GetItemCount method

1/11/2020 • 2 minutes to read • Edit Online

The current number of items stored by the ISpatialAudioMetadataItems.

Syntax

```
HRESULT GetItemCount(
  UINT16 *itemCount
);
```

Parameters

`itemCount`

The current number of stored items.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataItems

ISpatialAudioMetadataItems::GetMaxItemCount method

1/11/2020 • 2 minutes to read • Edit Online

The maximum number of items allowed by the ISpatialAudioMetadataItems, defined when the object is created.

Syntax

```
HRESULT GetMaxItemCount(
  UINT16 *maxItemCount
);
```

Parameters

`maxItemCount`

The maximum number of items allowed.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataItems

ISpatialAudioMetadataItems::GetMaxValueBufferLength method

1/11/2020 • 2 minutes to read • Edit Online

The size of the largest command value defined by the metadata format for the ISpatialAudioMetadataItems.

Syntax

```
HRESULT GetMaxValueBufferLength(
  UINT32 *maxValueBufferLength
);
```

Parameters

`maxValueBufferLength`

The size of the largest command value defined by the metadata format.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataItems

ISpatialAudioMetadataItemsBuffer interface

1/23/2020 • 2 minutes to read • Edit Online

Provides methods for attaching buffers to SpatialAudioMetadataItems for in-place storage of data. Get an instance of this object by passing a pointer to the interface into ActivateSpatialAudioMetadataItems. The buffer will be associated with the returned **SpatialAudioMetadataItems**. This interface allows you to attach a buffer and reset its contents to the empty set of metadata items or attach a previously-populated buffer and retain the data stored in the buffer.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioMetadataItemsBuffer** interface inherits from the IUnknown interface. **ISpatialAudioMetadataItemsBuffer** also has these types of members:

- Methods

Methods

The **ISpatialAudioMetadataItemsBuffer** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioMetadataItemsBuffer::AttachToBuffer	Attaches caller-provided memory for storage of ISpatialAudioMetadataItems objects.
ISpatialAudioMetadataItemsBuffer::AttachToPopulatedBuffer	Attaches a previously populated buffer for storage of ISpatialAudioMetadataItems objects. The metadata items already in the buffer are retained.
ISpatialAudioMetadataItemsBuffer::DetachBuffer	Detaches the buffer. Memory can only be attached to a single metadata item at a time.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudiometadata.h

ISpatialAudioMetadataItemsBuffer::AttachToBuffer method

1/11/2020 • 2 minutes to read • Edit Online

Attaches caller-provided memory for storage of ISpatialAudioMetadataItems objects.

Syntax

```
HRESULT AttachToBuffer(
  BYTE   *buffer,
  UINT32 bufferLength
);
```

Parameters

`buffer`

A pointer to memory to use for storage.

`bufferLength`

The length of the supplied buffer. This size must match the length required for the metadata format and maximum metadata item count.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU D_ MD _CL NT_ E_N O_I TEM S_O PEN	The ISpatialAudioMetadataItems has not been opened for copying with a call to Open or the object has been closed for writing with a call to Close.

SPT LAU D_ MD _CL NT_ E_A TTA CH_ FAIL ED_I NTE RN AL_ BUF FER	The ISpatialAudioMetadataItems was created to use a media pipeline internal buffer, so an external buffer can't be attached.
SPT LAU D_ MD _CL NT_ E_B UFF ER_ ALR EAD Y_A TTA CHE D	The supplied buffer has already been attached.
E_IN VAL IDA RG	One of the provided pointers is not valid. The supplied buffer is not large enough to hold the maximum number of metadata items.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataItemsBuffer

ISpatialAudioMetadataItemsBuffer::AttachToPopulatedBuffer method

1/11/2020 • 2 minutes to read • Edit Online

Attaches a previously populated buffer for storage of ISpatialAudioMetadataItems objects. The metadata items already in the buffer are retained.

Syntax

```
HRESULT AttachToPopulatedBuffer(
  BYTE   *buffer,
  UINT32 bufferLength
);
```

Parameters

`buffer`

A pointer to memory to use for storage.

`bufferLength`

The length of the supplied buffer. This size must match the length required for the metadata format and maximum metadata item count.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUD_MD_CLNT_E_NOT_OPEN	The ISpatialAudioMetadataItems has not been opened for copying with a call to Open or the object has been closed for writing with a call to Close.

SPTLAUD_MD_CLNT_E_BUFFER_ALREADY_ATTACHED	The supplied buffer has already been attached.
SPTLAUD_MD_CLNT_E_ATTACH_FAILED_INTERNAL_BUFFER	The ISpatialAudioMetadataItems was created to use a media pipeline internal buffer, so an external buffer can't be attached.
SPTLAUD_MD_CLNT_E_FORMAT_MISMATCH	The supplied populated buffer uses a format that is different from the current format.
E_INVALIDARG	One of the provided pointers is not valid. The supplied buffer is not large enough to hold the maximum number of metadata items. Call GetSpatialAudioMetadataItemsBufferLength to determine the required buffer size.

Requirements

Target Platform	Windows

Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataItemsBuffer

ISpatialAudioMetadataItemsBuffer::DetachBuffer method

1/11/2020 • 2 minutes to read • Edit Online

Detaches the buffer. Memory can only be attached to a single metadata item at a time.

Syntax

```
HRESULT DetachBuffer();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUD_MD_CLNT_E_NO_ITEMS_OPEN	The ISpatialAudioMetadataItems has not been opened for copying with a call to Open or the object has been closed for writing with a call to Close.
SPTLAUD_MD_CLNT_E_ATTACH_FAILED_INTERNAL_BUFFER	The ISpatialAudioMetadataItems was created to use a media pipeline internal buffer which can't be detached.

SPTLAUD_MD_CLNT_E_BUFFER_NOT_ATTACHED	The supplied buffer is not attached.
E_INVALIDARG	One of the provided pointers is not valid. The supplied buffer is not large enough to hold the maximum number of metadata items.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataItemsBuffer

ISpatialAudioMetadataReader interface

1/23/2020 • 2 minutes to read • Edit Online

Provides methods for extracting spatial audio metadata items and item command value pairs from an ISpatialAudioMetadataItems object. The **SpatialAudioMetadataItems** object, which is populated using an ISpatialAudioMetadataWriter or ISpatialAudioMetadataCopier, has a frame count, specified with the *frameCount* parameter to ActivateSpatialAudioMetadataItems, that represents the valid range of metadata item offsets. **ISpatialAudioMetadataReader** enables reading back groups of items within a subrange of the total frame count. The object maintains an internal read position, which is advanced by the number of frames specified when read operation is performed.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioMetadataReader** interface inherits from the IUnknown interface. **ISpatialAudioMetadataReader** also has these types of members:

- Methods

Methods

The **ISpatialAudioMetadataReader** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioMetadataReader::Close	Completes any necessary operations on the SpatialAudioMetadataItems object and releases the object.
ISpatialAudioMetadataReader::Open	Opens an ISpatialAudioMetadataItems object for reading.
ISpatialAudioMetadataReader::ReadNextItem	Gets the number of commands and the sample offset for the metadata item being read.
ISpatialAudioMetadataReader::ReadNextItemCommand	Reads metadata commands and value data for the current item.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudiometadata.h

ISpatialAudioMetadataReader::Close method

1/11/2020 • 2 minutes to read • Edit Online

Completes any necessary operations on the SpatialAudioMetadataItems object and releases the object.

Syntax

```
HRESULT Close();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU D_ MD _CL NT_ E_N O_I TEM S_O PEN	The ISpatialAudioMetadataItems has not been opened for reading with a call to Open or the object has been closed for writing with a call to Close.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataReader

ISpatialAudioMetadataReader::Open method

1/11/2020 • 2 minutes to read • Edit Online

Opens an ISpatialAudioMetadataItems object for reading.

Syntax

```
HRESULT Open(
  ISpatialAudioMetadataItems *metadataItems
);
```

Parameters

`metadataItems`

A pointer to an ISpatialAudioMetadataItems object to be opened for reading

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUD_MD_CLNT_E_ITEMS_ALREADY_OPEN	**Open** has already been called on the supplied ISpatialAudioMetadataItems since the object was created or since the last call to Close.
E_INVALIDARG	The provided pointer is not valid.

Requirements

Target Platform	Windows

Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataReader

ISpatialAudioMetadataReader::ReadNextItem method

1/11/2020 • 2 minutes to read • Edit Online

Gets the number of commands and the sample offset for the metadata item being read.

Syntax

```
HRESULT ReadNextItem(
  UINT8  *commandCount,
  UINT16 *frameOffset
);
```

Parameters

`commandCount`

Receives the number of command/value pairs in the metadata item being read.

`frameOffset`

Gets the frame offset associated with the metadata item being read.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU D_ MD _CL NT_ E_N O_I TEM S_O PEN	The ISpatialAudioMetadataItems has not been opened for reading with a call to Open or the object has been closed for writing with a call to Close.

SPTLAUD_MD_CLNT_E_NO_MORE_ITEMS	There are no more metadata items in the frame range specified in the call to ReadItemCountInFrames.
E_INVALIDARG	One of the provided pointers is not valid.

Remarks

Before calling **ReadNextItem**, you must open the ISpatialAudioMetadataReader for reading by calling Open after the object is created and after Close has been called. You must also call ReadItemCountInFrames before calling **ReadNextItem**.

The ISpatialAudioMetadataReader keeps an internal pointer to the current position within the total range of frames contained by the ISpatialAudioMetadataItems with which the reader is associated. Each call to this method causes the pointer to be advanced by the number of frames specified in the *readFrameCount* parameter.

The process for reading commands and the associated values is recursive. After each call to **ReadItemCountInFrames**, call **ReadNextItem** to get the number of commands in the next item. After every call to **ReadNextItem**, call ReadNextItemCommand to read each command for the item. Repeat this process until the entire frame range of the **ISpatialAudioMetadataItems** has been read.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataReader

ISpatialAudioMetadataReader::ReadNextItemCommand method

1/11/2020 • 2 minutes to read • Edit Online

Reads metadata commands and value data for the current item.

Syntax

```
HRESULT ReadNextItemCommand(
  BYTE   *commandID,
  void   *valueBuffer,
  UINT32 maxValueBufferLength,
  UINT32 *valueBufferLength
);
```

Parameters

`commandID`

Receives the command ID for the current command.

`valueBuffer`

A pointer to a buffer which receives data specific to the command as specified by the metadata format definition. The buffer must be at least *maxValueBufferLength* to ensure all commands can be successfully retrieved.

`maxValueBufferLength`

The maximum size of a command value.

`valueBufferLength`

The size, in bytes, of the data written to the *valueBuffer* parameter.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU D_ MD _CL NT_ E_N O_I TEM S_O PEN	The ISpatialAudioMetadataItems has not been opened for reading with a call to Open or the object has been closed for writing with a call to Close.

E_IN VAL IDA RG	One of the provided pointers is not valid.

Remarks

Before calling **ReadNextItem**, you must open the ISpatialAudioMetadataReader for reading by calling Open after the object is created and after Close has been called. You must also call ReadItemCountInFrames and then call ReadNextItem before calling **ReadNextItem**.

The ISpatialAudioMetadataReader keeps an internal pointer to the current position within the total range of frames contained by the ISpatialAudioMetadataItems with which the reader is associated. Each call to this method causes the pointer to be advanced by the number of frames specified in the *readFrameCount* parameter.

The process for reading commands and the associated values is recursive. After each call to **ReadItemCountInFrames**, call ReadNextItem to get the number of commands in the next item. After every call to **ReadNextItem**, call **ReadNextItemCommand** to read each command for the item. Repeat this process until the entire frame range of the **ISpatialAudioMetadataItems** has been read.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataReader

ISpatialAudioMetadataWriter interface

1/23/2020 • 2 minutes to read • Edit Online

Provides methods for storing spatial audio metadata items positioned within a range of corresponding audio frames. Each metadata item has a zero-based offset position within the specified frame. Each item can contain one or more commands specific to the metadata format ID provided in the SpatialAudioObjectRenderStreamForMetadataActivationParams when the ISpatialAudioMetadataClient was created.
This object does not allocate storage for the metadata it is provided, the caller is expected to manage the allocation of memory used to store the packed data. Multiple metadata items can be placed in the ISpatialAudioMetadataItems object. For each item, call WriteNextItem followed by a call to WriteNextItemCommand.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioMetadataWriter** interface inherits from the IUnknown interface.
ISpatialAudioMetadataWriter also has these types of members:

- Methods

Methods

The **ISpatialAudioMetadataWriter** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioMetadataWriter::Close	Completes any needed operations on the metadata buffer and releases the specified ISpatialAudioMetadataItems object.
ISpatialAudioMetadataWriter::Open	Opens an ISpatialAudioMetadataItems object for writing.
ISpatialAudioMetadataWriter::WriteNextItem	Starts a new metadata item at the specified offset.
ISpatialAudioMetadataWriter::WriteNextItemCommand	Writes metadata commands and value data to the current item.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows

Header	spatialaudiometadata.h

ISpatialAudioMetadataWriter::Close method

1/11/2020 • 2 minutes to read • Edit Online

Completes any needed operations on the metadata buffer and releases the specified ISpatialAudioMetadataItems object.

Syntax

```
HRESULT Close();
```

Parameters

This method has no parameters.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUD_MD_CLNT_E_NOT_OPEN	The supplied ISpatialAudioMetadataItems has not been opened with a call to Open.
SPTLAUD_MD_CLNT_E_NOTHING_WRITTEN	No metadata items have been written to the supplied ISpatialAudioMetadataItems.

SPT **LAU** **D_** **MD** **_CL** **NT_** **E_IT** **EM_** **MU** **ST_** **HA** **VE_** **CO** **MM** **AN** **DS**	No metadata commands have been written to the supplied ISpatialAudioMetadataItems.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataWriter

ISpatialAudioMetadataWriter::Open method

1/11/2020 • 2 minutes to read • Edit Online

Opens an ISpatialAudioMetadataItems object for writing.

Syntax

```
HRESULT Open(
  ISpatialAudioMetadataItems *metadataItems
);
```

Parameters

`metadataItems`

A pointer to an ISpatialAudioMetadataItems object to be opened for writing.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUD_MD_CLNT_E_ITEMS_ALREADY_OPEN	**Open** has already been called on the supplied ISpatialAudioMetadataItems since the object was created or since the last call to Close.
E_INVALIDARG	The provided pointer is not valid.

Requirements

Target Platform	Windows

Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataWriter

ISpatialAudioMetadataWriter::WriteNextItem method

1/11/2020 • 2 minutes to read • Edit Online

Starts a new metadata item at the specified offset.

Syntax

```
HRESULT WriteNextItem(
  UINT16 frameOffset
);
```

Parameters

`frameOffset`

The frame offset of the item within the range specified with the *frameCount* parameter to ActivateSpatialAudioMetadataItems.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPT LAU D_ MD _CL NT_ E_N O_I TEM S_O PEN	The ISpatialAudioMetadataItems has not been opened for writing with a call to Open or the object has been closed for writing with a call to Close.

SPTLAUD_MD_CLNT_E_FRAMEOFFSET_OUT_OF_RANGE	The number of items written in the writing session is greater than the value supplied in the **MaxMetadataItemCount** field in the SpatialAudioObjectRenderStreamForMetadataActivationParam passed into ISpatialAudioClient::ActivateSpatialAudioStream. The *frameCount* value is greater than the value of the *frameCount* parameter to ActivateSpatialAudioMetadataItems and the overflow mode was set to **SpatialAudioMetadataWriterOverflow_Fail**.
E_INVALIDARG	The value of *frameOffset* is not greater than the value provided in the previous call to WriteNextItem within the same writing session.

Remarks

Before calling **WriteNextItem**, you must open the ISpatialAudioMetadataWriter for writing by calling Open after the object is created and after Close has been called. During a writing session demarcated by calls to **Open** and **Close**, the value of the *frameOffset* parameter must be greater than the value in the preceding call.

Within a single writing session, you must not use **WriteNextItem** to write more items than the value supplied in the **MaxMetadataItemCount** field in the SpatialAudioObjectRenderStreamForMetadataActivationParam passed into ISpatialAudioClient::ActivateSpatialAudioStream or an SPTLAUD_MD_CLNT_E_FRAMEOFFSET_OUT_OF_RANGE error will occur.

If the overflow mode is set to **SpatialAudioMetadataWriterOverflow_Fail**, the value of the *frameOffset* parameter must be less than he value of the *frameCount* parameter to ActivateSpatialAudioMetadataItems or an SPTLAUD_MD_CLNT_E_FRAMEOFFSET_OUT_OF_RANGE error will occur.

After calling **WriteNextItem**, call WriteNextItemCommand to write metadata commands and value data for the item.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataWriter

ISpatialAudioMetadataWriter::WriteNextItemCommand method

1/11/2020 • 2 minutes to read • Edit Online

Writes metadata commands and value data to the current item.

Syntax

```
HRESULT WriteNextItemCommand(
  BYTE       commandID,
  const void *valueBuffer,
  UINT32     valueBufferLength
);
```

Parameters

`commandID`

A command supported by the metadata format of the object. The call will fail if the command not defined by metadata format. Each command can only be written once per item.

`valueBuffer`

A pointer to a buffer which stores data specific to the command as specified by the metadata format definition.

`valueBufferLength`

The size, in bytes, of the command data supplied in the *valueBuffer* parameter. The size must match command definition specified by the metadata format or the call will fail.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUD_MD_CLNT_E_NO_ITEMS_OPEN	The ISpatialAudioMetadataItems has not been opened for writing with a call to Open or the object has been closed for writing with a call to Close.

SPT **LAU** **D_** **MD** **_CL** **NT_** **E_N** **O_I** **TEM** **OFF** **SET** **_W** **RIT** **TEN**	WriteNextItem was not called after Open was called and before the call to **WriteNextItemCommand**.

Remarks

You must open the ISpatialAudioMetadataWriter for writing by calling Open, and set the current metadata item offset by calling WriteNextItem before calling **WriteNextItemCommand**.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioMetadataWriter

ISpatialAudioObjectForMetadataCommands interface

1/23/2020 • 2 minutes to read • Edit Online

Used to write metadata commands for spatial audio. Valid commands and value lengths are defined by the metadata format specified in the SpatialAudioObjectRenderStreamForMetadataActivationParams when the ISpatialAudioObjectRenderStreamForMetadata was created.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioObjectForMetadataCommands** interface inherits from ISpatialAudioObjectBase. **ISpatialAudioObjectForMetadataCommands** also has these types of members:

- Methods

Methods

The **ISpatialAudioObjectForMetadataCommands** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObjectForMetadataCommands::WriteNextMetadataCommand	Writes a metadata command to the spatial audio object, each command may only be added once per object per processing cycle.

Remarks

> **Note** Many of the methods provided by this interface are implemented in the inherited ISpatialAudioObjectBase interface.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudiometadata.h (include Spatialaudioclient.h)

See also

ISpatialAudioObjectBase

ISpatialAudioObjectForMetadataCommands::WriteNextMetadataCommand method

1/11/2020 • 2 minutes to read • Edit Online

Writes a metadata command to the spatial audio object, each command may only be added once per object per processing cycle. Valid commands and value lengths are defined by the metadata format specified in the SpatialAudioObjectRenderStreamForMetadataActivationParams when the ISpatialAudioObjectRenderStreamForMetadata was created.

Syntax

```
HRESULT WriteNextMetadataCommand(
  BYTE   commandID,
  void   *valueBuffer,
  UINT32 valueBufferLength
);
```

Parameters

`commandID`

The ID of the metadata command.

`valueBuffer`

The buffer containing the value data for the metadata command.

`valueBufferLength`

The length of the *valueBuffer*.

Return value

If the method succeeds, it returns S_OK.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h (include Spatialaudioclient.h)

See also

ISpatialAudioObjectForMetadataCommands

ISpatialAudioObjectForMetadataItems interface

1/23/2020 • 2 minutes to read • Edit Online

Used to write spatial audio metadata for applications that require multiple metadata items per buffer with frame-accurate placement. The data written via this interface must adhere to the format defined by the metadata format specified in the SpatialAudioObjectRenderStreamForMetadataActivationParams when the ISpatialAudioObjectRenderStreamForMetadata was created.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The **ISpatialAudioObjectForMetadataItems** interface inherits from ISpatialAudioObjectBase. **ISpatialAudioObjectForMetadataItems** also has these types of members:

- Methods

Methods

The **ISpatialAudioObjectForMetadataItems** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObjectForMetadataItems::GetSpatialAudioMetadataItems	Gets a pointer to the ISpatialAudioMetadataItems object which stores metadata items for the ISpatialAudioObjectForMetadataItems.

Remarks

> **Note** Many of the methods provided by this interface are implemented in the inherited ISpatialAudioObjectBase interface.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudiometadata.h (include Spatialaudioclient.h)

See also

ISpatialAudioObjectBase

ISpatialAudioObjectForMetadataItems::GetSpatialAudioMetadataItems method

1/11/2020 • 2 minutes to read • Edit Online

Gets a pointer to the ISpatialAudioMetadataItems object which stores metadata items for the ISpatialAudioObjectForMetadataItems.

Syntax

```
HRESULT GetSpatialAudioMetadataItems(
  ISpatialAudioMetadataItems **metadataItems
);
```

Parameters

`metadataItems`

Receives a pointer to the ISpatialAudioMetadataItems associated with the ISpatialAudioObjectForMetadataItems.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
E_POINTER	The supplied pointer is invalid.

Remarks

The client must free this object when it is no longer being used by calling Release.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h (include Spatialaudioclient.h)

See also

ISpatialAudioObjectForMetadataItems

ISpatialAudioObjectRenderStreamForMetadata interface

1/24/2020 • 2 minutes to read • Edit Online

Provides methods for controlling a spatial audio object render stream for metadata, including starting, stopping, and resetting the stream. Also provides methods for activating new ISpatialAudioObjectForMetadataCommands and ISpatialAudioObjectForMetadataItems instances and notifying the system when you are beginning and ending the process of updating activated spatial audio objects and data.

This interface is a part of Windows Sonic, Microsoft's audio platform for more immersive audio which includes integrated spatial sound on Xbox and Windows.

Inheritance

The ISpatialAudioObjectRenderStreamForMetadata interface inherits from the ISpatialAudioObjectRenderStreamBase interface.

Methods

The **ISpatialAudioObjectRenderStreamForMetadata** interface has these methods.

METHOD	DESCRIPTION
ISpatialAudioObjectRenderStreamForMetadata::ActivateSpatialAudioObjectForMetadataCommands	Activate an ISpatialAudioObjectForMetadataCommands for rendering.
ISpatialAudioObjectRenderStreamForMetadata::ActivateSpatialAudioObjectForMetadataItems	Activate an ISpatialAudioObjectForMetadataItems for rendering.

Remarks

> **Note** Many of the methods provided by this interface are implemented in the inherited ISpatialAudioObjectRenderStreamBase interface.

Requirements

Minimum supported client	Windows 10, version 1703 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioObjectRenderStreamBase

ISpatialAudioObjectRenderStreamForMetadata::ActivateSpatialAudioObjectForMetadataComm method

1/11/2020 • 2 minutes to read • Edit Online

Activate an ISpatialAudioObjectForMetadataCommands for rendering.

Syntax

```
HRESULT ActivateSpatialAudioObjectForMetadataCommands(
  AudioObjectType                       type,
  ISpatialAudioObjectForMetadataCommands **audioObject
);
```

Parameters

`type`

The type of audio object to activate. For dynamic audio objects, this value must be **AudioObjectType_Dynamic**. For static audio objects, specify one of the static audio channel values from the enumeration. Specifying **AudioObjectType_None** will produce an audio object that is not spatialized.

`audioObject`

Receives a pointer to the activated interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUDCLNT_E_NO_MORE_OBJECTS	The maximum number of simultaneous spatial audio objects has been exceeded. Call Release on unused audio objects before attempting to activate additional objects.
SPTLAUDCLNT_E_STATIC_OBJECT_NOT_AVAILABLE	The static channel specified in the *type* parameter was not included in the **StaticObjectTypeMask** field of the SpatialAudioObjectRenderStreamForMetadataActivationParams passed into ISpatialAudioClient::ActivateSpatialAudioStream.
SPTLAUDCLNT_E_OBJECT_ALREADY_ACTIVE	A spatial audio object has already been activated for the static channel specified in the *type* parameter.
E_POINTER	The supplied pointer is invalid.
E_INVALIDARG	The value specified in the *type* parameter is not one of the values defined by the AudioObjectType enumeration.

SPTLAUDCLNT_E_DESTROYED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
SPTLAUDCLNT_E_INTERNAL	An internal error has occurred.
AUDCLNT_E_UNSUPPORTED_FORMAT	The media associated with the spatial audio stream uses an unsupported format.

Remarks

A dynamic ISpatialAudioObjectForMetadataCommands is one that was activated by setting the *type* parameter to the **ActivateSpatialAudioObjectForMetadataCommands** method to **AudioObjectType_Dynamic**. The client has a limit of the maximum number of dynamic spatial audio objects that can be activated at one time. After the limit has been reached, attempting to activate additional audio objects will result in this method returning an SPTLAUDCLNT_E_NO_MORE_OBJECTS error. To avoid this, call Release on each dynamic **ISpatialAudioObjectForMetadataCommands** after it is no longer being used to free up the resource so that it can be reallocated. See ISpatialAudioObjectBase::IsActive and ISpatialAudioObjectBase::SetEndOfStream for more information on the managing the lifetime of spatial audio objects.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioObjectForMetadataItems

ISpatialAudioObjectRenderStreamForMetadata::ActivateSpatialAudioObjectForMetadataItems method

1/11/2020 • 2 minutes to read • Edit Online

Activate an ISpatialAudioObjectForMetadataItems for rendering.

Syntax

```
HRESULT ActivateSpatialAudioObjectForMetadataItems(
  AudioObjectType                    type,
  ISpatialAudioObjectForMetadataItems **audioObject
);
```

Parameters

`type`

The type of audio object to activate. For dynamic audio objects, this value must be **AudioObjectType_Dynamic**. For static audio objects, specify one of the static audio channel values from the enumeration. Specifying **AudioObjectType_None** will produce an audio object that is not spatialized.

`audioObject`

Receives a pointer to the activated interface.

Return value

If the method succeeds, it returns S_OK. If it fails, possible return codes include, but are not limited to, the values shown in the following table.

RETURN CODE	DESCRIPTION
SPTLAUDCLNT_E_NO_MORE_OBJECTS	The maximum number of simultaneous spatial audio objects has been exceeded. Call Release on unused audio objects before attempting to activate additional objects.
SPTLAUDCLNT_E_STATIC_OBJECT_NOT_AVAILABLE	The static channel specified in the *type* parameter was not included in the **StaticObjectTypeMask** field of the SpatialAudioObjectRenderStreamForMetadataActivationParams passed into ISpatialAudioClient::ActivateSpatialAudioStream.
SPTLAUDCLNT_E_OBJECT_ALREADY_ACTIVE	A spatial audio object has already been activated for the static channel specified in the *type* parameter.
E_POINTER	The supplied pointer is invalid.
E_INVALIDARG	The value specified in the *type* parameter is not one of the values defined by the AudioObjectType enumeration.

SPTLAUDCLNT_E_DESTROYED	The ISpatialAudioClient associated with the spatial audio stream has been destroyed.
AUDCLNT_E_DEVICE_INVALIDATED	The audio endpoint device has been unplugged, or the audio hardware or associated hardware resources have been reconfigured, disabled, removed, or otherwise made unavailable for use.
SPTLAUDCLNT_E_INTERNAL	An internal error has occurred.
AUDCLNT_E_UNSUPPORTED_FORMAT	The media associated with the spatial audio stream uses an unsupported format.

Remarks

A dynamic ISpatialAudioObjectForMetadataItems is one that was activated by setting the *type* parameter to the **ActivateSpatialAudioObjectForMetadataItems** method to **AudioObjectType_Dynamic**. The client has a limit of the maximum number of dynamic spatial audio objects that can be activated at one time. After the limit has been reached, attempting to activate additional audio objects will result in this method returning an SPTLAUDCLNT_E_NO_MORE_OBJECTS error. To avoid this, call Release on each dynamic **ISpatialAudioObjectForMetadataItems** after it is no longer being used to free up the resource so that it can be reallocated. See ISpatialAudioObjectBase::IsActive and ISpatialAudioObjectBase::SetEndOfStream for more information on the managing the lifetime of spatial audio objects.

Requirements

Target Platform	Windows
Header	spatialaudiometadata.h

See also

ISpatialAudioObjectForMetadataItems

SpatialAudioMetadataCopyMode enumeration

1/11/2020 • 2 minutes to read • Edit Online

Specifies the copy mode used when calling ISpatialAudioMetadataCopier::CopyMetadataForFrames.

Syntax

```
typedef enum SpatialAudioMetadataCopyMode {
  SpatialAudioMetadataCopy_Overwrite,
  SpatialAudioMetadataCopy_Append,
  SpatialAudioMetadataCopy_AppendMergeWithLast,
  SpatialAudioMetadataCopy_AppendMergeWithFirst
} ;
```

Constants

SpatialAudioMetadataCopy_Overwrite	Creates a direct copy of the number of metadata items specified with the *copyFrameCount* parameter into destination buffer, overwriting any previously existing data.
SpatialAudioMetadataCopy_Append	Performs an append operation which will fail if the resulting ISpatialAudioMetadataItemsBuffer has too many items.
SpatialAudioMetadataCopy_AppendMergeWithLast	Performs an append operation, and if overflow occurs, extra items are merged into last item, adopting last merged item's offset value.
SpatialAudioMetadataCopy_AppendMergeWithFirst	Performs an append operation, and if overflow occurs, extra items are merged, assigning the offset to the offset of the first non-overflow item.

Requirements

Header	spatialaudiometadata.h

SpatialAudioMetadataItemsInfo structure

1/23/2020 • 2 minutes to read • Edit Online

Provides information about an ISpatialAudioMetadataItems object. Get a copy of this structure by calling GetInfo.

Syntax

```
typedef struct SpatialAudioMetadataItemsInfo {
  UINT16 FrameCount;
  UINT16 ItemCount;
  UINT16 MaxItemCount;
  UINT32 MaxValueBufferLength;
} SpatialAudioMetadataItemsInfo;
```

Members

`FrameCount`

The total frame count, which defines valid item offsets.

`ItemCount`

The current number of items stored.

MaxItemCount

The maximum number of items allowed.

MaxValueBufferLength

The size of the largest command value defined by the metadata format.

`MaxItemCount`

`MaxValueBufferLength`

Requirements

Header	spatialaudiometadata.h

SpatialAudioMetadataWriterOverflowMode enumeration

1/11/2020 • 2 minutes to read • Edit Online

Specifies the desired behavior when an ISpatialAudioMetadataWriter attempts to write more items into the metadata buffer than was specified when the client was initialized.

Syntax

```
typedef enum SpatialAudioMetadataWriterOverflowMode {
  SpatialAudioMetadataWriterOverflow_Fail,
  SpatialAudioMetadataWriterOverflow_MergeWithNew,
  SpatialAudioMetadataWriterOverflow_MergeWithLast
} ;
```

Constants

SpatialAudioMetadataWriterOverflow_Fail	The write operation will fail.
SpatialAudioMetadataWriterOverflow_MergeWithNew	The write operation will succeed, the overflow item will be merged with previous item and adopt the frame offset of newest item.
SpatialAudioMetadataWriterOverflow_MergeWithLast	The write operation will succeed, the overflow item will be merged with previous item and keep the existing frame offset.

Requirements

Header	spatialaudiometadata.h

SpatialAudioObjectRenderStreamForMetadataActivationParams structure

1/23/2020 • 2 minutes to read • Edit Online

Represents activation parameters for a spatial audio render stream for metadata. Pass this structure to ISpatialAudioClient::ActivateSpatialAudioStream when activating a stream.

Syntax

```
typedef struct SpatialAudioObjectRenderStreamForMetadataActivationParams {
  const WAVEFORMATEX                    *ObjectFormat;
  AudioObjectType                       StaticObjectTypeMask;
  UINT32                                MinDynamicObjectCount;
  UINT32                                MaxDynamicObjectCount;
  AUDIO_STREAM_CATEGORY                 Category;
  HANDLE                                EventHandle;
  GUID                                  MetadataFormatId;
  UINT16                                MaxMetadataItemCount;
  const PROPVARIANT                     *MetadataActivationParams;
  ISpatialAudioObjectRenderStreamNotify *NotifyObject;
} SpatialAudioObjectRenderStreamForMetadataActivationParams;
```

Members

`ObjectFormat`

Format descriptor for a single spatial audio object. All objects used by the stream must have the same format and the format must be of type WAVEFORMATEX or WAVEFORMATEXTENSIBLE.

`StaticObjectTypeMask`

A bitwise combination of **AudioObjectType** values indicating the set of static spatial audio channels that will be allowed by the activated stream.

`MinDynamicObjectCount`

The minimum number of concurrent dynamic objects. If this number of dynamic audio objects can't be activated simultaneously, ISpatialAudioClient::ActivateSpatialAudioStream will fail with this error **SPTLAUDCLNT_E_NO_MORE_OBJECTS**.

`MaxDynamicObjectCount`

The maximum number of concurrent dynamic objects that can be activated with ISpatialAudioObjectRenderStream.

`Category`

The category of the audio stream and its spatial audio objects.

`EventHandle`

The event that will signal the client to provide more audio data. This handle will be duplicated internally before it is used.

`MetadataFormatId`

The identifier of the metadata format for the currently active spatial rendering engine.

`MaxMetadataItemCount`

The maximum number of metadata items per frame.

`MetadataActivationParams`

Additional activation parameters.

`NotifyObject`

The object that provides notifications for spatial audio clients to respond to changes in the state of an ISpatialAudioObjectRenderStream. This object is used to notify clients that the number of dynamic spatial audio objects that can be activated concurrently is about to change.

Requirements

Header	spatialaudiometadata.h