ASIO4ALL Private API v2.0

07/08/08

Table of Contents

Introduction

The ASIO4ALL driver provides a standard ASIO interface to the host application by means of which the host can control certain aspects of driver behaviour. Other aspects, such as the WDM device configuration, are not covered by the means of control achievable through a standard ASIO interface. With the requirement for more control in mind – including the option for the OEM to provide their own replacement for the original ASIO4ALL GUI – a proprietary interface to the ASIO4ALL core engine was created.

The scope of this interface definition applies to all 2.x versions of the freely available ASIO4ALL software starting with version 2.9, as well as to custom builds for which support for this interface had been specified.

Accessing the interface

Since ASIO under Windows is accessed as a COM in-process server object, the obvious design choice was to use the same type of mechanism for the proprietary interface as well.

A self registering driver DLL supporting the ASIO4ALL private interface will register the latter along with the standard ASIO interface in its `DLLRegisterServer` routine. Application developers can query any existing IASIO for an IA4APRIVATE and, once discovered, use the functionality provided.

Consequently, creating an instance of the ASIO4ALL private interface is as easy as:

```
CoCreateInstance(IID_IASIO,0,1,IID_IA4APRIVATE, (LPVOID*) &ifcInstance);
```

– see the constructor code of the a4aInterface.cpp wrapper class!

The interface CLSID (in CLSID.cpp) is {A26078C5-2840-4726-B427-E60FC8FEE403}.

The actual definition of the interface as well as all associated declarations are contained in ***COMMON\iaa4aprv.h***

Interface member functions

DWORD iA4APrivateGetVersion()

This function may be called at any time and returns the version of the private interface in the BCD coded form `0xMMMMmmmm`, where `MMMM` is the major and `mmmm` the minor revision number. Revision *1.23* would be encoded as `0x00010023`.

void iA4APrivateEnumerate()

Calling this function will cause the driver to re-run audio device enumeration. It is only safe to call this whenever the driver is in the idle state ("initialized" as per the ASIO specification). This function is normally used after a callback has been installed, since the callback would be invoked immediately after enumeration, allowing the application to take over control of WDM device configuration.

void iA4APrivateEnumSetCallback(BOOL (void *), void * pCbData)

Installs a callback the driver will invoke immediately after audio device enumeration. The `pCbData` argument is a pointer to arbitrary data and will be passed to the callback function. Normally, this would point to instance data etc. The callback is invoked always after the driver has finished its internal WDM device (re-)enumeration. Use `iA4APrivateEnumerate()` in order for the driver to immediately re-run enumeration!

If the callback function returns `TRUE`, the driver will re-run enumeration yet once more. Thus, you normally would want to return `FALSE`, in order to avoid endless looping!

Only in the context of the callback function it is safe to access audio device / interface / pin properties! Access to these properties outside of the callback can lead to unpredictable results!

A4AError iA4APrivateGetDeviceProperty (A4ADeviceProperty DeviceProperty, long DeviceIndex, void *PropertyData, long DataSize)

Reads the property `DeviceProperty` of enumerated audio device `DeviceIndex` into user supplied storage `PropertyData` of size `DataSize`.

Can only safely be used in the context of an enumeration callback!

A4AError iA4APrivateSetDeviceProperty (A4ADeviceProperty DeviceProperty, long DeviceIndex, void *PropertyData, long DataSize)

Writes the property `DeviceProperty` of enumerated audio device `DeviceIndex` from user supplied storage `PropertyData` of size `DataSize`.

Can only safely be used in the context of an enumeration callback!

`DeviceProperty` **can be either of the following:**

`kA4A_Device_dwFlags [DataSize=4]`

This is a bit field with the following bit definitions:

A4A_FLAG_HWBUFFER	Enable hardware buffer access method.
A4A_FLAG_FORCESRC	Enforce 44.1<-> 48kHz resampling
A4A_FLAG_PULLMODE	Allow "Pull" mode for WaveRT devices, if supported.
A4A_FLAG_FORCE16	Limit WDM interface bit depth to 16
A4A_FLAG_RUNNING	Indicates object is in the "running" state.
A4A_FLAG_ERROR	Unspecified error indicator.
A4A_FLAG_ENABLED	Enabled in current configuration.

`kA4A_Device_lpszName [DataSize=4]`

Pointer to device name string (as obtained from registry)

kA4A_Device_lpszVendorName [DataSize=4]

Pointer to vendor name string (as obtained from registry)

kA4A_Device_lpszServiceName [DataSize=4]

Pointer to service name string (as obtained from registry)

kA4A_Device_lpszLocationInfo [DataSize=4]

Pointer to location information string (as obtained from registry)

kA4A_Device_lpszDeviceID [DataSize=4]

Pointer to PnP ID string (as obtained from registry)

kA4A_Device_dwBusNumber [DataSize=4]

Bus number (as obtained from registry). Actually a 3 char string with trailing 0.

kA4A_Device_dwKsBuffers [DataSize=4]

Number of Kernel buffers to use.

kA4A_Device_dwIoDelay [DataSize=4]

Delay offset in h/w buffer (ms).

kA4A_Device_dwInputComp [DataSize=4]

Input latency compensation (samples)

kA4A_Device_dwOutputComp [DataSize=4]

Output latency compensation (samples)

kA4A_Device_dwBufferSize [DataSize=4]

ASIO buffer size (samples) for this device.

A4AError iA4APrivateGetInterfaceProperty (A4AInterfaceProperty InterfaceProperty, long DeviceIndex, long InterfaceIndex, void *PropertyData, long DataSize)

Reads the property `InterfaceProperty` of enumerated device interface `InterfaceIndex` on enumerated audio device `DeviceIndex` into user supplied storage `PropertyData` of size `DataSize`.

Can only safely be used in the context of an enumeration callback!

A4AError iA4APrivateSetInterfaceProperty(A4AInterfaceProperty InterfaceProperty, long DeviceIndex, long InterfaceIndex, void *PropertyData, long DataSize)

Writes the property `InterfaceProperty` of enumerated device interface `InterfaceIndex` on enumerated audio device `DeviceIndex` from user supplied storage `PropertyData` of size `DataSize`.

Can only safely be used in the context of an enumeration callback!

`InterfaceProperty` **can be either of the following:**

kA4A_Interface_dwFlags [DataSize=4]

This is a bit field with the following bit definitions:

A4A_FLAG_RTAUDIO	Indicates interface is WaveRT.
A4A_FLAG_RUNNING	Indicates object is in the "running" state.
A4A_FLAG_ERROR	Unspecified error indicator.
A4A_FLAG_ENABLED	Enabled in current configuration.

kA4A_Interface_lpszName [DataSize=4]

Interface name string as obtained from registry.

kA4A_Interface_pSPDIDD [DataSize=4]

Pointer to the SP_DEVICE_INTERFACE_DETAIL_DATA structure for this device interface, see Windows DDK for structure definition!

kA4A_Interface_dwProperties [DataSize=4]

Physical characteristics of the device interface. Can be either of the following:

- A4A_IFPROPERTY_ADC
- A4A_IFPROPERTY_DAC
- A4A_IFPROPERTY_SPDIF

A4AError iA4APrivateGetPinProperty (A4APinProperty PinProperty, long DeviceIndex, long InterfaceIndex, long PinIndex, void *PropertyData, long DataSize)

Reads the property `PinProperty` of audio pin `PinIndex` on enumerated device interface `InterfaceIndex` on enumerated audio device `DeviceIndex` into user supplied storage `PropertyData` of size `DataSize`.

Can only safely be used in the context of an enumeration callback!

A4AError iA4APrivateSetPinProperty (A4APinProperty PinProperty, long DeviceIndex, long InterfaceIndex, long PinIndex, void *PropertyData, long DataSize)

Writes the property `PinProperty` of audio pin `PinIndex` on enumerated device interface `InterfaceIndex` on enumerated audio device `DeviceIndex` from user supplied storage `PropertyData` of size `DataSize`.

Can only safely be used in the context of an enumeration callback!

`PinProperty` **can be either of the following:**

`kA4A_Pin_dwFlags [DataSize=4]`

This is a bit field with the following bit definitions:

`A4A_FLAG_AVAILABLE`	`At least one possible instance remaining.`
`A4A_FLAG_RUNNING`	`Indicates object is in the "running" state.`
`A4A_FLAG_ERROR`	`Unspecified error indicator.`
`A4A_FLAG_ENABLED`	`Enabled in current configuration.`

`kA4A_Pin_dwDataFlow [DataSize=4]`

Possible values:

- `KSPIN_DATAFLOW_IN` (for “output pin”)
- `KSPIN_DATAFLOW_OUT` (for “input pin”)

- see the Windows DDK for definitions!

kA4A_Pin_dwMaxChannels [DataSize=4]

Maximum PCM channels.

kA4A_Pin_dwMaxBits [DataSize=4]

Maximum audio bit depth.

kA4A_Pin_dwMinSR [DataSize=4]

Minimum sample frequency (Hz).

kA4A_Pin_dwMaxSR [DataSize=4]

Maximum sample frequency (Hz).

Error codes

Functions that return the `A4AError` type will return either of the following:

A4A_PRIVATE_NOERROR

The operation completed successfully.

A4A_PRIVATE_ERR_ENUM_REENTRY

An attempt was made to re-enter device enumeration while device enumeration is already in progress. The most likely reason is a callback installed using `iA4APrivateEnumSetCallback()` attempting to invoke `iA4APrivateEnumerate()`. `iA4APrivateEnumerate()` must never be used inside the callback!

A4A_PRIVATE_ERR_DRIVER_NOT_IDLE

An attempt was made to access enumeration data while the ASIO driver is not in the idle state. Stop all audio processing and only then call `iA4APrivateEnumerate()`!

A4A_PRIVATE_ERR_NO_SUCH_OBJECT

The object (device, interface, pin) with the index provided does not exist. You may legally encounter this error when enumerating objects. If this code is returned inside an enumeration loop, this would be an indicator that the end of the device / interface / pin list has been reached.

A4A_PRIVATE_ERR_ILLEGAL_ARGUMENT

An argument has been passed to the interface outside of what it can handle. Possible causes include out of range property identifiers, `NULL` pointer to property data or a data size argument too small for the property requested.

Sample application

An MSVC++ sample application project is provided that demonstrates the use of the API described in this document for the purpose of managing WDM device configuration from within the context of your application.

References

1. **ASIO SDK**, Steinberg Media Technologies GmbH
2. **Windows Driver Kit**, Microsoft, Corp.
3. **Win32 Software Development Kit**, Microsoft, Corp.
4. **ASIO4ALL Instruction Manual**, Michael Tippach