The Cinegy Cinecoder Library

API documentation

Table of Contents

Introduction

Cinegy's revolutionary new technology is **Cinegy Cinecoder**, a suite of powerful MPEG codecs that enables developers to quickly integrate MPEG capability into the desktop and broadcast applications.

Cinegy Cinecoder is an SDK for MPEG-1, MPEG-2 and H.264/MPEG-4 AVC development that offers unmatched quality, a flexible feature set, and the rigorous standards necessary to create professional level products in a short development cycle.

Currently in use with some of the largest names in broadcast television, **Cinegy Cinecoder** is the obvious choice for developers who want high quality, professional software tools for their MPEG development.

Now available through the Cinegy OEM Partner Program, the **Cinegy Cinecoder** SDK provides broadcast and consumer video software developers a full range of MPEG capabilities that can be adapted to any video application.

Cinegy Cinecoder is a result of more than 10 years of intensive codec development with the goal of achieving the highest possible PSNR values with the given bit rate for the high-end industrial applications such video archiving and TV production.

Cinegy Cinecoder is an industrial-level professional compression library, supporting a wide range of media compression standards:

Video - MPEG-1, MPEG-2, VCD, SVCD, DVD, Sony IMX/D10, H.264/AVC, AVC-Intra, Daniel and Daniel2;

Audio - MPEG-1, MP3, AAC, AES-3, LPCM;

Multiplexing - MPEG-1 System Stream, MPEG-2 Program Stream, MPEG-2 Transport Stream, HDV-1,2, XDCAM HD PRO 422, XDCAM EX, AVCHD.

Cinegy Cinecoder provides an extremely high performance. The code is hand-tuned for the top-notch features of the latest CPUs, exploiting all the extended low-level command sets (such as SSE-2, 3, 4) and the advanced parallel computing features of the modern multi-core and multi-CPU systems.

Cinegy Cinecoder is a cross-platforms library, supporting the Windows, Linux and MacOS platforms.

Cinegy Cinecoder is built on top of the COM API concept. This maximizes the reliability of the implementation, minimizing the risk of the resource drain.

The API explicitly provides interfacing with C and C++.

In addition, **Cinegy Cinecoder** provides the interoperability layer for the .NET platform. With the layer it becomes possible to use the library for Visual C#, Visual J# and Visual Basic, exploiting the entire platform’s power, increasing the development comfort and productivity. In the future, the Mono for Linux platform shall be supported, providing an unprecedented flexibility in creation of portable products.

Cinecoder Concepts

Terms and Definitions

All the Cinecoder API classes are logically separated into two main types:

- DataProcessors, the classes for data manipulation, and
- Settings, to set up the data processing parameters.

We will use the term "Cinecoder object" to describe an exemplar of a DataProcessor class.

COM API

As it was mentioned in the Introduction (see page 1), Cinecoder API is based on COM. This means in details:

- All the Cinecoder objects are described in a form of interfaces in IDL files;
- All the Cinecoder interfaces are derived from IUnknown;
- Every Cinecoder interface has its own UUID;
- All the Cinecoder objects are created by a class factory using the UUID.
-

All the Cinecoder objects "own itself" because these are created by a single class factory and shall be destroyed automatically when the number of references drops down to zero (using the IUnknown AddRef/Release methods).

Every Cinecoder object has as a rule several interfaces, available through the IUnknown QueryInterface method.

Thus, only the base platform-independent COM API part is used here. The approach solves effectively various practical problems such as data type information loss when calling from a DLL, or resource drain. In addition, the COM API allows using Cinecoder in the managed languages via the interoperability layer (see hereinafter).

In Windows environment, one can register the Cinecoder3.dll as a COM-server and using it this way if necessary.

Naming Conventions

The Cinecoder interfaces have the "ICC_ " (**I**nterface of **C**ine**C**oder) prefix.

Then the interface name follows (with the capitalized words), for example:

- ICC_MpegVideoEncoder (see page 197)
- ICC_ByteStreamConsumer (see page 12)

The Cinecoder classes have the "CC_" prefix followed by the capitalized class name:

- CC_MpegVideoEncoder
- CC_TransportStreamMultiplexer

The base types and enumerators in Cinecoder have as a rule the "CC_" prefix but sometimes may have "MPG_" or "H264_" depending on the compression standard (see hereinafter).

Only the upper case is used and the underline symbol separates words.

Data Types

Basic Data Types

Cinecoder defines a set of base platform-independent data types to represent the numeric, string, and simple types.

Types

Name	Description
CC_PID (see page 8)	

CC_PID

Syntax

```
typedef WORD CC_PID;
```

Multimedia Data Types

The main purpose of **Cinecoder** is to convert data from one format to another.

Therefore, a number of data type classes is defined.

ByteStream

The key concept of the library is a representation of any encoded data (e.g. elementary coded streams and multiplexed data as well) as a generalized concept of ByteSteam.

ByteStream is a binary stream of data of generally speaking undefined internal structure having, nevertheless, the byte granularity. Some of the bytes can be referred by a 64-bit number called CC_TIME, which corresponds to the presentation time of the media unit encoded somewhere starting from this byte.

MediaData

On the other hand in the Cinecoder data model the typified media data (waveform audio, video frames, subtitles, closed captions etc.) can be located. The decoders consume a ByteStream (see page 9) and produce a MediaData, and vice versa the encoders convert a MediaStream to a ByteStream (see page 9).

Consumers and Producers

To process data of a certain format and to expose the ability of a class to process the format, every class in Cinecoder has a set of dedicated interfaces, the Producers and the Consumers.

ByteStream interfaces

In order to deal with the ByteStream (see page 9) data, the following interfaces are defined in the library.

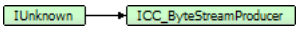
Interfaces

	Name	Description
	ICC_ByteStreamProducer (see page 11)	General interface for transferring chunks of binary data.
	ICC_ByteStreamConsumer (see page 12)	Provides a service to handle the generalized bytestream data. Used as interface of callback-objects which handles the data coming from elementary stream producers such as video or audio encoders. It is also used as interface to the input pin (see page 145) of the ICC_Multiplexer (see page 144) objects.
	ICC_ByteStreamCallback (see page 13)	Provides a service to handle the generalized bytestream data. Used as interface of callback-objects which handles the data coming from elementary stream producers such as video or audio encoders. It is also used as interface to the input pin (see page 145) of the ICC_Multiplexer (see page 144) objects.

ICC_ByteStreamProducer Interface

General interface for transferring chunks of binary data.

Class Hierarchy



Syntax

```
[object, uuid(BA54F9EB-498D-471c-8C01-A88830C6EC01), pointer_default(unique), local]
interface ICC_ByteStreamProducer : IUnknown;
```

Methods

	Name	Description
	GetData (see page 12)	The real number of bytes will be placed here

Methods

GetData

The real number of bytes will be placed here

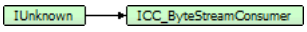
Syntax

```
HRESULT GetData(
    [out,size_is(cbBufSize)] CC_PBYTE pbData,
    [in] CC_UINT cbBufSize,
    [out,retval,defaultvalue(NULL)] CC_UINT * pcbRetSize
);
```

ICC_ByteStreamConsumer Interface

Provides a service to handle the generalized bytestream data. Used as interface of callback-objects which handles the data coming from elementary stream producers such as video or audio encoders. It is also used as interface to the input pin (see page 145) of the ICC_Multiplexer (see page 144) objects.

Class Hierarchy



Syntax

```
[object, uuid(2D8791E2-227D-4898-A64B-525ABAE583DA), pointer_default(unique), local]
interface ICC_ByteStreamConsumer : IUnknown;
```

Methods

	Name	Description
	ProcessData (see page 12)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

Methods

ProcessData

Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc.

In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

Syntax

```
[helpstring("Process a chunk of packed media data")]
HRESULT ProcessData(
    [in, size_is(cbSize)] CC_PCBYTE pbData,
    [in] CC_UINT cbSize,
    [in,defaultvalue(0)] CC_UINT cbOffset,
    [in,defaultvalue(CC_NO_TIME)] CC_TIME pts,
    [out,retval,defaultvalue(NULL)] CC_UINT * pcbProcessed
);
```

Parameters

pbData

Pointer to the buffer containing the elementary data

cbSize

The elementary data size in bytes.

cbOffset

Offset of the buffer's origin from where data will be processed.

pts

(Optional) presentation time of the first access unit commencing in the data.

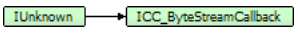
Returns

Returns S_OK if successful or an error value otherwise.

ICC_ByteStreamCallback Interface

Provides a service to handle the generalized bytestream data. Used as interface of callback-objects which handles the data coming from elementary stream producers such as video or audio encoders. It is also used as interface to the input pin (see page 145) of the ICC_Multiplexer (see page 144) objects.

Class Hierarchy



Syntax

```
[object, uuid(3190F038-CC19-405b-B65B-FED981E38F0D), pointer_default(unique), local]
interface ICC_ByteStreamCallback : IUnknown;
```

Methods

	Name	Description
	ProcessData (see page 14)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

Methods

ProcessData

Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc.

In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

Syntax

```
HRESULT ProcessData(
    [in, size_is(cbSize)] CC_PCBYTE pbData,
    [in] CC_AMOUNT cbSize,
    [in,defaultvalue(CC_NO_TIME)] CC_TIME pts,
    [in,defaultvalue(NULL)] IUnknown * pSender
);
```

Parameters

pbData

Pointer to the buffer containing the elementary data.

cbSize

The elementary data size in bytes.

pts

(Optional) presentation time of first access unit commencing in the data.

pSender

The object which generated the data

Returns

Returns S_OK if successful or an error value otherwise.

MediaData Interfaces

To work with the typified media data, there is a set of dedicated interfaces:

• For audio: Consumer + ICC_AudioProducer (see page 121)

• For video: ICC_VideoConsumer (see page 83) + ICC_VideoProducer (see page 85)

The interfaces are described in details in the corresponding chapters on the video and audio formats.

The Hierarchy

Base Interfaces

All the data processors in Cinecoder are inherited from the same base interface ICC_DataProcessor. The base data control methods are defined there.

The concept is very simple; there are only two main methods Init and Done, and the property method IsActive (get_IsActive() in C/C++) to receive the object's state.

TimeBase

The most interesting method in this interface is TimeBase.

Any media data naturally have information about the time when these shall be "published", the so-called presentation time. The TimeBase property sets up the scale for those time intervals, measured in Hertz.

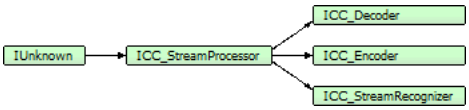
Interfaces

	Name	Description
	ICC_StreamProcessor (see page 17)	Provides the common methods of any cinecoder data processing objects. Used as base interface for all other cinecoder objects.

ICC_StreamProcessor Interface

Provides the common methods of any cinecoder data processing objects. Used as base interface for all other cinecoder objects.

Class Hierarchy



Syntax

```
[object, uuid(f43bfdf8-1ae4-4626-8535-15c49213d208), pointer_default(unique), local]
interface ICC_StreamProcessor : IUnknown;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.

	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

Methods

Done

Stops the current processing.

Syntax

```
HRESULT Done(
    [in] CC_BOOL bFlush,
    [out,retval,defaultvalue(NULL)]CC_BOOL * pbDone
);
```

Parameters

bFlush

CC_FALSE - stop immediately, dismiss the unprocessed data. CC_TRUE - flush the unprocessed data and then stop.

Returns

S_OK = done working.

S_FALSE = still active due to the unprocessed data inside.

And error code in case of error.

Init

Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.

Syntax

```
HRESULT Init(
    [in,defaultvalue(NULL)] ICC_Settings * pSettings
);
```

Parameters

pSettings

The settings.

Returns

Returns S_OK if successful or an error value otherwise.

InitByXml

Initialization of the object by XML profile.

Syntax

```
HRESULT InitByXml(
    [in] CC_STRING strXML
);
```

Parameters

strXML

The settings in XML format.

Properties

BitRate

The bitrate of the generated or processed bytestream

Syntax

```
__property CC_BITRATE * BitRate;
```

DataInfo

The description of the currently generated data. If NULL - data is not ready.

Syntax

```
__property ICC_Settings ** DataInfo;
```

IsActive

The object's state.

Syntax

```
__property CC_BOOL* IsActive;
```

IsDataReady

Indicates that object has prepared data.

Syntax

```
__property CC_BOOL* IsDataReady;
```

OutputCallback

The consumer for the data generated by the object.

Syntax

```
__property IUnknown* OutputCallback;
```

StreamInfo

The description of the stream which is currently being processed. If NULL - object is not active.

Syntax

```
__property ICC_Settings ** StreamInfo;
```

TimeBase

The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

Syntax

```
__property CC_TIMEBASE TimeBase;
```

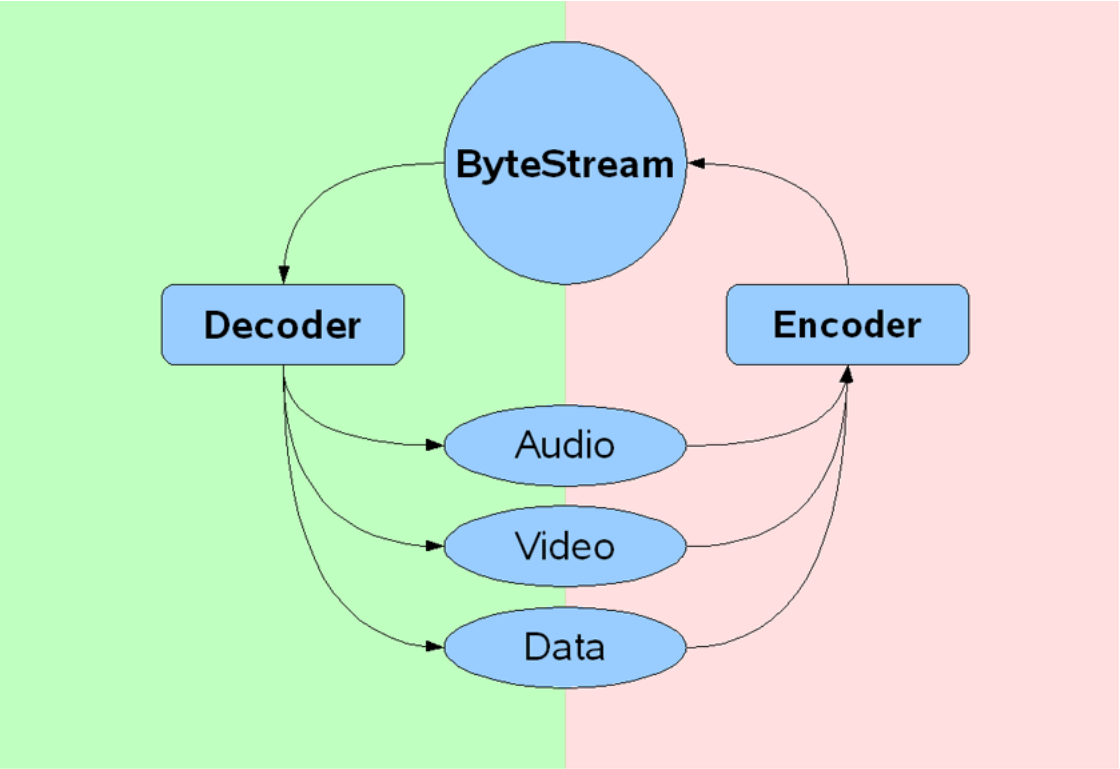
Decoders and Encoders

The Cinecoder objects are divided into two main types in relation to the ByteStream (see page 9); these are:

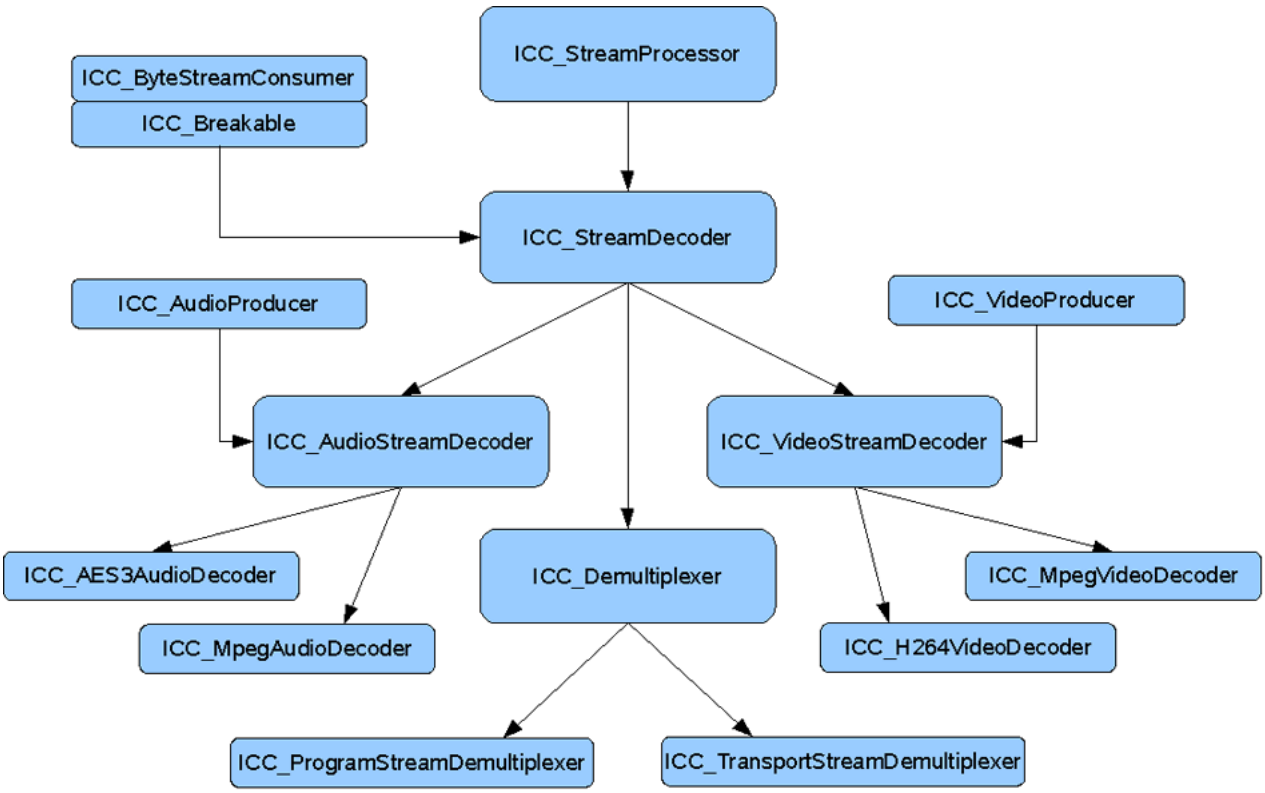
• The decoders (ICC_Decoder (see page 60)), and

• The encoders (ICC_Encoder (see page 63)).

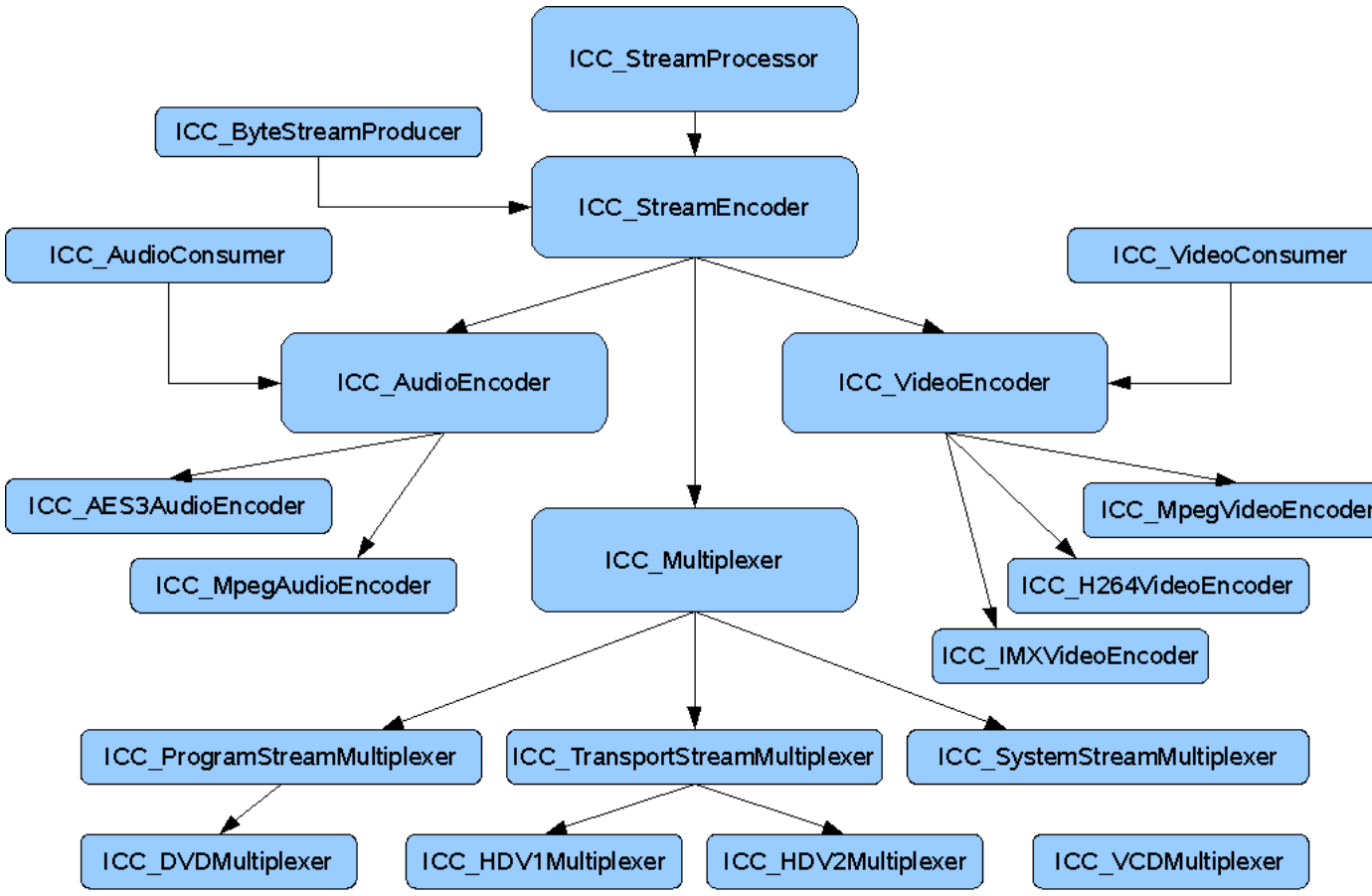
The decoders accept the ByteStream (see page 9) as the input, the encoders produce the ByteStream (see page 9). A simplified scheme is presented in the following illustration:



The Hierarchy of Decoders



The Hierarchy of Encoders



De- and Multiplexers

De-multiplexers and multiplexers are a separate category of the Cinecoder objects. On the one hand, they consume and produce the ByteStream (see page 9) data, but their peculiarity is that there can be several ByteStreams used.

The topic will be described in details in the chapter on the data multiplexing (see page 138).

The Low Latency Mode

Cinecoder is ready to be used in the low latency environment.

Virtually all the objects in the library work with the minimal possible latency.
The latency depends only on the data availability, the compression format traits
and/or the current compression parameters.

The Video Encoder

Due to the peculiarities of video coding, the video compressing process is the most "heavy" part of the compressor in the sense of latency. In order to provide the frame reordering feature the encoder needs to accumulate a required number of the incoming frames; only then the first encoded frame can be produced.

On top of that, to process gracefully a video content break (scene change), a forestall buffer is required.
The size of the buffer also affects the delay. The default size of the forestall buffer is 1 second; as a rule the amount is sufficient to avoid quality degradation in the video content break points.

If the scene detector is disabled, the minimal latency is be determined by the coding parameters.
Generally the delay is equal to the P frames interval:

$$Latency = T_{fr} * max (D_{sd}, GOP_{M}-1) + T_{enc}$$

where:

T_{fr} - duration of the video frame;

D_{sd} - scene detection buffer continuance;

GOP_M - distance between the P-frames;

T_{enc} - time to encode one frame.

The Video Decoder

The video decoder latency is 1 frame (because of the frame reordering), plus the frame decoding time:

$Latency = T_{fr} + T_{dec}$

The Multiplexer

The multiplexer latency is determined by the maximal latency of the input streams.

When the multiplexer has enough data from all the streams, it starts working immediately.

$Latency = max\ (Latency_n)$

Therefore, try to supply the data from all the streams simultaneously.

The De-multiplexer

The de-multiplexer has two types of outputs: the common callback for all the data types and the catch-outputs for a certain stream_id/pid.

In the latter case, there is no delay; the decoded package immediately goes to its catch-output.

In the former case, the first call must be the list of the pids and their stream_ids. Therefore, the latency depends on the moment of receiving the corresponding PAT+PMT tables (for transport streams) or system_header + optional PMT (for program streams).

Latency = Latencyheader

The Audio Encoder/Decoder

For the audio processing, the latency is determined only by the audio frame duration. This depends on the coding standard. In the case of the MPEG1 Layer 1, it is equal to 384 audio samples; for the MPEG1 Layers 2-3, it is 1152 audio samples:

Latency = Na_samples / Faud+ Ta_proc

Faud - audio sampling rate (frequency);

Na_samples - number of audio samples (384 or 1152);

Ta_proc - time to process the audio frame.

Class Factory

The Cinecoder class factory allows creating all the Cinecoder objects (except the factory itself).

In order to create the factory itself, one should use the only export DLL-function Cinecoder_CreateClassFactory:

C++:

```
extern "C"
STDAPI Cinecoder_CreateClassFactory(ICC_ClassFactory**);
```

The definition of the function can be found in the Cinecoder_h.h header file. To link the function, use the Cinecoder.lib library (supplied along with the SDK) in your project.

To access the function from other languages e.g. C# one might use own definition like that:

C#:

```
[DllImport("Cinecoder")]
public static extern
int Cinecoder_CreateClassFactory(out ICC_ClassFactory pFactory);
```

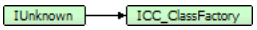
Interfaces

	Name	Description
	ICC_ClassFactory (see page 29)	Provides the methods to create any of cinecoder's objects.

ICC_ClassFactory Interface

Provides the methods to create any of cinecoder's objects.

Class Hierarchy



Syntax

```
[object, uuid(e7d4b675-63d4-43f4-91fa-0af230111d10), helpstring("The Cinecoder(R) ClassFactory
interface"), pointer_default(unique), local]
interface ICC_ClassFactory : IUnknown;
```

Methods

	Name	Description
	AddClassCreator (see page 30)	Add the external (user-defined) class creator to the Cinecoder's list of classes. this method allows users to extend the functionality of the Cinecoder in the run time. This methos suitable for creating the schema containing user-defined classes.
	AssignLicense (see page 30)	Method to assign your own license key for the factory.
	CreateInstance (see page 30)	The main method to create the instance of the specified Cinecoder class.
	CreateInstanceByName (see page 31)	The method to create the instance of the specified Cinecoder class, specified by its name. IUnknown interface will be returned.
	CreateSchema (see page 32)	
	LoadPlugin (see page 32)	The method for loading the external plugins (see ICC_pluginDescr)

Properties

	Name	Description
	DefaultThreadsAffinity (see page 32)	The default affinity mask (see page 56) for all subsequent stream processors with MT capability created by the factory.
	DefaultThreadsPriority (see page 32)	The default priority (see page 58) for all subsequent stream processors with MT capability created by the factory.
	DefaultTimeBase (see page 32)	The default time base for all subsequent stream processors (see page 17) created by the factory.

Methods

AddClassCreator

Add the external (user-defined) class creator to the Cinecoder's list of classes. this method allows users to extend the functionality of the Cinecoder in the run time. This methos suitable for creating the schema containing user-defined classes.

Syntax

```
HRESULT AddClassCreator(
    [in] CLSID clsid,
    [in] CC_STRING pClassName,
    [in] ICC_ClassCreator * pMaker
);
```

Parameters

clsid

CLSID of the class

pClassName

Name of the class

pMaker

The maker object

AssignLicense

Method to assign your own license key for the factory.

Syntax

```
[helpstring("Assigns the license to unlock some Cinecoder's classes")]
HRESULT AssignLicense(
    [in] LPCSTR pCompanyName,
    [in] LPCSTR pLicense
);
```

Returns

S_OK - Success.

MPG_E_INVALID_LICENSE - Incorrect license key and/or company name.

MPG_E_LICENSE_EXPIRED - The period of license is expired.

CreateInstance

The main method to create the instance of the specified Cinecoder class.

Syntax

```
HRESULT CreateInstance(
    [in] REFCLSID rclsid,
    [in] REFIID riid,
    [out,retval,iid_is(riid)]IUnknown** ppu
);
```

Parameters

rclsid

CLSID of the class to be created.

riid

Needed interface of the class.

ppu

Pointer by which the newly created object will be returned.

Returns

Returns S_OK if successful or an error value otherwise.

Remarks

The newly created object comes with RefCount=1 due to executed QueryInterface during creation. So you will need to manually call the Release() method after using the object to avoid the memory leaking or just use the smart COM pointers like CComPtr.

CreateInstanceByName

The method to create the instance of the specified Cinecoder class, specified by its name. IUnknown interface will be returned.

Syntax

```
HRESULT CreateInstanceByName(
    [in] CC_STRING pObjName,
    [out,retval]IUnknown** ppu
);
```

Parameters

pObjName

The enquoted name of a class without 'CC_' prefix (f.e. "MpegVideoEncoder" for CC_MpegVideoEncoder).

ppu

Place when the pointer to IUnknown interface of specified class will be stored.

Returns

S_OK if successful or an error value otherwise.

Remarks

The newly created object comes with RefCount=1. So you will need to manually call the Release() method after using the object to avoid the memory leaking or just use the smart COM pointers like CComPtr.

CreateSchema

Syntax

```
HRESULT CreateSchema(
    [in] CC_STRING strXML,
    [out,retval] ICC_Schema ** p
);
```

LoadPlugin

The method for loading the external plugins (see ICC_pluginDescr)

Syntax

```
HRESULT LoadPlugin(
    [in] CC_STRING pFileName
);
```

Properties

DefaultThreadsAffinity

The default affinity mask (see page 56) for all subsequent stream processors with MT capability created by the factory.

Syntax

```
__property [in] DefaultThreadsAffinity;
```

DefaultThreadsPriority

The default priority (see page 58) for all subsequent stream processors with MT capability created by the factory.

Syntax

```
__property [in] DefaultThreadsPriority;
```

DefaultTimeBase

The default time base for all subsequent stream processors (see page 17) created by the factory.

Syntax

```
__property [in] DefaultTimeBase;
```

Creating objects and Smart Pointers

The Cinecoder object (exemplars of classes) should be created with the factory methods CreateInstance and CreateInstanceByName. The former suits for C/C++; the latter is more for managed languages.

If you are using **Cinecoder** from C++ code, you should use smart pointers when working with the objects. These can be any smart pointers (of the CComPtr type) which “know” about the refcounted methods AddRef/Release of the objects. This prevents memory leakage automatically.

After creation the objects have the ReferenceCount = 1 and because of the fact that the smart pointer is passed to the creation function by reference, it will have the valid reference value when leaving the creator. Leaving a block and destroying the smart pointer shall automatically call the Release method, causing correct destruction of the object.

In the C code, you will need to call the Release method manually every time you finish using the object.

Licensing

Cinecoder SDK is supplied with a corresponding license key. The license defines the set of the supported features and the library objects.

After the class factory is created you need to pass the license key to it. You do this with the AssignLicense method, just passing the data you had received with the **Cinecoder** registration.

Please be attentive when passing the key. The number and the case of the symbols are important.

If you passed a wrong key, the method will return MPG_E_INVALID_LICENSE and MPG_E_LICENSE_EXPIRED if the license has expired.

Cinecoder Schema

Working with multiplexed streams is not as straightforward as with elementary streams.

In the latter case one just create a simple encoder/decoder object in of a required format, but for a multiplex it is far more complicated: one has to create manually a number of required objects and to establish logical links among them, to run and finally to stop and to destroy all of them. For a developer, this is a dreary task indeed.

To achieve more abstract approach to processing such multimedia data streams and to manage the work with the multiplexed data, Cinecoder introduces the concept of **Scheme** as an aggregated object containing simple Cinecoder objects together with established logical links among them.

From a user's perspective the scheme is an XML-structures described below.

Creating a scheme-object

Creating a scheme is different from making simple objects: the scheme type is unknown a priori so an CLSID or a name does not exist for it. Furthermore, a scheme can bear several simple Cinecoder objects and links between them.

Thus for scheme-object creation one should use a dedicated method ICC_ClassFactory::CreateSchema (see page 32) which receives an XML description and created and aggregated scheme-object.

XML scheme format

[xxxx] - optional data

%xxxx% - data names

<schema name="%schema_name%" [TimeBase="%default_time_base%"]>

<!-- example of out-of-place profile(settings) -->

<profile name="%profile_name%">

item0

item1

...

itemN

Please refer to Settings.XML data format to see the items definition.

</profile>

<!-- example of object definition with in-place profile -->

<object name="%object_name%" type="%class_name%">

<profile>

... as shown above

</profile>

</object>

<!-- example of object definition using out-of-place profile definition (profile shall be defined before) -->

<object name="%object_name%" type="%class_name%" profile="%profile_name%"/>

<!-- multiplexer as linkage object -->

<object name="%multiplexer_name%" type="%multiplexer_class_name%">

<input>

<pin object="%object_name_1%"/>

<pin object="%object_name_2%"/>

</input>

</object>

</schema>

Cinecoder API

Base API

This reference section contains descriptions of Base API interfaces, enumerations, and structures.

Interfaces

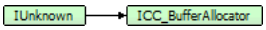
	Name	Description
	ICC_BufferAllocator (see page 42)	Provides the special service for memory allocation.
	ICC_InputBufferControl (see page 44)	Provides the methods to control the input queue.
	ICC_StreamRecovery (see page 45)	Provides the methods for recovering damaged streams (f.e. DVB translation).
	ICC_Breakable (see page 46)	The interface shows that the object can break the normal flow of the data processing and then continue with the new data at the same settings. Usually used when the stream reposition is needed.
	ICC_DataReadyCallback (see page 47)	
	ICC_ElementaryDataInfo (see page 48)	The description of elementary stream data unit
	ICC_ElementaryStreamInfo (see page 51)	
	ICC_InitialTimeCodeProp (see page 53)	Provides access to the InitialTimeCode (see page 53) property.
	ICC_StreamRecognizer (see page 54)	
	ICC_ThreadsAffinityProp (see page 56)	Controls the threads affinity of particular cinecoder object.
	ICC_ThreadsCountProp (see page 57)	Controls the number of threads of particular cinecoder object.
	ICC_ThreadsPriorityProp (see page 58)	Controls the threads priority of particular cinecoder object.
	ICC_TimeBaseProp (see page 59)	Provides access to the TimeBase (see page 59) property.

	ICC_Decoder (see page 60)	General interface for any cinecoder decoders. It is extension of ICC_StreamProcessor (see page 17), including ICC_ByteStreamConsumer (see page 12) and ICC_Breakable (see page 46) interfaces.
	ICC_Encoder (see page 63)	General interface for any cinecoder encoders. It is extension of ICC_StreamProcessor (see page 17), including ICC_ByteStreamProducer (see page 11).
	ICC_ErrorHandler (see page 65)	

ICC_BufferAllocator Interface

Provides the special service for memory allocation.

Class Hierarchy



Syntax

```
[object, uuid(CFB56A68-B3F4-422d-B976-1C265A1D3064), pointer_default(unique), local]
interface ICC_BufferAllocator : IUnknown;
```

Remarks

The interface is used to prevent double memory copy when moving data from source to destination (in the case the destination has an internal buffer for the incoming data). The receiver object can use the interface to allow the source object distributing the memory directly in its buffers to prevent extra memory copying. If the receiver does not have the buffer, the data should be allocated in the source object. In this case, the additional copying is necessary to transfer the data.

Methods

	Name	Description
	Alloc (see page 42)	Method to allocate the specified number of bytes in the internal buffer.

Methods

Alloc

Method to allocate the specified number of bytes in the internal buffer.

Syntax

```
HRESULT Alloc(
    [in] DWORD cbSize,
    [out,retval] BYTE ** pBuffer
);
```

Parameters

cbSize

Number of bytes to allocate.

pBuffer

Pointer to store the pointer to allocated data.

Returns

Returns S_OK if successful or an error value otherwise.

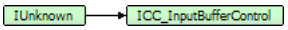
Remarks

If no memory is allocated, NULL will be stored and E_OUTOFMEMORY will be returned. In this case, all generated data will remain in the source (none will be copied). and the destination side will need to copy the data manually, or return the error.

ICC_InputBufferControl Interface

Provides the methods to control the input queue.

Class Hierarchy



Syntax

```
[object, uuid(d96ad003-b4ed-4154-afed-c0f778051415), pointer_default(unique), local]
interface ICC_InputBufferControl : IUnknown;
```

Properties

	Name	Description
	InputBufferSize (see page 44)	The size of the input queue.
R	InputBufferSpace (see page 44)	Retrieves the current input queue free space.
R	InputBufferUsage (see page 44)	Retrieves the current input queue fullness.

Properties

InputBufferSize

The size of the input queue.

Syntax

```
__property [in] InputBufferSize;
```

InputBufferSpace

Retrieves the current input queue free space.

Syntax

```
__property CC_AMOUNT* InputBufferSpace;
```

InputBufferUsage

Retrieves the current input queue fullness.

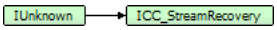
Syntax

```
__property CC_AMOUNT* InputBufferUsage;
```

ICC_StreamRecovery Interface

Provides the methods for recovering damaged streams (f.e. DVB translation).

Class Hierarchy



Syntax

```
[object, uuid(DE7A69AB-560E-4777-BAAA-573F9DEFE7EC), pointer_default(unique), local]
interface ICC_StreamRecovery : IUnknown;
```

Properties

	Name	Description
	MaxPatchDuration (see page 45)	The max duration of breaks which recoverer can handle. Measured in TimeBase units.
	StreamRecoveryMode (see page 45)	The stream recovery mode. See CC_RECOVER_FLAGS.

Properties

MaxPatchDuration

The max duration of breaks which recoverer can handle. Measured in TimeBase units.

Syntax

```
__property CC_TIME MaxPatchDuration;
```

Remarks

Remember that long durations lead to long delays during patch generation.

StreamRecoveryMode

The stream recovery mode. See CC_RECOVER_FLAGS.

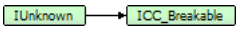
Syntax

```
__property DWORD StreamRecoveryMode;
```

ICC_Breakable Interface

The interface shows that the object can break the normal flow of the data processing and then continue with the new data at the same settings. Usually used when the stream reposition is needed.

Class Hierarchy



Syntax

```
[object, uuid(96218572-1941-41de-854B-0FD5F938BA0E), pointer_default(unique), local]
interface ICC_Breakable : IUnknown;
```

Methods

	Name	Description
	Break (see page 46)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().

Methods

Break

Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().

Syntax

```
HRESULT Break(
    [in] CC_BOOL bFlush,
    [out,retval,defaultvalue(NULL)]CC_BOOL * pbDone
);
```

Parameters

bFlush

CC_FALSE - dismiss the unprocessed data. CC_TRUE - flush the unprocessed data.

Returns

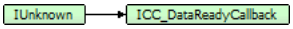
S_OK = done working.

S_FALSE = still active due to the unprocessed data inside (when no callback is assigned).

And error code in case of error.

ICC_DataReadyCallback Interface

Class Hierarchy



Syntax

```
[object, uuid(55813708-a883-4bca-be08-061a546e0d3f), pointer_default(unique), local]
interface ICC_DataReadyCallback : IUnknown;
```

Methods

	Name	Description
	DataReady (see page 47)	

Methods

DataReady

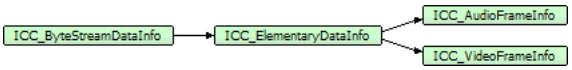
Syntax

```
HRESULT DataReady(
    [in] IUnknown * pDataProducer
);
```

ICC_ElementaryDataInfo Interface

The description of elementary stream data unit

Class Hierarchy



Syntax

```
[object, uuid(D2C8A578-2495-4271-8F99-1DFC469E7B32), pointer_default(unique), local]
interface ICC_ElementaryDataInfo : ICC_ByteStreamDataInfo;
```

Properties

	Name	Description
R	DTS (see page 49)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 49), except the coded video with B-frames.
R	Duration (see page 49)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
R	NumSamples (see page 49)	Number of samples of the elementary data. In case of coded video frames, there is usually 1 or 2 sample(s) (1 frame or 2 fields). In case of coded audio, there are the number of audio samples, encoded into audio frame.
R	PresentationDelta (see page 49)	The presentation delta, in samples, of the elementary data. It is actual for data which was reordered during encoding process (f.e. coded video with B-frames).
R	PTS (see page 49)	The Presentation Data Stamp (PTS) of the elementary data. The PTS based on CC_TIMEBASE, specified for object which generates the elementary data.
R	SampleOffset (see page 49)	The frame's first sample order number.
R	SequenceEntryFlag (see page 50)	The sequence entry point flag. In the case of mpeg video, it means that SEQUENCE_HEADER presents in the elementary data. This flag is used to signal the multiplexers to generate the entry point (like System Header) at this elementary data.

Properties

DTS

The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 49), except the coded video with B-frames.

Syntax

```
__property CC_TIME* DTS;
```

Duration

Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.

Syntax

```
__property CC_TIME* Duration;
```

NumSamples

Number of samples of the elementary data. In case of coded video frames, there is usually 1 or 2 sample(s) (1 frame or 2 fields). In case of coded audio, there are the number of audio samples, encoded into audio frame.

Syntax

```
__property CC_UINT* NumSamples;
```

PresentationDelta

The presentation delta, in samples, of the elementary data. It is actual for data which was reordered during encoding process (f.e. coded video with B-frames).

Syntax

```
__property CC_INT* PresentationDelta;
```

PTS

The Presentation Data Stamp (PTS) of the elementary data. The PTS based on CC_TIMEBASE, specified for object which generates the elementary data.

Syntax

```
__property CC_TIME* PTS;
```

SampleOffset

The frame's first sample order number.

Syntax

```
__property CC_OFFSET* SampleOffset;
```

SequenceEntryFlag

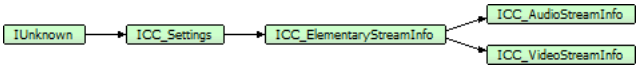
The sequence entry point flag. In the case of mpeg video, it means that SEQUENCE_HEADER presents in the elementary data. This flag is used to signal the multiplexers to generate the entry point (like System Header) at this elementary data.

Syntax

```
__property CC_BOOL* SequenceEntryFlag;
```

ICC_ElementaryStreamInfo Interface

Class Hierarchy



Syntax

```
[object, uuid(03AF145E-6633-4cbd-B6CF-286473E55860), pointer_default(unique), local]
interface ICC_ElementaryStreamInfo : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ElementaryStreamInfo Interface

	Name	Description
R	BitRate (see page 52)	The bitrate (max) of the elementary stream.
R	FrameRate (see page 52)	The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.
R	StreamType (see page 52)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

Properties

BitRate

The bitrate (max) of the elementary stream.

Syntax

```
__property CC_BITRATE * BitRate;
```

FrameRate

The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.

Syntax

```
__property CC_FRAME_RATE * FrameRate;
```

StreamType

The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

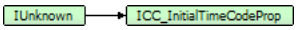
Syntax

```
__property CC_ELEMENTARY_STREAM_TYPE * StreamType;
```

ICC_InitialTimeCodeProp Interface

Provides access to the InitialTimeCode (see page 53) property.

Class Hierarchy



Syntax

```
[object, uuid(020CC64E-0BCD-4d5b-B68A-E210716F2D9E), pointer_default(unique), local]
interface ICC_InitialTimeCodeProp : IUnknown;
```

Properties

	Name	Description
	InitialTimeCode (see page 53)	

Properties

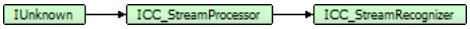
InitialTimeCode

Syntax

```
__property CC_TIMECODE InitialTimeCode;
```

ICC_StreamRecognizer Interface

Class Hierarchy



Syntax

```
[object, uuid(00007777-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_StreamRecognizer : ICC_StreamProcessor;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_StreamRecognizer Interface

	Name	Description
	GetStreamInfo (see page 55)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

Methods

GetStreamInfo

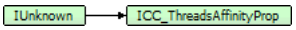
Syntax

```
HRESULT GetStreamInfo(
    [out,retval]ICC_MultiplexedStreamInfo ** pStreamInfo
);
```

ICC_ThreadsAffinityProp Interface

Controls the threads affinity of particular cinecoder object.

Class Hierarchy



Syntax

```
[object, uuid(452854e9-2033-46a3-a3cd-a95aa1f466c1), pointer_default(unique), local]
interface ICC_ThreadsAffinityProp : IUnknown;
```

Properties

	Name	Description
	ThreadsAffinity (see page 56)	The affinity mask for object's threads. 0 = default (current process) affinity mask.

Properties

ThreadsAffinity

The affinity mask for object's threads. 0 = default (current process) affinity mask.

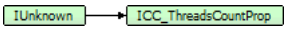
Syntax

```
__property [in] ThreadsAffinity;
```

ICC_ThreadsCountProp Interface

Controls the number of threads of particular cinecoder object.

Class Hierarchy



Syntax

```
[object, uuid(8f73a09e-3419-42f9-b6e5-28425443fc5d), pointer_default(unique), local]
interface ICC_ThreadsCountProp : IUnknown;
```

Properties

	Name	Description
	ThreadsCount (see page 57)	A maximal number of threads which an object can use. 0 = automatic.

Properties

ThreadsCount

A maximal number of threads which an object can use. 0 = automatic.

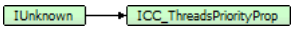
Syntax

```
__property [in] ThreadsCount;
```

ICC_ThreadsPriorityProp Interface

Controls the threads priority of particular cinecoder object.

Class Hierarchy



Syntax

```
[object, uuid(635a623f-f077-4083-963d-6aa66268b0c8), pointer_default(unique), local]
interface ICC_ThreadsPriorityProp : IUnknown;
```

Properties

	Name	Description
	ThreadsPriority (see page 58)	The priority for object's threads.

Properties

ThreadsPriority

The priority for object's threads.

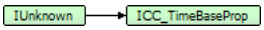
Syntax

```
__property [in] ThreadsPriority;
```

ICC_TimeBaseProp Interface

Provides access to the TimeBase (see page 59) property.

Class Hierarchy



Syntax

```
[object, uuid(BF96E276-449C-47ec-BFA0-0BACB3447F1D), pointer_default(unique), local]
interface ICC_TimeBaseProp : IUnknown;
```

Properties

	Name	Description
	TimeBase (see page 59)	

Properties

TimeBase

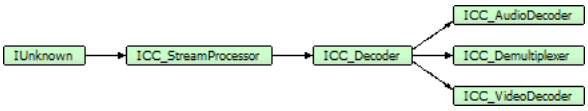
Syntax

```
__property [in] TimeBase;
```

ICC_Decoder Interface

General interface for any cinecoder decoders. It is extension of ICC_StreamProcessor (see page 17), including ICC_ByteStreamConsumer (see page 12) and ICC_Breakable (see page 46) interfaces.

Class Hierarchy



Syntax

```
[object, uuid(5E8AF531-FB3F-4345-A6E9-50DF4A4030EB), pointer_default(unique), local]
interface ICC_Decoder : ICC_StreamProcessor;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.

R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

Methods

Break

Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().

Syntax

```
HRESULT Break(
    [in] CC_BOOL bFlush,
    [out,retval,defaultvalue(NULL)]CC_BOOL * pbDone
);
```

Parameters

bFlush

CC_FALSE - dismiss the unprocessed data. CC_TRUE - flush the unprocessed data.

Returns

S_OK = done working.

S_FALSE = still active due to the unprocessed data inside (when no callback is assigned).

And error code in case of error.

ProcessData

Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc.

In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

Syntax

```
HRESULT ProcessData(
    [in, size_is(cbSize)] CC_PCBYTE pbData,
    [in] CC_UINT cbSize,
    [in,defaultvalue(0)] CC_UINT cbOffset,
    [in,defaultvalue(CC_NO_TIME)] CC_TIME pts,
    [out,retval,defaultvalue(NULL)] CC_UINT * pcbProcessed
);
```

Parameters

pbData

Pointer to the buffer containing the elementary data

cbSize

The elementary data size in bytes.

cbOffset

Offset of the buffer's origin from where data will be processed.

pts

(Optional) presentation time of the first access unit commencing in the data.

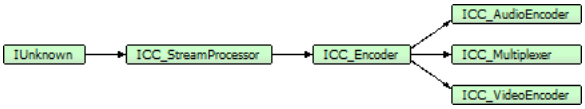
Returns

Returns S_OK if successful or an error value otherwise.

ICC_Encoder Interface

General interface for any cinecoder encoders. It is extension of ICC_StreamProcessor (see page 17), including ICC_ByteStreamProducer (see page 11).

Class Hierarchy



Syntax

```
[object, uuid(c5352932-d890-4631-a41b-054ef932d58f), pointer_default(unique), local]
interface ICC_Encoder : ICC_StreamProcessor;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.

	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

Methods

GetData

The real number of bytes will be placed here

Syntax

```
HRESULT GetData(
    [out,size_is(cbBufSize)] CC_PBYTE pbData,
    [in] CC_UINT cbBufSize,
    [out,retval,defaultvalue(NULL)] CC_UINT * pcbRetSize
);
```

Properties

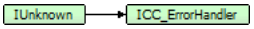
DataSize

Syntax

```
__property CC_UINT* DataSize;
```

ICC_ErrorHandler Interface

Class Hierarchy



Syntax

```
[object, uuid(1fb875da-bdf4-4de9-b591-a3dc77d370c9), pointer_default(unique), local]
interface ICC_ErrorHandler : IUnknown;
```

Methods

	Name	Description
	ErrorHandlerFunc (see page 65)	

Methods

ErrorHandlerFunc

Syntax

```
HRESULT ErrorHandlerFunc(
    HRESULT ErrCode,
    LPCSTR ErrDescription,
    LPCSTR pFileName,
    INT LineNo
);
```

Settings API

All the objects of the **Cinecoder** library must be set up in order to fulfill the required task.

Some of the objects simply will not work if not set up. For example, the encoders need some vital parameters, such as the frame rate and size for video, the sample frequency and the number of channels for audio, to be set up. Not knowing the parameters the activity makes no sense.

Some parameters can be undefined, meaning the class defaults the values according to the other parameters and source data.

Many of the objects can work even without initial setup, such as decoders, and de-multiplexers can draw the information from the source stream, self-tuning on-the-fly. Though sometimes these should be set up as well -- choosing certain data format or selecting the environment.

NB: you can read in more details about the settings of every class in the corresponding chapter.

As **Cinecoder** library consists of a hierarchy of classes operating at a high level abstraction, the settings also should have some means of abstraction in order to implement some generalized actions, unified for all the parameters, such as serialization, copying, defaulting and verification.

For that purpose a special class was introduced in **Cinecoder** implementing the ICC_Settings (see page 72) interface. The interface supports the basic functionality for serialization and the like. The class is the base for all the other specific settings.

Interfaces

	Name	Description
	ICC_Settings (see page 72)	This interface is a base for any info or settings objects. Provides the methods to clone/serialize or manually check or clear the particular value by its name.

Settings elements

Considering a setting object as the analogy of a structure and its every "field" is represented as a pair of get/set methods (or get for read-only fields).

For example (IDL, the definition of the read/write CC_BITRATE Bitrate field):

```
[propget] HRESULT BitRate([out,retval] CC_BITRATE *p);
[propput] HRESULT BitRate([in]         CC_BITRATE  v);
```

Another sample (IDL, the definition of the read-only CC_AAC_FORMAT (see page 316) Format filed):

```
[propget] HRESULT Format([out,retval] CC_AAC_FORMAT *p);
```

Every field can contain a value or do not contain any. During creation of an object, all (or majority) of its fields do not matter.

The absence of a value in a field can be tested with the result of the get method. If the value was set, S_OK is returned and S_FALSE otherwise.

To check the presence of the set up value, you can pass NULL to the method (C++):

```
if(S_FALSE == pSettings->get_Bitrate(NULL))
{  // has no bitrate value
}

CC_BITRATE bitrate;
if(S_OK == pSettings->get_Bitrate(&bitrate))
{  // has bitrate value set
}
```

In the case of C# we don't have the luxury to check the set up value. To test the field from C# one can use the ICC_Settings::Assigned (see page 73) method:

```
if(Settings.Assigned("Bitrate"))
{// has bitrate value set
}
```

Frankly, in this case we get a runtime check (instead of the static check in C++), and in the case of an error, the HRESULT E_INVALIDARG exception will be generated.

In return the handling of the object itself is far easier:

```
CC_BITRATE bitrate = Settings.Bitrate;
```

And in the case of reading an uninitialized field, usually a special value will be returned, indicating the absence of the set up value. For example, in the following code:

C#:

```
CC_BITRATE_MODE br_mode = Settings.RateMode;
```

C++:

```
CC_BITRATE_MODE br_mode;
pSettings->get_RateMode(&br_mode);
```

In the case of the uninitialized field, the CC_BITRATE_MODE_UNKNOWN will be returned (see CC_BITRATE_MODE description).

General settings properties

Assigned()

Through the base interface ICC_Settings (see page 72) one can check the set up value in the corresponding field. The method also allows one to check if at least one field has a value. To so that, provide the "*" as the field name.

Clear()

In addition to value check, one can clear it up using the Clear() method:

```
Settings.Clear("Bitrate");
```

or

```
Settings.Clear("*")
```

to clear all the fileds.

Serialization

The serialization of the Settings-objects in **Cinecoder** is provided via the XML-formatted data.

In order to save and load the data in XML format, use the following methods of the ICC_Settings (see page 72) interface:

```
[propget] HRESULT XML([out,retval] CC_STRING *pstrXml) and
[propput] HRESULT XML([in]         CC_STRING  strXml);
```

XML data format

[xxxx] - optional data

%xxxx% - data names

```
<settings [name="%settings_name%"] [object="%class_name%"]>
  ...
  <!-- values of simple data types or enums -->
  <%item_name% [type="%typename%"] value="%value%"/>
  <BitRate type="CC_BITRATE" value="3000000"/>
  <RateMode type="CC_BITRATE_MODE" value="CBR"/>  ...

  <!-- structures -->
  <%item_name% [type="%typename%"] %field0%="%value0%" ...
%fieldN%="%valueN%"/>
  <FrameRate type="CC_FRAME_RATE" num="25" denom="1"/>
  <FrameSize type="CC_SIZE" cx="352" cy="288"/>  ...
</settings>
```

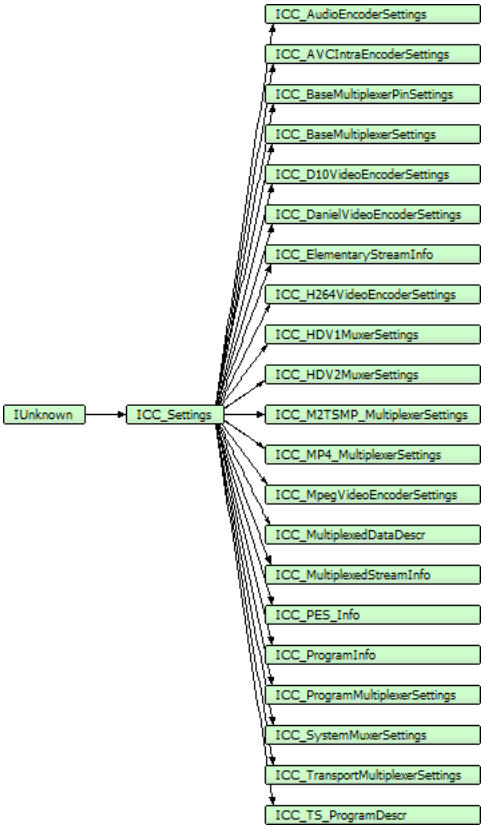
An example:

```
  <settings name="AAC_LC_192000x2_384">
    <Profile     value="LC"   />
    <SampleRate  value="192000"/>
    <NumChannels value="1"    />
    <BitRate     value="384000"/>
  </settings>
```

ICC_Settings Interface

This interface is a base for any info or settings objects. Provides the methods to clone/serialize or manually check or clear the particular value by its name.

Class Hierarchy



Syntax

```
[object, uuid(AA8AE4DE-938F-4eb3-AD44-363464D10A5D), pointer_default(unique), local]
interface ICC_Settings : IUnknown;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

Methods

Assigned

Tests the specified value (or any of values if specified "*") assigned.

Syntax

```
HRESULT Assigned(
    [in] LPCSTR strVarName,
    [out,retval,defaultvalue(NULL)]CC_BOOL*
);
```

Clear

Marks the specified value (or all values if specified "*") as not assigned.

Syntax

```
HRESULT Clear(
    [in] LPCSTR strVarName
);
```

Properties

XML

Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

Syntax

```
__property CC_STRING XML;
```

Schemas API

The process of the basic stream operating is more or less straight forward - a simple object-encoder or decoder of the necessary format is created and configured. However, the case with multiplexer streams becomes a little bit more complicated. Here creating necessary objects and linking them together as well as managing their work are performed manually.

To automate the multimedia data processing and centralize the work with multiplexer data, **Cinecoder** introduces a Schema – an aggregate object containing Cinecoder objects with the already established links.

From the user's point of view, a schema is formed as a specially structured in XML.

Interfaces

	Name	Description
	ICC_Schema (see page 78)	Interface to handle the aggregate objects. See The Schema chapter for details.
	ICC_ClassCreator (see page 81)	

Creation of Schema objects

Schemas are created in a different way than other simple objects due to the following features:

• A schema type is not predefined, so there are no CLSID or name assigned for it;

• During the initialization, more than one simple Cinecoder object is created and links between them are established.

Therefore, to create objects-schemas, the object factory Cinecoder uses a special ICC_ClassFactory::CreateSchema (see page 32) method. A schema description in XML format is passed to the method, and an aggregate object-schema is received on the output.

Schema definition

[xxxx] - optional data

%xxxx% - data names

```
<schema name="%schema_name%" [TimeBase="%default_time_base%"]>

  <!-- example of out-of-place profile(settings) -->
  <profile name="%profile_name%">
    item0
    item1
    ...
    itemN

    Please refer Settings.XML data format to see the items definition.

  </profile>

  <!-- example of object definition with in-place profile  -->
  <object name="%object_name%" type="%class_name%">
    <profile>
      ... as shown above
    </profile>
  </object>

  <!-- example of object definition using out-of-place profile
definition (profile shall be defined before) -->
  <object name="%object_name%" type="%class_name%"
profile="%profile_name%"/>

  <!-- multiplexer as linkage object -->
  <object name="%multiplexer_name%" type="%multiplexer_class_name%">
    <input>
      <pin object="%object_name_1%"/>
      <pin object="%object_name_2%"/>
    </input>
  </object>
</schema>
```

The Schema Sample

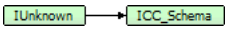
This sample represents the definition of schema for generation iPod-compatible MP4 multiplexer.

```
<schema name="PAL_iPod_352x288@512" TimeBase="90000">

  <!-- This is example of out-place settings -->
  <profile name="AAC_LC_44100x2_96">
    <Profile             value="LC"   />
    <SampleRate          value="44100"/>
    <NumChannels         value="1"    />
    <BitRate             value="96000"/>
  </profile>

  <object name="VideoEncoder" type="H264VideoEncoder">
    <profile>
      <Profile           value="baseline"/>
      <Level             value="30"/>
      <BitRate           value="800000"/>
      <AvgBitRate        value="180000"/>
      <RateMode          value="vbr"/>
      <FrameRate         num="25" denom="1"/>
      <FrameSize         cx="352" cy="288"/>
      <AspectRatio       num="4" denom="3"/>
      <IDR_Period        value="1"/>
      <GOP               N="25" M="1"/>
      <ChromaFormat      value="4:2:0"/>
      <InterlaceType     value="1"/>
      <PictureStructure  value="0"/>
      <MB_Struct         value="0"/>
      <EntropyCodingMode value="cavlc"/>
    </profile>
  </object>

  <object name="AudioEncoder" type="AAC_AudioEncoder"
profile="AAC_LC_44100x2_96">
  </object>

  <object name="Multiplexer" type="MP4_Multiplexer">
    <input>
      <pin object="VideoEncoder"/>
      <pin object="AudioEncoder"/>
    </input>
  </object>
</schema>
```

ICC_Schema Interface

Interface to handle the aggregate objects. See The Schema chapter for details.

Class Hierarchy



Syntax

```
[object, uuid(48b6acc6-3d65-41f4-8005-cf19058afc7e), pointer_default(unique), local]
interface ICC_Schema : IUnknown;
```

Methods

	Name	Description
	Done (see page 78)	Done all of internal objects in the safe order.
	FindObjectByInterface (see page 79)	The method searches for nth object with specified interface.
	FindObjectByName (see page 79)	Finds the subobject by its name in the XML.
	GetObject (see page 79)	Returns the nth subobject of scheme
	GetObjectCountByInterface (see page 80)	Returns the number of scheme subobjects with specified interface

Properties

	Name	Description
R	Name (see page 80)	Returns the name specified by "name" tag. If no name tag specified - returns NULL and S_FALSE
R	ObjectCount (see page 80)	Returns the number of objects in the scheme

Methods

Done

Done all of internal objects in the safe order.

Syntax

```
HRESULT Done(
    [in] CC_BOOL bFlush,
    [out,retval,defaultvalue(NULL)]CC_BOOL * pbDone
);
```

Parameters

bFlush

CC_FALSE - stop immediately, dismiss the unprocessed data. CC_TRUE - flush the unprocessed data, then stop.

FindObjectByInterface

The method searches for nth object with specified interface.

Syntax

```
HRESULT FindObjectByInterface(
    [in] IID ObjIID,
    [in] CC_UINT OrderNumber,
    [out, retval] IUnknown ** ppObj
);
```

Returns

S_OK or S_FALSE if n'th object with specified interface is not found

Notes

The objects ordered as declared in the XML.

FindObjectByName

Finds the subobject by its name in the XML.

Syntax

```
HRESULT FindObjectByName(
    [in] CC_STRING pObjName,
    [out, retval] IUnknown ** ppObj
);
```

Returns

S_OK or S_FALSE if nothing associated with the name is found.

GetObject

Returns the nth subobject of scheme

Syntax

```
HRESULT GetObject(
    [in] CC_UINT OrderNumber,
    [out, retval] IUnknown ** ppObj
);
```

Returns

S_OK or E_INVALIDARG

Notes

The objects ordered as declared in the XML.

GetObjectCountByInterface

Returns the number of scheme subobjects with specified interface

Syntax

```
HRESULT GetObjectCountByInterface(
    [in] IID ObjIID,
    [out,retval] CC_UINT * ObjNumber
);
```

Returns

S_OK

Properties

Name

Returns the name specified by "name" tag. If no name tag specified - returns NULL and S_FALSE

Syntax

```
__property CC_STRING * Name;
```

ObjectCount

Returns the number of objects in the scheme

Syntax

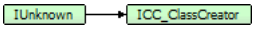
```
__property CC_UINT * ObjectCount;
```

Returns

S_OK

ICC_ClassCreator Interface

Class Hierarchy



Syntax

```
[object, uuid(4735fb44-54bc-478a-8ffa-3ac6cb320a1a), pointer_default(unique), local]
interface ICC_ClassCreator : IUnknown;
```

Methods

	Name	Description
	CreateInstance (see page 81)	The method to create the instance of the specified class.

Methods

CreateInstance

The method to create the instance of the specified class.

Syntax

```
HRESULT CreateInstance(
    [in] REFCLSID rclsid,
    [out,retval] IUnknown** ppu
);
```

Parameters

rclsid

CLSID of the class

ppu

Pointer to pointer for created object

Returns

Returns S_OK if successful or an error value otherwise.

Remarks

The newly created object comes with RefCount=1 due to executed QueryInterface during creation. So you will need to manually call the Release() method after using the object to avoid the memory leaking or just use the smart COM pointers like CComPtr.

Video API

This reference section contains descriptions of Video API interfaces and structures.

Interfaces

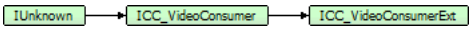
Interfaces

	Name	Description
	ICC_VideoConsumer (see page 83)	Provides the methods specific for the generic video data consumer.
	ICC_VideoProducer (see page 85)	Provides the methods specific for the general video data producers.
	ICC_VideoConsumerExt (see page 88)	Provides the methods specific for the generic video data consumer.
	ICC_VideoDecoder (see page 89)	The default and main interface to control the instance of CC_MpegVideoDecoder class.
	ICC_VideoEncoder (see page 92)	The default and main interface to control the instance of CC_MpegVideoEncoder class.
	ICC_VideoFrameInfo (see page 96)	Interface provides the common description of the video frame.
	ICC_VideoFrameQualityInfo (see page 98)	Provides methods for retrieving the video PSNR (see page 99) (quality measure) information of the video frame.
	ICC_VideoStreamInfo (see page 99)	Interface represents the common video stream description.

ICC_VideoConsumer Interface

Provides the methods specific for the generic video data consumer.

Class Hierarchy



Syntax

```
[object, uuid(00002003-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_VideoConsumer : IUnknown;
```

Methods

	Name	Description
	AddFrame (see page 84)	Add another frame to the encoder's input queue.
	GetStride (see page 84)	

	GetVideoFrameInfo (see page 85)	Get the description of the current video frame.
	GetVideoStreamInfo (see page 85)	Get the description of the video stream which is being decoded.
	IsFormatSupported (see page 85)	

Methods

AddFrame

Add another frame to the encoder's input queue.

Syntax

```
HRESULT AddFrame(
    [in] CC_COLOR_FMT Format,
    [in, size_is(cbSize)] const BYTE * pData,
    [in] DWORD cbSize,
    [in,defaultvalue(0)] CC_INT stride,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

Parameters

Format

The color format (see page 103) of the frame.

pData

Pointer to the frame data.

cbSize

The size of the frame in bytes.

stride

The stride of a single video row, measured in bytes. A negative stride value means that order of rows is reversed (the first row is on the bottom of the frame). If stride is set to zero, the real value is derived from the video consumer settings and frame's color format (see page 103).

pResult

The result of the AddFrame operation.

Returns

Returns S_OK if successful or an error value otherwise.

GetStride

Syntax

```
HRESULT GetStride(
    [in] CC_COLOR_FMT fmt,
```

```
    [out,retval] DWORD * pNumBytes
);
```

GetVideoFrameInfo

Get the description of the current video frame.

Syntax

```
HRESULT GetVideoFrameInfo(
    [out,retval] ICC_VideoFrameInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

GetVideoStreamInfo

Get the description of the video stream which is being decoded.

Syntax

```
HRESULT GetVideoStreamInfo(
    [out,retval] ICC_VideoStreamInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

IsFormatSupported

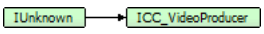
Syntax

```
HRESULT IsFormatSupported(
    [in] CC_COLOR_FMT fmt,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

ICC_VideoProducer Interface

Provides the methods specific for the general video data producers.

Class Hierarchy



Syntax

```
[object, uuid(00002002-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_VideoProducer : IUnknown;
```

Methods

	Name	Description
	GetFrame (see page 86)	Retrieve the current video frame.
	GetStride (see page 87)	

	GetVideoFrameInfo (see page 87)	Get the description of the current video frame.
	GetVideoStreamInfo (see page 87)	Get the description of the video stream which is being decoded.
	IsFormatSupported (see page 87)	
	IsFrameAvailable (see page 87)	Check if the video frame is ready.

Methods

GetFrame

Retrieve the current video frame.

Syntax

```
HRESULT GetFrame(
    [in] CC_COLOR_FMT Format,
    [out,size_is(cbSize)] BYTE * pbVideoData,
    [in] DWORD cbSize,
    [in,defaultvalue(0)] INT stride,
    [out,retval,defaultvalue(NULL)]DWORD * pcbRetSize
);
```

Parameters

Format

The color format (see page 103) of the frame.

pbVideoData

Buffer where the frame data will be stored.

cbSize

The size of the buffer in bytes.

stride

The stride of a single video row, measured in bytes. A negative stride value means reversed row order (the first row is on the bottom of the frame). If stride=0, the real value is derived from the frame size and frame's color format (see page 103).

pcbRetSize

The resulting frame size in bytes.

Returns

Returns S_OK if successful or an error value otherwise.

GetStride

Syntax

```
HRESULT GetStride(
    [in] CC_COLOR_FMT fmt,
    [out,retval] DWORD * pNumBytes
);
```

GetVideoFrameInfo

Get the description of the current video frame.

Syntax

```
HRESULT GetVideoFrameInfo(
    [out,retval] ICC_VideoFrameInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

GetVideoStreamInfo

Get the description of the video stream which is being decoded.

Syntax

```
HRESULT GetVideoStreamInfo(
    [out,retval] ICC_VideoStreamInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

IsFormatSupported

Syntax

```
HRESULT IsFormatSupported(
    [in] CC_COLOR_FMT fmt,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

IsFrameAvailable

Check if the video frame is ready.

Syntax

```
HRESULT IsFrameAvailable(
    [out,retval,defaultvalue(NULL)]CC_BOOL * pResult
);
```

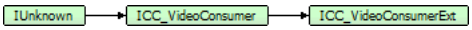
Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

ICC_VideoConsumerExt Interface

Provides the methods specific for the generic video data consumer.

Class Hierarchy



Syntax

```
[object, uuid(00002004-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_VideoConsumerExt : ICC_VideoConsumer;
```

Methods

	Name	Description
	AddFrame (see page 84)	Add another frame to the encoder's input queue.
	GetStride (see page 84)	
	GetVideoFrameInfo (see page 85)	Get the description of the current video frame.
	GetVideoStreamInfo (see page 85)	Get the description of the video stream which is being decoded.
	IsFormatSupported (see page 85)	

ICC_VideoConsumerExt Interface

	Name	Description
	AddScaleFrame (see page 88)	Add a frame with different size to the processing queue.
	IsScaleAvailable (see page 89)	

Methods

AddScaleFrame

Add a frame with different size to the processing queue.

Syntax

```
HRESULT AddScaleFrame(
    [in, size_is(cbSize)] const BYTE * pData,
    [in] DWORD cbSize,
    [in] CC_VIDEO_FRAME_DESCR * pParams,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

Parameters

pData

Pointer to the frame data.

cbSize

The size of the frame in bytes.

pParams

Frame description structure.

pResult

The result of the AddFrame operation.

Returns

Returns S_OK if successful or an error value otherwise.

IsScaleAvailable

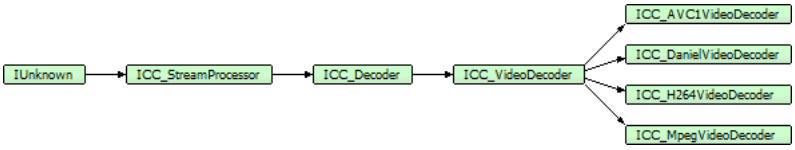
Syntax

```
HRESULT IsScaleAvailable(
    [in] CC_VIDEO_FRAME_DESCR * pParams,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

ICC_VideoDecoder Interface

The default and main interface to control the instance of CC_MpegVideoDecoder class.

Class Hierarchy



Syntax

```
[object, uuid(00002005-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_VideoDecoder : ICC_Decoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.

	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_VideoDecoder Interface

	Name	Description
	GetFrame (see page 91)	Retrieve the current video frame.
	GetStride (see page 91)	
	GetVideoFrameInfo (see page 91)	Get the description of current video frame.
	GetVideoStreamInfo (see page 92)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 92)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

Methods

GetFrame

Retrieve the current video frame.

Syntax

```
HRESULT GetFrame(
    [in] CC_COLOR_FMT Format,
    [out,size_is(cbSize)] BYTE * pbVideoData,
    [in] DWORD cbSize,
    [in,defaultvalue(0)] INT stride,
    [out,retval,defaultvalue(NULL)]DWORD * pcbRetSize
);
```

Parameters

Format

The color format (see page 103) of the frame.

pbVideoData

Buffer for storing the video data

cbSize

The size of the buffer in bytes.

stride

The stride of a single video row, measured in bytes. A negative stride value means reversed row order (the first row is on the bottom of the frame). If stride=0, the real value is derived from the frame size and frame's color format (see page 103).

pcbRetSize

The resulting frame size in bytes.

Returns

Returns S_OK if successful or an error value otherwise.

GetStride

Syntax

```
HRESULT GetStride(
    [in] CC_COLOR_FMT fmt,
    [out,retval] DWORD * pNumBytes
);
```

GetVideoFrameInfo

Get the description of current video frame.

Syntax

```
HRESULT GetVideoFrameInfo(
```

```
    [out,retval] ICC_VideoFrameInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

GetVideoStreamInfo

Get the description of video stream which is being decoded.

Syntax

```
HRESULT GetVideoStreamInfo(
    [out,retval] ICC_VideoStreamInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

IsFormatSupported

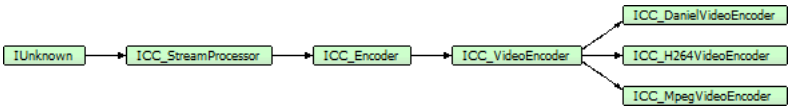
Syntax

```
HRESULT IsFormatSupported(
    [in] CC_COLOR_FMT fmt,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

ICC_VideoEncoder Interface

The default and main interface to control the instance of CC_MpegVideoEncoder class.

Class Hierarchy



Syntax

```
[object, uuid(00002006-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_VideoEncoder : ICC_Encoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_VideoEncoder Interface

	Name	Description
	AddFrame (see page 94)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 94)	Add a frame with different size to the processing queue.
	GetStride (see page 95)	
	GetVideoFrameInfo (see page 95)	Get the description of current video frame.
	GetVideoStreamInfo (see page 95)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 95)	
	IsScaleAvailable (see page 96)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

Methods

AddFrame

Add another frame to the encoder's input queue.

Syntax

```
HRESULT AddFrame(
    [in] CC_COLOR_FMT Format,
    [in, size_is(cbSize)] const BYTE * pData,
    [in] DWORD cbSize,
    [in,defaultvalue(0)] INT stride,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

Parameters

Format

The color format (see page 103) of the frame.

pData

Pointer to the frame data.

cbSize

The size of the frame in bytes.

stride

The stride of a single video row, measured in bytes. A negative stride value means that order of rows is reversed (the first row is on the bottom of the frame) If stride is set to zero, the real value is derived from the video consumer settings and frame's color format (see page 103).

pResult

The result of the AddFrame operation.

Returns

Returns S_OK if successful or an error value otherwise.

AddScaleFrame

Add a frame with different size to the processing queue.

Syntax

```
HRESULT AddScaleFrame(
    [in, size_is(cbSize)] const BYTE * pData,
    [in] DWORD cbSize,
    [in] CC_ADD_VIDEO_FRAME_PARAMS * pParams,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

Parameters

pData

Pointer to the frame data.

cbSize

The size of the frame in bytes.

pParams

Frame description structure.

pResult

The result of the AddFrame operation.

Returns

Returns S_OK if successful or an error value otherwise.

GetStride

Syntax

```
HRESULT GetStride(
    [in] CC_COLOR_FMT fmt,
    [out,retval] DWORD * pNumBytes
);
```

GetVideoFrameInfo

Get the description of current video frame.

Syntax

```
HRESULT GetVideoFrameInfo(
    [out,retval] ICC_VideoFrameInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

GetVideoStreamInfo

Get the description of video stream which is being decoded.

Syntax

```
HRESULT GetVideoStreamInfo(
    [out,retval] ICC_VideoStreamInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

IsFormatSupported

Syntax

```
HRESULT IsFormatSupported(
    [in] CC_COLOR_FMT fmt,
```

```
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

IsScaleAvailable

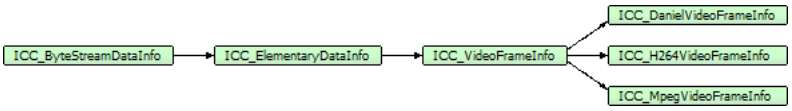
Syntax

```
HRESULT IsScaleAvailable(
    [in] CC_ADD_VIDEO_FRAME_PARAMS * pParams,
    [out,retval,defaultvalue(NULL)] CC_BOOL*
);
```

ICC_VideoFrameInfo Interface

Interface provides the common description of the video frame.

Class Hierarchy



Syntax

```
[object, uuid(00002203-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_VideoFrameInfo : ICC_ElementaryDataInfo;
```

Properties

	Name	Description
R	DTS (see page 49)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 49), except the coded video with B-frames.
R	Duration (see page 49)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
R	NumSamples (see page 49)	Number of samples of the elementary data. In case of coded video frames, there is usually 1 or 2 sample(s) (1 frame or 2 fields). In case of coded audio, there are the number of audio samples, encoded into audio frame.
R	PresentationDelta (see page 49)	The presentation delta, in samples, of the elementary data. It is actual for data which was reordered during encoding process (f.e. coded video with B-frames).
R	PTS (see page 49)	The Presentation Data Stamp (PTS) of the elementary data. The PTS based on CC_TIMEBASE, specified for object which generates the elementary data.
R	SampleOffset (see page 49)	The frame's first sample order number.

	SequenceEntryFlag (see page 50)	The sequence entry point flag. In the case of mpeg video, it means that SEQUENCE_HEADER presents in the elementary data. This flag is used to signal the multiplexers to generate the entry point (like System Header) at this elementary data.

ICC_VideoFrameInfo Interface

	Name	Description
	CodingNumber (see page 97)	The number of video frame in the coding order. Zero-based.
	Flags (see page 97)	Various flags of the coded video frame. Value of this field dependend of the video stream type.
	FrameType (see page 97)	The frame coding type (see page 107).
	InterlaceType (see page 98)	The field order of video frame.
	Number (see page 98)	The number of video frame in native (display) order. zero-based.
	PictStruct (see page 98)	The picture structure of the MPEG video frame.
	TimeCode (see page 98)	The timecode of video frame.

Properties

CodingNumber

The number of video frame in the coding order. Zero-based.

Syntax

```
__property CC_UINT * CodingNumber;
```

Flags

Various flags of the coded video frame. Value of this field dependend of the video stream type.

Syntax

```
__property DWORD * Flags;
```

FrameType

The frame coding type (see page 107).

Syntax

```
__property CC_FRAME_TYPE * FrameType;
```

InterlaceType

The field order of video frame.

Syntax

```
__property CC_INTERLACE_TYPE * InterlaceType;
```

Number

The number of video frame in native (display) order. zero-based.

Syntax

```
__property CC_UINT * Number;
```

PictStruct

The picture structure of the MPEG video frame.

Syntax

```
__property CC_PICTURE_STRUCTURE* PictStruct;
```

TimeCode

The timecode of video frame.

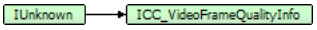
Syntax

```
__property CC_TIMECODE * TimeCode;
```

ICC_VideoFrameQualityInfo Interface

Provides methods for retrieving the video PSNR (see page 99) (quality measure) information of the video frame.

Class Hierarchy



Syntax

```
[object, uuid(00002205-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_VideoFrameQualityInfo : IUnknown;
```

Properties

	Name	Description
R	AvgQuantScale (see page 99)	
R	PSNR (see page 99)	Retrieves the PSNR value of the video frame.

Properties

AvgQuantScale

Syntax

```
__property CC_FLOAT * AvgQuantScale;
```

PSNR

Retrieves the PSNR value of the video frame.

Syntax

```
__property CC_VIDEO_QUALITY_INFO * PSNR;
```

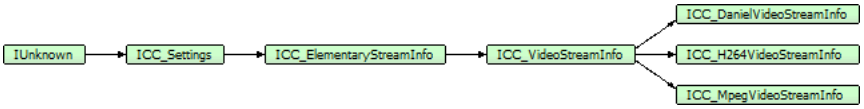
Returns

Returns S_OK if successful or an error value otherwise.

ICC_VideoStreamInfo Interface

Interface represents the common video stream description.

Class Hierarchy



Syntax

```
[object, uuid(00002201-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_VideoStreamInfo : ICC_ElementaryStreamInfo;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ElementaryStreamInfo Interface

	Name	Description
	BitRate (see page 52)	The bitrate (max) of the elementary stream.
	FrameRate (see page 52)	The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.
	StreamType (see page 52)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

ICC_VideoStreamInfo Interface

	Name	Description
	AspectRatio (see page 100)	The aspect ratio cx:cy.
	FrameSize (see page 100)	The size in pixels of video frame.
	ProgressiveSequence (see page 100)	If TRUE - all frames in the stream coded without fields (progressive).

Properties

AspectRatio

The aspect ratio cx:cy.

Syntax

```
__property CC_RATIONAL * AspectRatio;
```

FrameSize

The size in pixels of video frame.

Syntax

```
__property CC_SIZE * FrameSize;
```

ProgressiveSequence

If TRUE - all frames in the stream coded without fields (progressive).

Syntax

```
__property CC_BOOL * ProgressiveSequence;
```

Structures

Enumerations

	Name	Description
	CC_CHROMA_FORMAT (see page 102)	The MPEG-2 video chrominance format. (see iso/iec 13818-2 6.3.5 Table 6-5 for details).
	CC_COLOR_FMT (see page 103)	The color format of video frames.
	CC_FRAME_TYPE (see page 107)	Coded video frame type.
	CC_COLOUR_PRIMARIES (see page 108)	The chromaticity coordinates of the source primaries (see iso/iec 13818-2 6.3.6 Table 6-7 for details).
	CC_GOP_PATTERN (see page 110)	Symbolic names for CC_GOP_DESCR::N field.
	CC_INTERLACE_TYPE (see page 111)	The field order of video frame.
	CC_MATRIX_COEFFICIENTS (see page 111)	The matrix coefficients used in deriving luminance and chrominance signals from the green, blue, and red primaries (see iso/iec 13818-2 6.3.6 Table 6-9 for details).
	CC_MB_STRUCTURE (see page 112)	The coded macroblock structure.
	CC_PICTURE_STRUCTURE (see page 113)	The coding structure of coded video frame.
	CC_TRANSFER_CHARACTERISTICS (see page 113)	The opto-electronic transfer characteristic of the source picture (see iso/iec 13818-2 6.3.6 Table 6-8 for details).
	CC_VIDEO_FORMAT (see page 115)	The representation of the pictures before being coded in accordance with iso/iec 13818-2 (MPEG2) specs. (see iso/iec 13818-2 6.3.6 Table 6-6 for details).
	CC_AFF_TYPE (see page 116)	

Structures

	Name	Description
	CC_COLOUR_DESCRIPTION (see page 108)	The aggregate representation of CC_COLOUR_PRIMARIES (see page 108), CC_TRANSFER_CHARACTERISTICS (see page 113) ans CC_MATRIX_COEFFICIENTS (see page 111).
	CC_GOP_DESCR (see page 110)	Group of Pictures (GOP) description.
	CC_QUANT_DESCR (see page 116)	Quantization parameter for defining mquant parameter in VBR modes.
	CC_ADD_VIDEO_FRAME_PARAMS (see page 117)	

CC_CHROMA_FORMAT

The MPEG-2 video chrominance format. (see iso/iec 13818-2 6.3.5 Table 6-5 for details).

Syntax

```
[v1_enum]
enum CC_CHROMA_FORMAT {
  CC_CHROMA_FORMAT_UNKNOWN = -1,
  CC_CHROMA_400,
  CC_CHROMA_420,
  CC_CHROMA_422,
  CC_CHROMA_444,
  CC_CHROMA_RGB,
  CC_CHROMA_ALPHA = 8,
  CC_CHROMA_4000 = CC_CHROMA_400,
  CC_CHROMA_4004 = CC_CHROMA_400|CC_CHROMA_ALPHA,
  CC_CHROMA_4200 = CC_CHROMA_420,
  CC_CHROMA_4204 = CC_CHROMA_420|CC_CHROMA_ALPHA,
  CC_CHROMA_4220 = CC_CHROMA_422,
  CC_CHROMA_4224 = CC_CHROMA_422|CC_CHROMA_ALPHA,
  CC_CHROMA_4440 = CC_CHROMA_444,
  CC_CHROMA_4444 = CC_CHROMA_444|CC_CHROMA_ALPHA,
  CC_CHROMA_RGBA = CC_CHROMA_RGB|CC_CHROMA_ALPHA
};
```

Members

CC_CHROMA_FORMAT_UNKNOWN

Unknown chroma format.

CC_CHROMA_400

Grayscale

CC_CHROMA_420

YUV 4:2:0 Half vertical, half horizontal color resolution.

CC_CHROMA_422

YUV 4:2:2 Full vertical, half horizontal color resolution.

CC_CHROMA_444

YUV 4:4:4 Full color resolution.

CC_CHROMA_RGB

RGB, full color resolution

CC_CHROMA_ALPHA

Chroma has alpha channel

CC_COLOR_FMT

The color format of video frames.

Syntax

```
[v1_enum]
enum CC_COLOR_FMT {
  CCF_UNKNOWN = 0x00000000,
  CCF_RGB32 = 0x00000001,
  CCF_RGB24 = 0x00000002,
  CCF_BGR32 = 0x00000011,
  CCF_BGR24 = 0x00000012,
  CCF_RGB30 = 0x00000202,
  CCF_RGBA = CCF_RGB32,
  CCF_RGB = CCF_RGB24,
  CCF_BGRA = CCF_BGR32,
  CCF_BGR = CCF_BGR24,
  CCF_RGB64 = 0x00000801,
  CCF_RGB48 = 0x00000802,
  CCF_BGR64 = 0x00000811,
  CCF_BGR48 = 0x00000812,
  CCF_ADOBE_RGBA_16u = 0x00000901,
  CCF_ADOBE_BGRA_16u = 0x00000911,
  CCF_RGBA_32F = 0x00000A01,
  CCF_BGRA_32F = 0x00000A11,
  CCF_RGBA_16BIT = CCF_RGB64,
  CCF_RGB_16BIT = CCF_RGB48,
  CCF_BGRA_16BIT = CCF_BGR64,
  CCF_BGR_16BIT = CCF_BGR48,
  CCF_UYVY = 0x00000031,
  CCF_YUY2 = 0x00000032,
  CCF_YUV400 = 0x00000040,
  CCF_YUV420 = 0x00000041,
  CCF_YUV422 = 0x00000042,
  CCF_YUV444 = 0x00000043,
  CCF_YUV400_10BIT = 0x00000240,
  CCF_YUV420_10BIT = 0x00000241,
  CCF_YUV422_10BIT = 0x00000242,
  CCF_YUV444_10BIT = 0x00000243,
  CCF_YUV400_12BIT = 0x00000440,
  CCF_YUV420_12BIT = 0x00000441,
  CCF_YUV422_12BIT = 0x00000442,
  CCF_YUV444_12BIT = 0x00000443,
  CCF_YUV400_14BIT = 0x00000640,
  CCF_YUV420_14BIT = 0x00000641,
  CCF_YUV422_14BIT = 0x00000642,
  CCF_YUV444_14BIT = 0x00000643,
  CCF_YUV400_16BIT = 0x00000840,
  CCF_YUV420_16BIT = 0x00000841,
```

```
  CCF_YUV422_16BIT = 0x00000842,
  CCF_YUV444_16BIT = 0x00000843,
  CCF_UYVY_10BIT = 0x00000231,
  CCF_YUY2_10BIT = 0x00000232,
  CCF_UYVY_12BIT = 0x00000431,
  CCF_YUY2_12BIT = 0x00000432,
  CCF_UYVY_14BIT = 0x00000631,
  CCF_YUY2_14BIT = 0x00000632,
  CCF_UYVY_16BIT = 0x00000831,
  CCF_YUY2_16BIT = 0x00000832,
  CCF_UYVY_32F = 0x00000A31,
  CCF_YUY2_32F = 0x00000A32,
  CCF_YV12 = CCF_YUV420,
  CCF_YV16 = CCF_YUV422,
  CCF_NV12 = 0x00000112,
  CCF_NV12_10BIT = 0x00000113,
  CCF_V210 = 0x00000210,
  CCF_SC10 = 0x00000211,
  CCF_Y216 = CCF_YUY2_16BIT,
  CCF_Y8 = 0x00000008,
  CCF_YUY2_4x4K = 0x00004032,
  CCF_V210_4x4K = 0x00004210
};
```

Members

CCF_UNKNOWN

Unknown color format.

CCF_RGB32

The color data in the RGBA format, 4 bytes per pixel.

CCF_RGB24

The color data in the RGB format, 3 bytes per pixel.

CCF_BGR32

The color data in the BGRA format, 4 bytes per pixel.

CCF_BGR24

The color data in the BGR format, 3 bytes per pixel.

CCF_RGB30

10-bit RGB format, packed into 4-byte dword

CCF_RGB64

The color data in the 16-bit RGBA format, 4 words per pixel.

CCF_RGB48

The color data in the 16-bit RGB format, 3 words per pixel.

CCF_BGR64

The color data in the 16-bit BGRA format, 4 words per pixel.

CCF_BGR48

The color data in the 16-bit BGR format, 3 words per pixel.

CCF_ADOBE_RGBA_16u

The color data in the 16-bit RGBA format, 4 words per pixel, black at 0 to white at 32768

CCF_ADOBE_BGRA_16u

The color data in the 16-bit BGRA format, 4 words per pixel, black at 0 to white at 32768

CCF_RGBA_32F

The color data in the 32-bit float RGBA format, 4 floats per pixel, black at 0 to white at 1

CCF_BGRA_32F

The color data in the 32-bit float BGRA format, 4 floats per pixel, black at 0 to white at 1

CCF_UYVY

The packed YUV 4:2:2 color data, pixel order {u,y0,v,y1}, 4 bytes per 2 pixels

CCF_YUY2

The packed YUV 4:2:2 color data, pixel order {y0,u,y1,v}, 4 bytes per 2 pixels

CCF_YUV400

Planar YUV 4:2:0

CCF_YUV420

Planar YUV 4:2:0

CCF_YUV422

Planar YUV 4:2:2

CCF_YUV444

Planar YUV 4:4:4

CCF_YUV400_10BIT

Planar YUV 4:0:0 (10 of 16 bit data, LSB).

CCF_YUV420_10BIT

Planar YUV 4:2:0 (10 of 16 bit data, LSB).

CCF_YUV422_10BIT

Planar YUV 4:2:2 (10 of 16 bit data, LSB).

CCF_YUV444_10BIT

Planar YUV 4:4:4 (10 of 16 bit data, LSB).

CCF_YUV400_12BIT

Planar YUV 4:0:0 (12 of 16 bit data, LSB).

CCF_YUV420_12BIT

Planar YUV 4:2:0 (12 of 16 bit data, LSB).

CCF_YUV422_12BIT

Planar YUV 4:2:2 (12 of 16 bit data, LSB).

CCF_YUV444_12BIT

Planar YUV 4:4:4 (12 of 16 bit data, LSB).

CCF_YUV400_14BIT

Planar YUV 4:0:0 (14 of 16 bit data, LSB).

CCF_YUV420_14BIT

Planar YUV 4:2:0 (14 of 16 bit data, LSB).

CCF_YUV422_14BIT

Planar YUV 4:2:2 (14 of 16 bit data, LSB).

CCF_YUV444_14BIT

Planar YUV 4:4:4 (14 of 16 bit data, LSB).

CCF_YUV400_16BIT

Planar YUV 4:0:0 16 bit data (LSB).

CCF_YUV420_16BIT

Planar YUV 4:2:0 16 bit data (LSB).

CCF_YUV422_16BIT

Planar YUV 4:2:2 16 bit data (LSB).

CCF_YUV444_16BIT

Planar YUV 4:4:4 16 bit data (LSB).

CCF_UYVY_10BIT

The packed YUV 4:2:2 10-bit color data in order {u,y0,v,y1} 16-bits each (LSB).

CCF_YUY2_10BIT

The packed YUV 4:2:2 10-bit color data in order {y0,u,y1,v} 16-bits each (LSB).

CCF_UYVY_12BIT

The packed YUV 4:2:2 12-bit color data in order {u,y0,v,y1} 16-bits each (LSB).

CCF_YUY2_12BIT

The packed YUV 4:2:2 12-bit color data in order {y0,u,y1,v} 16-bits each (LSB).

CCF_UYVY_14BIT

The packed YUV 4:2:2 14-bit color data in order {u,y0,v,y1} 16-bits each (LSB).

CCF_YUY2_14BIT

The packed YUV 4:2:2 14-bit color data in order {y0,u,y1,v} 16-bits each (LSB).

CCF_UYVY_16BIT

The packed YUV 4:2:2 16-bit color data in order {u,y0,v,y1} 16-bits each (LSB).

CCF_YUY2_16BIT

The packed YUV 4:2:2 16-bit color data in order {y0,u,y1,v} 16-bits each (LSB).

CCF_UYVY_32F

The packed YUV 4:2:2 64-bit color data in order {u,y0,v,y1} float32 (Y:0..1, UV:-0.5..+0.5).

CCF_YUY2_32F

The packed YUV 4:2:2 64-bit color data in order {y0,u,y1,v} float32 (Y:0..1, UV:-0.5..+0.5).

CCF_NV12

4:2:0 Planar Y, UV interleaved

CCF_NV12_10BIT

4:2:0 Planar Y, UV interleaved, 10-bit data in 16-bit words each (LSB)

CCF_V210

V210 10-bit 4:2:2 color format.

CCF_SC10

Seachange 10-bit 4:2:2 color format (the 16-pixel 40-bytes chunks with 8-bit YUY2 data (32 bytes) followed by packed 2-bit trails (8 bytes)).

CCF_Y8

Monochrome (greyscale) data, 1 byte per pixel.

CCF_YUY2_4x4K

8K-specific format, consists of four 3840x2160 YUY2 buffers. The pData should point to an array of 4 pointers in order: { Top-Left, Top-Right, Bottom-Left, Bottom-Right }.

CCF_V210_4x4K

The same as above, but format of the buffers is V210.

CC_FRAME_TYPE

Coded video frame type.

Syntax

```
[v1_enum]
enum CC_FRAME_TYPE {
  CC_FRAME_TYPE_UNKNOWN = 0,
  CC_I_FRAME,
  CC_P_FRAME,
  CC_B_FRAME
};
```

Members

CC_I_FRAME

"INTRA" frame, no predictions, consists of intra blocks only. Used as reference for other types of frames.

CC_P_FRAME

The frame "PREDICTED" from previous I- or P-frame. Used as reference for next P- or B-frames.

CC_B_FRAME

"BIDIRECTIONAL" frame. Predicted form previous and next I- or P-frames in the temporal order. There is the reordering of frames in the bitstream to make the ability to obtain the "next" reference frame _before_ decoding the current B-frame. Not used for prediction.

CC_COLOUR_DESCRIPTION

The aggregate representation of CC_COLOUR_PRIMARIES (see page 108), CC_TRANSFER_CHARACTERISTICS (see page 113) ans CC_MATRIX_COEFFICIENTS (see page 111).

Syntax

```
struct CC_COLOUR_DESCRIPTION {
  CC_COLOUR_PRIMARIES CP;
  CC_TRANSFER_CHARACTERISTICS TC;
  CC_MATRIX_COEFFICIENTS MC;
};
```

Members

CP

see CC_COLOUR_PRIMARIES (see page 108) description.

TC

see CC_TRANSFER_CHARACTERISTICS (see page 113) description.

MC

see CC_MATRIX_COEFFICIENTS (see page 111) description.

CC_COLOUR_PRIMARIES

The chromaticity coordinates of the source primaries (see iso/iec 13818-2 6.3.6 Table 6-7 for details).

Syntax

```
[v1_enum]
enum CC_COLOUR_PRIMARIES {
  CC_CPRIMS_UNKNOWN = 0,
  CC_CPRIMS_ITUR_BT_709,
  CC_CPRIMS_UNSPECIFIED,
  CC_CPRIMS_RESERVED,
  CC_CPRIMS_ITUR_BT_470_M,
  CC_CPRIMS_ITUR_BT_470_BG,
  CC_CPRIMS_SMPTE_170M,
  CC_CPRIMS_SMPTE_240M,
  CC_CPRIMS_GENERIC_FILM,
  CC_CPRIMS_ITUR_BT_2020,
  CC_CPRIMS_SMPTE_ST_428_1,
  CC_CPRIMS_SMPTE_ST_431_2,
  CC_CPRIMS_SMPTE_ST_432_1,
  CC_CPRIMS_DCI_P3 = CC_CPRIMS_SMPTE_ST_431_2,
```

```
  CC_CPRIMS_P3_D65 = CC_CPRIMS_SMPTE_ST_432_1
};
```

Members

CC_CPRIMS_UNKNOWN

0 Forbidden value.

CC_CPRIMS_ITUR_BT_709

1 Recommendation ITU-R BT.709.

CC_CPRIMS_UNSPECIFIED

2 Unspecified Video - image characteristics are unknown.

CC_CPRIMS_RESERVED

3 Reserved.

CC_CPRIMS_ITUR_BT_470_M

4 Recommendation ITU-R BT.470 System M.

CC_CPRIMS_ITUR_BT_470_BG

5 Recommendation ITU-R BT.470 System B, G.

CC_CPRIMS_SMPTE_170M

6 SMPTE 170M.

CC_CPRIMS_SMPTE_240M

7 SMPTE 240M (1987).

CC_CPRIMS_GENERIC_FILM

8 Generic film (colour filters using Illuminant C)

CC_CPRIMS_ITUR_BT_2020

9 Recommendation ITU-R BT.2020

CC_CPRIMS_SMPTE_ST_428_1

10 SMPTE ST 428-1 (CIE 1931 XYZ as in ISO 11664-1)

CC_CPRIMS_SMPTE_ST_431_2

11 SMPTE ST 431-2 (2011)

CC_CPRIMS_SMPTE_ST_432_1

12 SMPTE ST 432-1 (2010)

CC_CPRIMS_DCI_P3

P3 with DCI white point

CC_CPRIMS_P3_D65

P3 with D65 white point

CC_GOP_DESCR

Group of Pictures (GOP) description.

Syntax

```
struct CC_GOP_DESCR {
  CC_PERIOD N;
  CC_UINT M;
};
```

Members

N

The distance between consecutive I frames.

M

The distance between consecutive P frames. N must be multiple of M.

CC_GOP_PATTERN

Symbolic names for CC_GOP_DESCR::N field.

Syntax

```
[v1_enum]
enum CC_GOP_PATTERN {
  CC_GOP_I = 0,
  CC_GOP_IP = 1,
  CC_GOP_IBP = 2,
  CC_GOP_IBBP = 3,
  CC_GOP_IBBBP = 4,
  CC_GOP_IBBBBP = 5
};
```

Members

CC_GOP_I

"IIIIII..." - No P- nor B-frames used.

CC_GOP_IP

"I(P)..." - No B-frames used. GOP_MaxLength specifies the number of P-frames between consecutive I-frames.

CC_GOP_IBP

"IbPbPbP..." - one B-frame between consecutive I- or P-frames.

CC_GOP_IBBP

"IbbPbbPbbP..." - two B-frames between consecutive I- or P-frames (most commonly used).

CC_GOP_IBBBP

"IbbbPbbbP..." - three B-frames between consecutive I- or P-frames.

CC_GOP_IBBBBP

"IbbbbPbbbbP..." - four B-frames between consecutive I- or P-frames.

CC_INTERLACE_TYPE

The field order of video frame.

Syntax

```
[v1_enum]
enum CC_INTERLACE_TYPE {
  CC_UNKNOWN_INTERLACE_TYPE = -1,
  CC_NO_INTERLACE = 0,
  CC_PROGRESSIVE = 0,
  CC_TOP_FIELD_FIRST = 1,
  CC_TFF = 1,
  CC_BOTTOM_FIELD_FIRST = 2,
  CC_BFF = 2,
  CC_TELECINE = 23
};
```

Members

CC_NO_INTERLACE

Progressive frame (default).

CC_TOP_FIELD_FIRST

Interlaced frame with Top Field First.

CC_BOTTOM_FIELD_FIRST

Interlaced frame with Bottom Field First.

CC_TELECINE

2:3 pulldown (valuable only for 23.976 & 24 fps).

CC_MATRIX_COEFFICIENTS

The matrix coefficients used in deriving luminance and chrominance signals from the green, blue, and red primaries (see iso/iec 13818-2 6.3.6 Table 6-9 for details).

Syntax

```
[v1_enum]
enum CC_MATRIX_COEFFICIENTS {
  CC_MCOEFS_UNKNOWN = 0,
  CC_MCOEFS_ITUR_BT_709,
  CC_MCOEFS_UNSPECIFIED,
  CC_MCOEFS_RESERVED,
  CC_MCOEFS_FCC,
  CC_MCOEFS_ITUR_BT_470_BG,
  CC_MCOEFS_SMPTE_170M,
  CC_MCOEFS_SMPTE_240M,
  CC_MCOEFS_YCGCO,
  CC_MCOEFS_ITUR_BT_2020_NON_CONST,
  CC_MCOEFS_ITUR_BT_2020_CONST,
```

```
  CC_MCOEFS_SMPTE_2085
};
```

Members

CC_MCOEFS_UNKNOWN

0 Forbidden value.

CC_MCOEFS_ITUR_BT_709

1 Recommendation ITU-R BT.709 (Kr = 0.2126; Kb = 0.0722).

CC_MCOEFS_UNSPECIFIED

2 Unspecified Video - image characteristics are unknown.

CC_MCOEFS_RESERVED

3 Reserved for future use by ITU-T / ISO/IEC.

CC_MCOEFS_FCC

4 FCC (Kr = 0.30; Kb = 0.11)

CC_MCOEFS_ITUR_BT_470_BG

5 Recommendation ITU-R BT.470 System B,G (Kr = 0.299; Kb = 0.114).

CC_MCOEFS_SMPTE_170M

6 SMPTE 170M (Kr = 0.299; Kb = 0.114).

CC_MCOEFS_SMPTE_240M

7 SMPTE 240M (1987) (Kr = 0.212; Kb = 0.087).

CC_MCOEFS_YCGCO

8 YCgCo

CC_MCOEFS_ITUR_BT_2020_NON_CONST

9 ITU-R BT.2020 non-constant luminance system (Kr = 0.2627; Kb = 0.0593).

CC_MCOEFS_ITUR_BT_2020_CONST

10 ITU-R BT.2020 constant luminance system (Kr = 0.2627; Kb = 0.0593).

CC_MCOEFS_SMPTE_2085

11 SMPTE ST 2085 (2015) (aka Y' D'z D'x)

CC_MB_STRUCTURE

The coded macroblock structure.

Syntax

```
[v1_enum]
enum CC_MB_STRUCTURE {
  CC_MBAFF = 0,
  CC_MB_PROGRESSIVE,
  CC_MB_INTERLACED
```

```
};
```

Members

CC_MBAFF

Automatic selection of the macroblock structure, depending on the contents (MBAFF).

CC_MB_PROGRESSIVE

Progressive macroblock structure.

CC_MB_INTERLACED

Interlaced macroblock structure.

CC_PICTURE_STRUCTURE

The coding structure of coded video frame.

Syntax

```
[v1_enum]
enum CC_PICTURE_STRUCTURE {
  CC_PICTURE_STRUCTURE_UNKNOWN = -1,
  CC_WHOLE_FRAME = 0,
  CC_PAIR_OF_FIELDS,
  CC_PAFF
};
```

Members

CC_WHOLE_FRAME

Whole frame (default).

CC_PAIR_OF_FIELDS

Two consecutive fields, encoded independently.

CC_PAFF

Automatic selection of the picture structure, depending on the contents (PAFF).

CC_TRANSFER_CHARACTERISTICS

The opto-electronic transfer characteristic of the source picture (see iso/iec 13818-2 6.3.6 Table 6-8 for details).

Syntax

```
[v1_enum]
enum CC_TRANSFER_CHARACTERISTICS {
  CC_TXCHRS_UNKNOWN = 0,
  CC_TXCHRS_ITUR_BT_709,
  CC_TXCHRS_UNSPECIFIED,
  CC_TXCHRS_RESERVED,
  CC_TXCHRS_ITUR_BT_470_M,
  CC_TXCHRS_ITUR_BT_470_BG,
  CC_TXCHRS_SMPTE_170M,
```

```
  CC_TXCHRS_SMPTE_240M,
  CC_TXCHRS_LINEAR,
  CC_TXCHRS_LOG_100,
  CC_TXCHRS_LOG_316,
  CC_TXCHRS_IEC_61966_2_4,
  CC_TXCHRS_ITUR_BT_1361,
  CC_TXCHRS_IEC_61966_2_1,
  CC_TXCHRS_ITUR_BT_2020_10BIT,
  CC_TXCHRS_ITUR_BT_2020_12BIT,
  CC_TXCHRS_SMPTE_ST_2084,
  CC_TXCHRS_SMPTE_ST_428_1,
  CC_TXCHRS_HLG
};
```

Members

CC_TXCHRS_UNKNOWN

0 Forbidden value.

CC_TXCHRS_ITUR_BT_709

1 Recommendation ITU-R BT.709.

CC_TXCHRS_UNSPECIFIED

2 Unspecified Video - image characteristics are unknown.

CC_TXCHRS_RESERVED

3 Reserved.

CC_TXCHRS_ITUR_BT_470_M

4 Recommendation ITU-R BT.470 System M (assumed display gamma 2.2).

CC_TXCHRS_ITUR_BT_470_BG

5 Recommendation ITU-R BT.470 System B, G (assumed display gamma 2.8).

CC_TXCHRS_SMPTE_170M

6 SMPTE 170M.

CC_TXCHRS_SMPTE_240M

7 SMPTE 240M (1987).

CC_TXCHRS_LINEAR

8 Linear transfer characteristics.

CC_TXCHRS_LOG_100

9 Logarithmic transfer characteristic (100:1 range).

CC_TXCHRS_LOG_316

10 Logarithmic transfer characteristic (316.22777:1 range).

CC_TXCHRS_IEC_61966_2_4

11 IEC 61966-2-4

CC_TXCHRS_ITUR_BT_1361

12 Recommendation ITU-R BT.1361 extended colour gamut system

CC_TXCHRS_IEC_61966_2_1
13 IEC 61966-2-1 (sRGB or sYCC)

CC_TXCHRS_ITUR_BT_2020_10BIT
14 Recommendation ITU-R BT.2020 for 10 bit system

CC_TXCHRS_ITUR_BT_2020_12BIT
15 Recommendation ITU-R BT.2020 for 12 bit system

CC_TXCHRS_SMPTE_ST_2084
16 SMPTE ST 2084 for 10, 12, 14, and 16-bit systems.

CC_TXCHRS_SMPTE_ST_428_1
17 SMPTE ST 428-1

CC_TXCHRS_HLG
18 Hybrid Log Gamma

CC_VIDEO_FORMAT

The representation of the pictures before being coded in accordance with iso/iec 13818-2 (MPEG2) specs. (see iso/iec 13818-2 6.3.6 Table 6-6 for details).

Syntax

```
[v1_enum]
enum CC_VIDEO_FORMAT {
  CC_VIDEO_FORMAT_COMPONENT = 0,
  CC_VIDEO_FORMAT_PAL,
  CC_VIDEO_FORMAT_NTSC,
  CC_VIDEO_FORMAT_SECAM,
  CC_VIDEO_FORMAT_MAC,
  CC_VIDEO_FORMAT_UNSPECIFIED
};
```

Members

CC_VIDEO_FORMAT_COMPONENT
Component.

CC_VIDEO_FORMAT_PAL
PAL.

CC_VIDEO_FORMAT_NTSC
NTSC.

CC_VIDEO_FORMAT_SECAM
SECAM.

CC_VIDEO_FORMAT_MAC
MAC.

CC_VIDEO_FORMAT_UNSPECIFIED
Unspecified.

CC_QUANT_DESCR

Quantization parameter for defining mquant parameter in VBR modes.

Syntax

```
struct CC_QUANT_DESCR {
  CC_BYTE Qi;
  CC_BYTE Qp;
  CC_BYTE Qb;
};
```

Members

Qi
Start quantization parameter for I frames.

Qp
Start quantization parameter for P frames.

Qb
Start quantization parameter for B frames.

CC_AFF_TYPE

Syntax

```
[v1_enum]
enum CC_AFF_TYPE {
  CC_AFF_FRAME = 0,
  CC_AFF_FIELD,
  CC_AFF_ADAPTIVE
};
```

Members

CC_AFF_FRAME
Frame coding.

CC_AFF_FIELD
Field coding.

CC_AFF_ADAPTIVE
Adaptive frame/field coding.

CC_ADD_VIDEO_FRAME_PARAMS

Syntax

```
struct CC_ADD_VIDEO_FRAME_PARAMS {
  CC_COLOR_FMT cFormat;
  CC_SIZE szFrame;
  CC_INT iStride;
  RECT rcCrop;
};
```

Members

cFormat

The color format (see page 103) of the frame.

szFrame

The frame size in pixels.

iStride

The stride of a single video row, measured in bytes. A negative stride value means that order of rows is reversed (the first row is on the bottom of the frame). If stride is set to zero, the real value is derived from the *szFrame* and *cFormat*.

rcCrop

The crop area. Set all to 0's if there is no crop.

Audio API

This reference section contains descriptions of Audio API interfaces and enumerations.

Interfaces

	Name	Description
	ICC_AudioConsumer (see page 119)	Provides the methods specific for the generic audio data consumer.
	ICC_AudioProducer (see page 121)	Provides the methods specific for the generic waveform-audio producer.
	ICC_AudioDecoder (see page 124)	The default and main interface to control the instance of CinecoderMpegAudioDecoder class.
	ICC_AudioEncoder (see page 128)	The default and main interface to control the instance of CinecoderMpegAudioEncoder class.
	ICC_AudioFrameInfo (see page 131)	Represents the generic audio stream description.
	ICC_AudioStreamInfo (see page 132)	Represents the generic audio stream description.
	ICC_AudioEncoderSettings (see page 134)	The common settings for any audio encoder initialization.

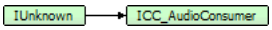
Enumerations

	Name	Description
	CC_AUDIO_FMT (see page 136)	The format of uncompressed audio data.

ICC_AudioConsumer Interface

Provides the methods specific for the generic audio data consumer.

Class Hierarchy



Syntax

```
[object, uuid(00003002-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_AudioConsumer : IUnknown;
```

Methods

	Name	Description
	GetAudioStreamInfo (see page 119)	Get the description of audio stream which is being decoded.
	ProcessAudio (see page 119)	Method to process the uncompressed audio data.

Methods

GetAudioStreamInfo

Get the description of audio stream which is being decoded.

Syntax

```
HRESULT GetAudioStreamInfo(
    [out,retval] ICC_AudioStreamInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

ProcessAudio

Method to process the uncompressed audio data.

Syntax

```
HRESULT ProcessAudio(
    [in] CC_AUDIO_FMT fmt,
    [in, size_is(cbSize)] const BYTE* pbData,
    [in] DWORD cbSize,
    [out,retval,defaultvalue(NULL)] DWORD * pcbRetSize
);
```

Parameters

fmt

The format of incoming waveform.

pbData

The waveform's data.

cbSize

Number of bytes in the buffer, must be multiple of BlockAlign of corresponding CC_AUDIO_FMT.

pcbRetSize

Number of bytes actually put.

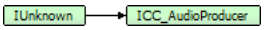
Returns

Returns S_OK if successful or an error value otherwise.

ICC_AudioProducer Interface

Provides the methods specific for the generic waveform-audio producer.

Class Hierarchy



Syntax

```
[object, uuid(00003001-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_AudioProducer : IUnknown;
```

Methods

	Name	Description
	GetAudio (see page 121)	Retrieves the uncompressed audio data from the audio producer. Remark: To obtain the size of the buffer to hold the resulted samples, put NULL instead of pbData.
	GetAudioFrameInfo (see page 122)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 122)	Get the description of the audio stream.
	GetSampleBytes (see page 122)	
	IsFormatSupported (see page 123)	

Methods

GetAudio

Retrieves the uncompressed audio data from the audio producer. Remark: To obtain the size of the buffer to hold the resulted samples, put NULL instead of pbData.

Syntax

```
HRESULT GetAudio(
    [in] CC_AUDIO_FMT fmt,
    [out, size_is(cbSize)] BYTE * pbData,
    [in] DWORD cbSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

fmt

The format of incoming waveform.

pbData

Buffer for incoming audio.

cbSize

Buffer size, in bytes.

pcbRetSize

Pointer to the variable that receives the number of got bytes. GetAudio sets this value to zero before doing any work.

Returns

Returns S_OK if successful or an error value otherwise.

GetAudioFrameInfo

Get the description of the current audio frame.

Syntax

```
HRESULT GetAudioFrameInfo(
    [out,retval] ICC_AudioFrameInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

GetAudioStreamInfo

Get the description of the audio stream.

Syntax

```
HRESULT GetAudioStreamInfo(
    [out,retval] ICC_AudioStreamInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

GetSampleBytes

Syntax

```
HRESULT GetSampleBytes(
    [in] CC_AUDIO_FMT fmt,
    [out,retval] DWORD * pNumBytes
);
```

IsFormatSupported

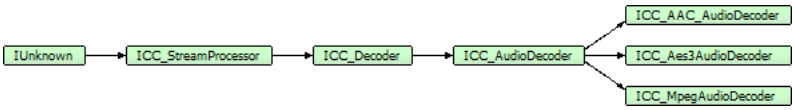
Syntax

```
HRESULT IsFormatSupported(
    [in] CC_AUDIO_FMT fmt,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pBool
);
```

ICC_AudioDecoder Interface

The default and main interface to control the instance of CinecoderMpegAudioDecoder class.

Class Hierarchy



Syntax

```
[object, uuid(00003700-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_AudioDecoder : ICC_Decoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_AudioDecoder Interface

	Name	Description
	GetAudio (see page 125)	Retrieves the uncompressed audio data from the audio producer. Remark: To obtain the size of the buffer to hold the resulted samples, put NULL instead of pbData.
	GetAudioFrameInfo (see page 126)	Get the description of the current audio frame.

	GetAudioStreamInfo (see page 126)	Get the description of the audio stream.
	GetSampleBytes (see page 126)	
	IsFormatSupported (see page 127)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_AudioDecoder Interface

	Name	Description
R	NumSamples (see page 127)	The number of ready audio samples at the output.

Methods

GetAudio

Retrieves the uncompressed audio data from the audio producer. Remark: To obtain the size of the buffer to hold the resulted samples, put NULL instead of pbData.

Syntax

```
HRESULT GetAudio(
    [in] CC_AUDIO_FMT fmt,
    [out, size_is(cbSize)] BYTE * pbData,
    [in] DWORD cbSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

fmt

The format of incoming waveform.

pbData

Buffer for incoming audio.

cbSize

Buffer size, in bytes.

pcbRetSize

Pointer to the variable that receives the number of got bytes. GetAudio sets this value to zero before doing any work.

Returns

Returns S_OK if successful or an error value otherwise.

GetAudioFrameInfo

Get the description of the current audio frame.

Syntax

```
HRESULT GetAudioFrameInfo(
    [out,retval] ICC_AudioFrameInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

GetAudioStreamInfo

Get the description of the audio stream.

Syntax

```
HRESULT GetAudioStreamInfo(
    [out,retval] ICC_AudioStreamInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

GetSampleBytes

Syntax

```
HRESULT GetSampleBytes(
    [in] CC_AUDIO_FMT fmt,
    [out,retval] DWORD * pNumBytes
```

```
);
```

IsFormatSupported

Syntax

```
HRESULT IsFormatSupported(
    [in] CC_AUDIO_FMT fmt,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pBool
);
```

Properties

NumSamples

The number of ready audio samples at the output.

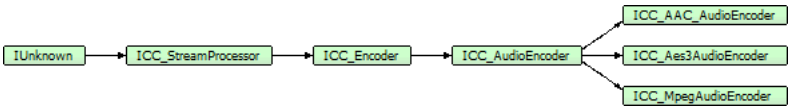
Syntax

```
__property DWORD* NumSamples;
```

ICC_AudioEncoder Interface

The default and main interface to control the instance of CinecoderMpegAudioEncoder class.

Class Hierarchy



Syntax

```
[object, uuid(00003400-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_AudioEncoder : ICC_Encoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_AudioEncoder Interface

	Name	Description
	GetAudioFrameInfo (see page 129)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 129)	Get the description of audio stream which is being decoded.
	ProcessAudio (see page 130)	Puts another audio uncompressed data to the audio consumer.

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.

R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

Methods

GetAudioFrameInfo

Get the description of the current audio frame.

Syntax

```
HRESULT GetAudioFrameInfo(
    [out,retval] ICC_AudioFrameInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

GetAudioStreamInfo

Get the description of audio stream which is being decoded.

Syntax

```
HRESULT GetAudioStreamInfo(
    [out,retval] ICC_AudioStreamInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

ProcessAudio

Puts another audio uncompressed data to the audio consumer.

Syntax

```
HRESULT ProcessAudio(
    [in] CC_AUDIO_FMT Format,
    [in, size_is(cbSize)] const BYTE* pbData,
    [in] DWORD cbSize,
    [out,retval,defaultvalue(NULL)] DWORD * pcbRetSize
);
```

Parameters

Format

The format of incoming audio data.

pbData

Buffer with the audio data.

cbSize

Number of bytes in the buffer, must be multiple of BlockAlign of corresponding CC_AUDIO_FMT.

pcbRetSize

Number of bytes actually put.

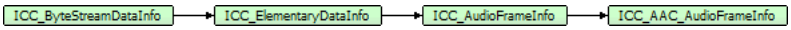
Returns

Returns S_OK if successful or an error value otherwise.

ICC_AudioFrameInfo Interface

Represents the generic audio stream description.

Class Hierarchy



Syntax

```
[object, uuid(00003302-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_AudioFrameInfo : ICC_ElementaryDataInfo;
```

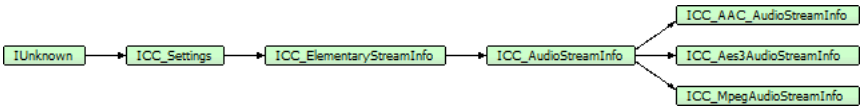
Properties

	Name	Description
R	DTS (see page 49)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 49), except the coded video with B-frames.
R	Duration (see page 49)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
R	NumSamples (see page 49)	Number of samples of the elementary data. In case of coded video frames, there is usually 1 or 2 sample(s) (1 frame or 2 fields). In case of coded audio, there are the number of audio samples, encoded into audio frame.
R	PresentationDelta (see page 49)	The presentation delta, in samples, of the elementary data. It is actual for data which was reordered during encoding process (f.e. coded video with B-frames).
R	PTS (see page 49)	The Presentation Data Stamp (PTS) of the elementary data. The PTS based on CC_TIMEBASE, specified for object which generates the elementary data.
R	SampleOffset (see page 49)	The frame's first sample order number.
R	SequenceEntryFlag (see page 50)	The sequence entry point flag. In the case of mpeg video, it means that SEQUENCE_HEADER presents in the elementary data. This flag is used to signal the multiplexers to generate the entry point (like System Header) at this elementary data.

ICC_AudioStreamInfo Interface

Represents the generic audio stream description.

Class Hierarchy



Syntax

```
[object, uuid(00003301-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_AudioStreamInfo : ICC_ElementaryStreamInfo;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ElementaryStreamInfo Interface

	Name	Description
R	BitRate (see page 52)	The bitrate (max) of the elementary stream.
R	FrameRate (see page 52)	The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.
R	StreamType (see page 52)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

ICC_AudioStreamInfo Interface

	Name	Description
	BitsPerSample (see page 133)	The bit depth of one audio sample (of one channel).
	ChannelMask (see page 133)	The mask of channels. See the CC_AUDIO_CHANNEL_MASK for details
	NumChannels (see page 133)	The number of channels in the waveform-audio data.
	SampleRate (see page 133)	The audio sampling frequency.

Properties

BitsPerSample

The bit depth of one audio sample (of one channel).

Syntax

```
__property CC_UINT * BitsPerSample;
```

ChannelMask

The mask of channels. See the CC_AUDIO_CHANNEL_MASK for details

Syntax

```
__property CC_UINT * ChannelMask;
```

NumChannels

The number of channels in the waveform-audio data.

Syntax

```
__property CC_UINT * NumChannels;
```

SampleRate

The audio sampling frequency.

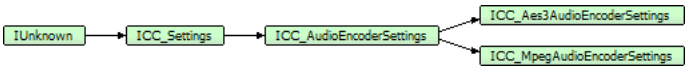
Syntax

```
__property CC_UINT * SampleRate;
```

ICC_AudioEncoderSettings Interface

The common settings for any audio encoder initialization.

Class Hierarchy



Syntax

```
[object, uuid(a12bf062-fd75-44a1-9adf-ad6d1a353f74), pointer_default(unique), local]
interface ICC_AudioEncoderSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_AudioEncoderSettings Interface

	Name	Description
	BitRate (see page 135)	The bitrate of entire audio bitstream or corresponding audio frame.
	BitsPerSample (see page 135)	The bit depth of one audio sample (of one channel).
	FrameRate (see page 135)	The number of audio frames per second.
	NumChannels (see page 135)	The number of channels in the waveform-audio data.
	SampleRate (see page 135)	The audio sampling frequency.

Properties

BitRate

The bitrate of entire audio bitstream or corresponding audio frame.

Syntax

```
__property CC_BITRATE BitRate;
```

BitsPerSample

The bit depth of one audio sample (of one channel).

Syntax

```
__property CC_UINT BitsPerSample;
```

FrameRate

The number of audio frames per second.

Syntax

```
__property CC_FRAME_RATE FrameRate;
```

NumChannels

The number of channels in the waveform-audio data.

Syntax

```
__property CC_UINT NumChannels;
```

SampleRate

The audio sampling frequency.

Syntax

```
__property CC_UINT SampleRate;
```

CC_AUDIO_FMT

The format of uncompressed audio data.

Syntax

```
[v1_enum]
enum CC_AUDIO_FMT {
  CAF_UNKNOWN = 0x00000000,
  CAF_PCM8 = 0x00020000,
  CAF_PCM16 = 0x00010000,
  CAF_PCM24 = 0x00030000,
  CAF_PCM32 = 0x00040000,
  CAF_PCM32_24 = 0x00070000,
  CAF_FLOAT32 = 0x00100000,
  CAF_PCM8_1CH = 0x00020001,
  CAF_PCM8_2CH = 0x00020002,
  CAF_PCM16_1CH = 0x00010001,
  CAF_PCM16_2CH = 0x00010002,
  CAF_PCM16_6CH = 0x00010006,
  CAF_PCM24_1CH = 0x00030001,
  CAF_PCM24_2CH = 0x00030002,
  CAF_PCM24_6CH = 0x00030006,
  CAF_PCM32_1CH = 0x00040001,
  CAF_PCM32_2CH = 0x00040002,
  CAF_PCM32_6CH = 0x00040006,
  CAF_PCM32_24_1CH = 0x00070001,
  CAF_PCM32_24_2CH = 0x00070002,
  CAF_PCM32_24_6CH = 0x00070006,
  CAF_FLOAT32_1CH = 0x00100001,
  CAF_FLOAT32_2CH = 0x00100002,
  CAF_FMT_MASK = 0x00FF0000,
  CAF_CNL_MASK = 0x000000FF
};
```

Members

CAF_UNKNOWN

Unknown audio format.

CAF_PCM8

8-bit signed integer samples. Number of samples is stream-defined.

CAF_PCM16

16-bit signed integer samples. Number of samples is stream-defined.

CAF_PCM24

24-bit signed integer samples. Number of samples is stream-defined.

CAF_PCM32

32-bit signed integer samples. Number of samples is stream-defined.

CAF_PCM32_24

24-bit signed integer samples, sign-extended to 32-bit width. Number of samples is stream-defined.

CAF_FLOAT32
32-bit IEEE-754 floating point samples. Number of samples is stream-defined.

CAF_PCM8_1CH
8-bit signed samples, 1 channel (mono).

CAF_PCM8_2CH
8-bit signed samples, 2 channel (stereo), left channel first.

CAF_PCM16_1CH
16-bit signed samples, 1 channel (mono).

CAF_PCM16_2CH
16-bit signed samples, 2 channels (stereo), left channel first.

CAF_PCM16_6CH
16-bit signed samples, 6 channeld (5+1) LF, RF, LR, RR, Center, Bass

CAF_PCM24_1CH
24-bit signed samples, 1 channel (mono).

CAF_PCM24_2CH
24-bit signed samples, 2 channels (stereo), left channel first.

CAF_PCM24_6CH
24-bit signed samples, 6 channeld (5+1) LF, RF, LR, RR, Center, Bass

CAF_PCM32_1CH
32-bit signed samples, 1 channel (mono).

CAF_PCM32_2CH
32-bit signed samples, 2 channels (stereo), left channel first.

CAF_PCM32_6CH
32-bit signed samples, 6 channeld (5+1) LF, RF, LR, RR, Center, Bass

CAF_PCM32_24_1CH
24-in-32-bit signed samples, 1 channel (mono).

CAF_PCM32_24_2CH
24-in-32-bit signed samples, 2 channels (stereo), left channel first.

CAF_PCM32_24_6CH
24-in-32-bit signed samples, 6 channeld (5+1) LF, RF, LR, RR, Center, Bass

CAF_FLOAT32_1CH
32-bit float samples (0..1), 1 channel (mono).

CAF_FLOAT32_2CH
32-bit float samples (0..1), 2 channels (stereo), left channel first.

Multiplex API

This reference section contains descriptions of Multiplex API interfaces, enumerations, and structures.

Interfaces

	Name	Description
	ICC_Demultiplexer (see page 140)	Interface provides the methods specific for general stream demultiplexer object.
	ICC_Multiplexer (see page 144)	Interface provides the methods specific for general stream multiplexer object.
	ICC_BaseMultiplexerPinSettings (see page 148)	
	ICC_DemultiplexedDataCallback (see page 150)	Unified demultiplexer data output - for Program & Transport demultiplexers.
	ICC_MultiplexedStreamInfo (see page 152)	
	ICC_MultiplexedDataDescr (see page 154)	The description of data coming from the multiplexer (via ICC_StreamDescr::DataInfo property).
	ICC_BaseMultiplexerSettings (see page 156)	

Enumerations

	Name	Description
	CC_ELEMENTARY_STREAM_TYPE (see page 160)	List of the elementary streams types (see ISO/IEC 13818-1 2.5.4.2 "Semantic definition of fields in Program Stream map").
	CC_MULTIPLEXED_STREAM_TYPE (see page 165)	The stream types.
	CC_MUX_OUTPUT_POLICY (see page 166)	
	CC_PES_ID (see page 168)	A list of the elementary stream IDs. Refer to table 2-18 of ISO/IEC 13818-1 2.4.3.7 "Semantic definition of fields in PES packets".
	CC_PSI_TABLE_ID (see page 169)	A list of the standard PSI Table ID's. Refer to the ISO/IEC 13818-1 2.4.4 "Program specific information".

	MPEG_SYSTEM_DESCRIPTOR_TAG (see page 170)	

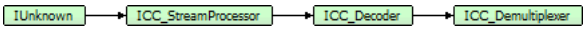
Structures

	Name	Description
	CC_ACCESS_UNIT_DESCR (see page 159)	Description of the multiplexed elementary stream data unit.
	CC_PACKET_DESCR (see page 167)	Description of the multiplexed packet.

ICC_Demultiplexer Interface

Interface provides the methods specific for general stream demultiplexer object.

Class Hierarchy



Syntax

```
[object, uuid(00001102-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_Demultiplexer : ICC_Decoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_Demultiplexer Interface

	Name	Description
	CatchStream (see page 141)	
	GetData (see page 142)	The method to obtain the demultiplexed packet directly (not via callback).
	GetStreamInfo (see page 142)	

	ReleaseAllStreams (see page 142)	
	ReleaseStream (see page 142)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Demultiplexer Interface

	Name	Description
R	DataSize (see page 143)	
R	SCR (see page 143)	The last clock reference (SCR or PCR) read from the multiplexed stream
R	StreamID (see page 143)	

Methods

CatchStream

Syntax

```
HRESULT CatchStream(
    [in] DWORD stream_id,
    [in,defaultvalue(NULL)] IUnknown * pDecoder,
    [in,defaultvalue(CC_CATCH_MODE_DEFAULT)] CC_CATCH_STREAM_MODE mode
);
```

Parameters

pDecoder

Expected an object with ICC_ByteStreamCallback interface

GetData

The method to obtain the demultiplexed packet directly (not via callback).

Syntax

```
HRESULT GetData(
    [out] CC_UINT * p_stream_id,
    [out,size_is(cbBufSize)] CC_PBYTE pbData,
    [in] CC_UINT cbBufSize,
    [out] CC_TIME * p_pts,
    [out,retval,defaultvalue(NULL)] CC_UINT * pcbRetSize
);
```

Parameters

p_stream_id

Can be NULL, otherwise the stream_id will be placed here.

pbData

Can be NULL, in such case the full data size will be returned via pcbRetSize.

cbBufSize

The number of bytes you have to copy to.

p_pts

Can be NULL, otherwise the first commenced pts of the data will be placed (or CC_NO_TIME if no pts).

pcbRetSize

Can be NULL.

GetStreamInfo

Syntax

```
HRESULT GetStreamInfo(
    [out,retval]ICC_MultiplexedStreamInfo ** pStreamInfo
);
```

ReleaseAllStreams

Syntax

```
HRESULT ReleaseAllStreams();
```

ReleaseStream

Syntax

```
HRESULT ReleaseStream(
    [in] DWORD stream_id
);
```

Properties

DataSize

Syntax

```
__property CC_UINT* DataSize;
```

SCR

The last clock reference (SCR or PCR) read from the multiplexed stream

Syntax

```
__property CC_SCR* SCR;
```

StreamID

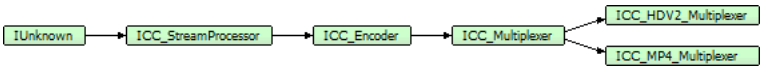
Syntax

```
__property CC_UINT* StreamID;
```

ICC_Multiplexer Interface

Interface provides the methods specific for general stream multiplexer object.

Class Hierarchy



Syntax

```
[object, uuid(6EEBE83C-8F8B-4321-8C1E-D950C7E0A282), pointer_default(unique), local]
interface ICC_Multiplexer : ICC_Encoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_Multiplexer Interface

	Name	Description
	CreatePin (see page 145)	Method creates an input pin to add the specified type of elementary data needed to be multipolexed.
	CreatePinByXml (see page 146)	Method creates an input pin to add the specified type of elementary data needed to be multipolexed.
	GetPin (see page 146)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.

	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
	DataSize (see page 64)	

ICC_Multiplexer Interface

	Name	Description
	PinCount (see page 146)	/// Method creates an input pin to add the specified type of elementary data needed to be multipolexed. /// Returns: Returns S_OK if successful or error code otherwise. HRESULT CreatePinByType([in] CC_ELEMENTARY_STREAM_TYPE (see page 160) stream_type, // Pin type [out,retval] ICC_ByteStreamConsumer (see page 12) **pOutput // Address where pointer to the created object will be stored.);

Methods

CreatePin

Method creates an input pin to add the specified type of elementary data needed to be multipolexed.

Syntax

```
HRESULT CreatePin(
    [in] ICC_Settings * pPinDescr,
    [out,retval] ICC_ByteStreamConsumer ** pOutput
);
```

Parameters

pPinDescr

Pin description.

pOutput

Address where pointer to the created object will be stored.

Returns

Returns S_OK if successful or error code otherwise.

CreatePinByXml

Method creates an input pin to add the specified type of elementary data needed to be multipolexed.

Syntax

```
HRESULT CreatePinByXml(
    [in] CC_STRING pPinXmlDescr,
    [out,retval] ICC_ByteStreamConsumer ** pOutput
);
```

Parameters

pPinXmlDescr

Pin description in XML format

pOutput

Address where pointer to the created object will be stored.

Returns

Returns S_OK if successful or error code otherwise.

GetPin

Syntax

```
HRESULT GetPin(
    [in] CC_UINT PinNumber,
    [out,retval] ICC_ByteStreamConsumer ** pOutput
);
```

Properties

PinCount

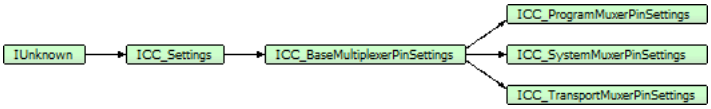
/// Method creates an input pin to add the specified type of elementary data needed to be multipolexed. /// Returns: Returns S_OK if successful or error code otherwise. HRESULT CreatePinByType([in] CC_ELEMENTARY_STREAM_TYPE (see page 160) stream_type, // Pin type [out,retval] ICC_ByteStreamConsumer (see page 12) **pOutput // Address where pointer to the created object will be stored.);

Syntax

```
__property CC_UINT * PinCount;
```

ICC_BaseMultiplexerPinSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001A02-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_BaseMultiplexerPinSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_BaseMultiplexerPinSettings Interface

	Name	Description
	BasePTS (see page 149)	
	BitRate (see page 149)	
	DataAlignPeriod (see page 149)	The period at which the access units will align to the packet's boundary. By default, audio streams aligned only at the beginning, video streams are aligned each I-frame.
	FrameRate (see page 149)	
	SampleRate (see page 149)	

	StreamID (see page 149)	
	StreamType (see page 149)	

Properties

BasePTS

Syntax

```
__property [in] BasePTS;
```

BitRate

Syntax

```
__property [in] BitRate;
```

DataAlignPeriod

The period at which the access units will align to the packet's boundary. By default, audio streams aligned only at the beginning, video streams are aligned each I-frame.

Syntax

```
__property [in] DataAlignPeriod;
```

FrameRate

Syntax

```
__property [in] FrameRate;
```

SampleRate

Syntax

```
__property [in] SampleRate;
```

StreamID

Syntax

```
__property [in] StreamID;
```

StreamType

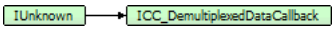
Syntax

```
__property [in] StreamType;
```

ICC_DemultiplexedDataCallback Interface

Unified demultiplexer data output - for Program & Transport demultiplexers.

Class Hierarchy



Syntax

```
[object, uuid(00001101-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_DemultiplexedDataCallback : IUnknown;
```

Methods

	Name	Description
	ProcessData (see page 150)	

Methods

ProcessData

Syntax

```
HRESULT ProcessData(
    [in] DWORD stream_id,
    [in,size_is(cbSize)] const BYTE * pbData,
    [in] DWORD cbSize,
    [in] CC_TIME pts,
    [in] IUnknown * pDescr
);
```

Parameters

stream_id

stream_id (or pid in case of TS).

pbData

Pointer to the data.

cbSize

The size, in bytes, of the incoming data.

pts

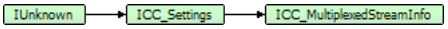
The presentation time stamp of the first unit commencing in the packet. If no units commencing, the value of CC_NO_TIME will be assigned.

pDescr

The demultiplexer itself.

ICC_MultiplexedStreamInfo Interface

Class Hierarchy



Syntax

```
[object, uuid(00001803-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_MultiplexedStreamInfo : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_MultiplexedStreamInfo Interface

	Name	Description
	GetProgram (see page 153)	

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_MultiplexedStreamInfo Interface

	Name	Description
R	BitRate (see page 153)	
R	NumPrograms (see page 153)	
R	StreamID (see page 153)	
R	StreamType (see page 153)	

Methods

GetProgram

Syntax

```
HRESULT GetProgram(
    [in] DWORD ProgramIdx,
    [out,retval]ICC_ProgramInfo**
);
```

Properties

BitRate

Syntax

```
__property CC_BITRATE* BitRate;
```

NumPrograms

Syntax

```
__property DWORD* NumPrograms;
```

StreamID

Syntax

```
__property WORD* StreamID;
```

StreamType

Syntax

```
__property CC_MULTIPLEXED_STREAM_TYPE* StreamType;
```

ICC_MultiplexedDataDescr Interface

The description of data coming from the multiplexer (via ICC_StreamDescr::DataInfo property).

Class Hierarchy

Syntax

```
[object, uuid(022813a8-3467-4f49-b17a-57bffd5fd21f), pointer_default(unique), local]
interface ICC_MultiplexedDataDescr : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_MultiplexedDataDescr Interface

	Name	Description
	GetAccessUnitInfo (see page 155)	
	GetPacketInfo (see page 155)	

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_MultiplexedDataDescr Interface

	Name	Description
R	NumAccessUnits (see page 155)	
R	NumPackets (see page 155)	

Methods

GetAccessUnitInfo

Syntax

```
HRESULT GetAccessUnitInfo(
    [in] CC_UINT idx,
    [out,retval]MPG_ACCESS_UNIT_DESCR*
);
```

GetPacketInfo

Syntax

```
HRESULT GetPacketInfo(
    [in] CC_UINT idx,
    [out,retval]CC_PACKET_DESCR*
);
```

Properties

NumAccessUnits

Syntax

```
__property CC_UINT* NumAccessUnits;
```

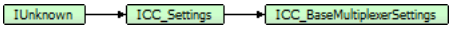
NumPackets

Syntax

```
__property CC_UINT* NumPackets;
```

ICC_BaseMultiplexerSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001A01-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_BaseMultiplexerSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_BaseMultiplexerSettings Interface

	Name	Description
	AsyncInputMode (see page 157)	
	BitRate (see page 157)	Target bitrate of the multiplexed stream. If zero - calculates automatically.
	InitialPTS (see page 157)	Clock reference for the first byte of multiplexed stream, in 90000 Hz units.
	MaxOutputBlkSize (see page 157)	
	OutputPolicy (see page 157)	The output policy for multiplexer.
	RateMode (see page 157)	Target bitrate mode - constant or variable.
	StreamType (see page 158)	Target multiplexed stream type.

	TrailingAlignment (see page 158)	The trailing stream alignment, specified in bytes. (1000 means 1000-bytes alignment of the stream).

Properties

AsyncInputMode

Syntax

```
__property CC_BOOL AsyncInputMode;
```

BitRate

Target bitrate of the multiplexed stream. If zero - calculates automatically.

Syntax

```
__property CC_BITRATE BitRate;
```

InitialPTS

Clock reference for the first byte of multiplexed stream, in 90000 Hz units.

Syntax

```
__property CC_TIME InitialPTS;
```

MaxOutputBlkSize

Syntax

```
__property CC_UINT MaxOutputBlkSize;
```

OutputPolicy

The output policy for multiplexer.

Syntax

```
__property CC_MUX_OUTPUT_POLICY OutputPolicy;
```

RateMode

Target bitrate mode - constant or variable.

Syntax

```
__property CC_BITRATE_MODE RateMode;
```

StreamType

Target multiplexed stream type.

Syntax

```
__property CC_MULTIPLEXED_STREAM_TYPE StreamType;
```

TrailingAlignment

The trailing stream alignment, specified in bytes. (1000 means 1000-bytes alignment of the stream).

Syntax

```
__property CC_UINT TrailingAlignment;
```

CC_ACCESS_UNIT_DESCR

Description of the multiplexed elementary stream data unit.

Syntax

```
struct CC_ACCESS_UNIT_DESCR {
  DWORD size;
  DWORD offset;
  CC_TIME pts;
  CC_TIME dts;
};
```

Members

size

Size (in bytes) of the elementary stream unit.

offset

Offset (in bytes) of the first byte of the elementary data unit. From the beginning of the transmitted packet.

pts

The Presentation Time Stamp (PTS) of the elementary data unit.

dts

The Decoding Time Stamp (PTS) of the elementary data unit.

CC_ELEMENTARY_STREAM_TYPE

List of the elementary streams types (see ISO/IEC 13818-1 2.5.4.2 "Semantic definition of fields in Program Stream map").

Syntax

```
[v1_enum]
enum CC_ELEMENTARY_STREAM_TYPE {
  CC_ES_TYPE_UNKNOWN = 0x00,
  CC_ES_TYPE_VIDEO_MPEG1 = 0x01,
  CC_ES_TYPE_VIDEO_MPEG2 = 0x02,
  CC_ES_TYPE_AUDIO_MPEG1 = 0x03,
  CC_ES_TYPE_AUDIO_MPEG2 = 0x04,
  CC_ES_TYPE_PRIVATE_SECTION = 0x05,
  CC_ES_TYPE_PRIVATE_DATA = 0x06,
  CC_ES_TYPE_MHEG = 0x07,
  CC_ES_TYPE_DSM_CC = 0x08,
  CC_ES_TYPE_H_222_1 = 0x09,
  CC_ES_TYPE_13818_6_A = 0x0A,
  CC_ES_TYPE_13818_6_B = 0x0B,
  CC_ES_TYPE_13818_6_C = 0x0C,
  CC_ES_TYPE_13818_6_D = 0x0D,
  CC_ES_TYPE_13818_1_AUX = 0x0E,
  CC_ES_TYPE_AUDIO_AAC = 0x0f,
  CC_ES_TYPE_VIDEO_MPEG4 = 0x10,
  CC_ES_TYPE_AUDIO_LATM = 0x11,
  CC_ES_TYPE_FLEXMUX_1 = 0x12,
  CC_ES_TYPE_FLEXMUX_2 = 0x13,
  CC_ES_TYPE_SYNC_DOWNLOAD_PROTOCOL = 0x14,
  CC_ES_TYPE_VIDEO_H264 = 0x1b,
  CC_ES_TYPE_VIDEO_AVC = 0x1b,
  CC_ES_TYPE_VIDEO_J2K = 0x21,
  CC_ES_TYPE_VIDEO_H265 = 0x24,
  CC_ES_TYPE_VIDEO_HEVC = 0x24,
  CC_ES_TYPE_AUDIO_LPCM = 0x80,
  CC_ES_TYPE_AUDIO_AC3 = 0x81,
  CC_ES_TYPE_AUDIO_AC3_ATSC = 0x81,
  CC_ES_TYPE_AUDIO_AC3_DVB = 0x8106,
  CC_ES_TYPE_SCTE_35 = 0x86,
  CC_ES_TYPE_AUDIO_DTS = 0x8a,
  CC_ES_PRESENTATION_GRAPHICS = 0x90,
  CC_ES_INTERACTIVE_GRAPHICS = 0x91,
  CC_ES_TYPE_AUDIO_AES3 = 0x98,
  CC_ES_TYPE_DATA_AES3 = 0x99,
  CC_ES_TYPE_AUDIO_SMPTE302 = 0x9806,
  CC_ES_TYPE_AUDIO_DOLBY_E = 0x9906,
  CC_ES_TYPE_HDV2_AUX_A = 0xA0,
  CC_ES_TYPE_HDV2_AUX_V = 0xA1,
  CC_ES_TYPE_VIDEO_DV = 0x1d0,
  CC_ES_TYPE_VIDEO_DVCPRO = 0x1d1,
  CC_ES_TYPE_VIDEO_DNxHD = 0x1d2,
  CC_ES_TYPE_VIDEO_AVC_INTRA = 0x11b,
  CC_ES_TYPE_VIDEO_PRORES = 0x120,
  CC_ES_TYPE_EBU_TELETEXT = 0x4206,
  CC_ES_TYPE_DVB_SUBTITLES = 0xBF06,
  CC_ES_TYPE_SMPTE_436 = 0x436,
  CC_ES_TYPE_VIDEO_DANIEL = 0xD206,
  CC_ES_TYPE_VIDEO_Y4M = 0xF406,
  CC_ES_TYPE_VIDEO_UNCOMPRESSED = 0xFF06
};
```

Members

CC_ES_TYPE_UNKNOWN

ITU-T | ISO/IEC Reserved.

CC_ES_TYPE_VIDEO_MPEG1

ISO/IEC 11172 Video.

CC_ES_TYPE_VIDEO_MPEG2

ITU-T Rec. H.262 | ISO/IEC 13818-2 Video or ISO/IEC 11172-2 constrained parameter video stream.

CC_ES_TYPE_AUDIO_MPEG1

ISO/IEC 11172 Audio.

CC_ES_TYPE_AUDIO_MPEG2

ISO/IEC 13818-3 Audio.

CC_ES_TYPE_PRIVATE_SECTION

ITU-T Rec. H.222.0 | ISO/IEC 13818-1 private_sections.

CC_ES_TYPE_PRIVATE_DATA

ITU-T Rec. H.222.0 | ISO/IEC 13818-1 PES packets containing private data.

CC_ES_TYPE_MHEG

ISO/IEC 13522 MHEG.

CC_ES_TYPE_DSM_CC

ITU-T Rec. H.222.0 | ISO/IEC 13818-1 Annex A DSM CC.

CC_ES_TYPE_H_222_1

ITU-T Rec. H.222.1.

CC_ES_TYPE_13818_6_A

ISO/IEC 13818-6 type A.

CC_ES_TYPE_13818_6_B

ISO/IEC 13818-6 type B.

CC_ES_TYPE_13818_6_C

ISO/IEC 13818-6 type C.

CC_ES_TYPE_13818_6_D

ISO/IEC 13818-6 type D.

CC_ES_TYPE_13818_1_AUX

ISO/IEC 13818-1 auxiliary.

CC_ES_TYPE_AUDIO_AAC

ISO/IEC 13818-7 Audio with ADTS transport syntax

CC_ES_TYPE_VIDEO_MPEG4
ISO/IEC 14496-2 Visual (H.263 Video)

CC_ES_TYPE_AUDIO_LATM
ISO/IEC 14496-3 Audio with the LATM transport syntax as defined in ISO/IEC 14496-3 / AMD 1

CC_ES_TYPE_FLEXMUX_1
ISO/IEC 14496-1 SL-packetized stream or FlexMux stream carried in PES packets

CC_ES_TYPE_FLEXMUX_2
ISO/IEC 14496-1 SL-packetized stream or FlexMux stream carried in ISO/IEC14496_sections

CC_ES_TYPE_SYNC_DOWNLOAD_PROTOCOL
ISO/IEC 13818-6 Synchronized Download Protocol

CC_ES_TYPE_VIDEO_H264
H.264 Video

CC_ES_TYPE_VIDEO_AVC
H.264 Video

CC_ES_TYPE_VIDEO_J2K
Jpeg-2000 elementary stream

CC_ES_TYPE_VIDEO_H265
H.265/HEVC Video

CC_ES_TYPE_VIDEO_HEVC
H.265/HEVC Video

CC_ES_TYPE_AUDIO_LPCM
LPCM Audio

CC_ES_TYPE_AUDIO_AC3
AC3 Audio

CC_ES_TYPE_AUDIO_AC3_ATSC
AC3 Audio (system A)

CC_ES_TYPE_AUDIO_AC3_DVB
AC3 Audio (system B) carried by stream type 06

CC_ES_TYPE_SCTE_35
SCTE-35 splice messages

CC_ES_TYPE_AUDIO_DTS
DTS Audio

CC_ES_PRESENTATION_GRAPHICS
BluRay PGS (subtitles)

CC_ES_INTERACTIVE_GRAPHICS
BluRay IGS

CC_ES_TYPE_AUDIO_AES3
SMPTE-302M Audio.

CC_ES_TYPE_DATA_AES3
SMPTE-302M Data.

CC_ES_TYPE_AUDIO_SMPTE302
SMPTE-302M Audio (standard 06 stream type).

CC_ES_TYPE_AUDIO_DOLBY_E
SMPTE-302M Data carried by stream type 06.

CC_ES_TYPE_HDV2_AUX_A
HDV-2 Auxillary Audio Stream.

CC_ES_TYPE_HDV2_AUX_V
HDV-2 Auxillary Video Stream.

CC_ES_TYPE_VIDEO_DV
DV Video.

CC_ES_TYPE_VIDEO_DVCPRO
DVCPRO Video.

CC_ES_TYPE_VIDEO_DNxHD
VC-3 Video (DNxHD Codec).

CC_ES_TYPE_VIDEO_AVC_INTRA
AVC-Intra Video

CC_ES_TYPE_VIDEO_PRORES
Apple ProRes Video

CC_ES_TYPE_EBU_TELETEXT
ITU-R System B Teletext (OP42) carried by stream type 06.

CC_ES_TYPE_DVB_SUBTITLES
REN/JTC-DVB-191 DVB Subtitles

CC_ES_TYPE_SMPTE_436
SMPTE 436 Encoded VANC data

CC_ES_TYPE_VIDEO_DANIEL
Daniel Video Stream.

CC_ES_TYPE_VIDEO_Y4M
Y4M Video Stream

CC_ES_TYPE_VIDEO_UNCOMPRESSED
Uncompressed Video

CC_MULTIPLEXED_STREAM_TYPE

The stream types.

Syntax

```
[v1_enum]
enum CC_MULTIPLEXED_STREAM_TYPE {
  CC_MUX_UNKNOWN_STREAM = 0,
  CC_MUX_ELEMENTARY_STREAM = 1,
  CC_MUX_PES_STREAM = 2,
  CC_MUX_SYSTEM_STREAM = 3,
  CC_MUX_PROGRAM_STREAM = 4,
  CC_MUX_TRANSPORT_STREAM = 5,
  CC_MUX_MP4_STREAM = 6
};
```

CC_MUX_OUTPUT_POLICY

Syntax

```
enum CC_MUX_OUTPUT_POLICY {
  CC_FLUSH_EACH_PACKET = 0,
  CC_FLUSH_AT_ACCESS_UNIT_START = 1,
  CC_FLUSH_AT_BUFFER_FULL = 2,
  CC_MUX_OUTPUT_POLICIES_COUNT
};
```

CC_PACKET_DESCR

Description of the multiplexed packet.

Syntax

```
struct CC_PACKET_DESCR {
  CC_PES_ID pes_id;
  CC_PID pid;
  CC_SCR scr;
  CC_UINT duration;
  CC_UINT size;
  CC_UINT offset;
};
```

Members

pes_id

PES ID of the stream.

pid

PID of the stream.

scr

The stream clock reference of the first byte of the packet.

duration

Duration of the packet in CC_PCR units.

size

Packet size in bytes.

offset

Offset in bytes.

CC_PES_ID

A list of the elementary stream IDs. Refer to table 2-18 of ISO/IEC 13818-1 2.4.3.7 "Semantic definition of fields in PES packets".

Syntax

```
enum CC_PES_ID {
  CC_PESID_UNKNOWN = 0x00,
  CC_PESID_MIN = 0xBC,
  CC_PESID_PROGRAM_STREAM_MAP = 0xBC,
  CC_PESID_PRIVATE_1 = 0xBD,
  CC_PESID_PADDING = 0xBE,
  CC_PESID_PRIVATE_2 = 0xBF,
  CC_PESID_AUDIO = 0xC0,
  CC_PESID_VIDEO = 0xE0,
  CC_PESID_MPEG1_DATA = 0xF0,
  CC_PESID_ECM = 0xF0,
  CC_PESID_EMM = 0xF1,
  CC_PESID_DSMCC = 0xF2,
  CC_PESID_ISO_IEC_13522 = 0xF3,
  CC_PESID_H222_1_TYPE_A = 0xF4,
  CC_PESID_H222_1_TYPE_B = 0xF5,
  CC_PESID_H222_1_TYPE_C = 0xF6,
  CC_PESID_H222_1_TYPE_D = 0xF7,
  CC_PESID_H222_1_TYPE_E = 0xF8,
  CC_PESID_ANCILLARY = 0xF9,
  CC_PESID_MPEG4_SL_PACKETIZED = 0xFA,
  CC_PESID_MPEG4_FLEX_MUX = 0xFB,
  CC_PESID_METADATA = 0xFC,
  CC_PESID_EXTENDED = 0xFD,
  CC_PESID_RESERVED = 0xFE,
  CC_PESID_PROGRAM_STREAM_DIRECTORY = 0xFF,
  CC_PESID_MAX = 0xFF
};
```

Members

CC_PESID_UNKNOWN

Unknown stream type.

CC_PESID_AUDIO

32 different stream_id allowed in range 0xC0..0xDF.

CC_PESID_VIDEO

16 different stream_id allowed in range 0xE0..0xEF.

CC_PSI_TABLE_ID

A list of the standard PSI Table ID's. Refer to the ISO/IEC 13818-1 2.4.4 "Program specific information".

Syntax

```
enum CC_PSI_TABLE_ID {
  CC_PSITBL_PROGRAM_ASSOCIATION_SECTION = 0x00,
  CC_PSITBL_CONDITIONAL_ACCESS_SECTION = 0x01,
  CC_PSITBL_PROGRAM_MAP_SECTION = 0x02,
  CC_PSITBL_DESCRIPTION_SECTION = 0x03,
  CC_PSITBL_SCENE_DESCRIPTION_SECTION = 0x04,
  CC_PSITBL_OBJECT_DESCRIPTOR_SECTION = 0x05
};
```

Members

CC_PSITBL_PROGRAM_ASSOCIATION_SECTION
ISO/IEC 13818-1 Transport Stream.

CC_PSITBL_CONDITIONAL_ACCESS_SECTION
ISO/IEC 13818-1 Transport Stream.

CC_PSITBL_PROGRAM_MAP_SECTION
ISO/IEC 13818-1 Program & Transport Streams.

CC_PSITBL_DESCRIPTION_SECTION
ISO/IEC 13818-1 Program & Transport Streams.

CC_PSITBL_SCENE_DESCRIPTION_SECTION
ISO/IEC 14496.

CC_PSITBL_OBJECT_DESCRIPTOR_SECTION
ISO/IEC 14496.

MPEG_SYSTEM_DESCRIPTOR_TAG

Syntax

```
enum MPEG_SYSTEM_DESCRIPTOR_TAG {
  CC_DESCR_UNKNOWN = 0x00,
  CC_DESCR_VIDEO_STREAM = 0x02,
  CC_DESCR_AUDIO_STREAM = 0x03,
  CC_DESCR_HIERARCHY = 0x04,
  CC_DESCR_REGISTRATION = 0x05,
  CC_DESCR_DATA_STREAM_ALIGNMENT = 0x06,
  CC_DESCR_TARGET_BACKGROUND_GRID = 0x07,
  CC_DESCR_VIDEO_WINDOW = 0x08,
  CC_DESCR_CA = 0x09,
  CC_DESCR_ISO_639_LANGUAGE = 0x0A,
  CC_DESCR_SYSTEM_CLOCK = 0x0B,
  CC_DESCR_MULTIPLEX_BUFFER_UTILIZATION = 0x0C,
  CC_DESCR_COPYRIGHT = 0x0D,
  CC_DESCR_MAXIMUM_BITRATE = 0x0E,
  CC_DESCR_PRIVATE_DATA_INDICATOR = 0x0F,
  CC_DESCR_SMOOTHING_BUFFER = 0x10,
  CC_DESCR_STD = 0x11,
  CC_DESCR_IBP = 0x12,
  CC_DESCR_MPEG4_VIDEO = 0x1B,
  CC_DESCR_MPEG4_AUDIO = 0x1C,
  CC_DESCR_IOD = 0x1D,
  CC_DESCR_SL = 0x1E,
  CC_DESCR_FMC = 0x1F,
  CC_DESCR_EXTERNAL_ES_ID = 0x20,
  CC_DESCR_MUX_CODE = 0x21,
  CC_DESCR_FMX_BUFFER_SIZE = 0x22,
  CC_DESCR_MULTIPLEX_BUFFER = 0x23,
  CC_DESCR_CONTENT_LABELING = 0x24,
  CC_DESCR_METADATA_POINTER = 0x25,
  CC_DESCR_METADATA = 0x26,
  CC_DESCR_METADATA_STD = 0x27,
  CC_DESCR_AVC_VIDEO = 0x28,
  CC_DESCR_IPMP = 0x29,
  CC_DESCR_AVC_TIMING_AND_HDR = 0x30,
  CC_DESCR_NETWORK_NAME = 0x40,
  CC_DESCR_SERVICE_LIST = 0x41,
  CC_DESCR_STUFFING = 0x42,
  CC_DESCR_SATELLITE_DELIVERY_SYSTEM = 0x43,
  CC_DESCR_CABLE_DELIVERY_SYSTEM = 0x44,
  CC_DESCR_VBI_DATA = 0x45,
  CC_DESCR_VBI_TELETEXT = 0x46,
  CC_DESCR_BOUQUET_NAME = 0x47,
  CC_DESCR_SERVICE = 0x48,
  CC_DESCR_COUNTRY_AVAILABILITY = 0x49,
  CC_DESCR_LINKAGE = 0x4A,
  CC_DESCR_NVOD_REFERENCE = 0x4B,
  CC_DESCR_TIME_SHIFTED_SERVICE = 0x4C,
  CC_DESCR_SHORT_EVENT = 0x4D,
  CC_DESCR_EXTENDED_EVENT = 0x4E,
  CC_DESCR_TIME_SHIFTED_EVENT = 0x4F,
  CC_DESCR_COMPONENT = 0x50,
  CC_DESCR_MOSAIC = 0x51,
  CC_DESCR_STREAM_IDENTIFIER = 0x52,
  CC_DESCR_CA_IDENTIFIER = 0x53,
  CC_DESCR_CONTENT = 0x54,
  CC_DESCR_PARENTAL_RATING = 0x55,
  CC_DESCR_TELETEXT = 0x56,
  CC_DESCR_TELEPHONE = 0x57,
```

```
  CC_DESCR_LOCAL_TIME_OFFSET = 0x58,
  CC_DESCR_SUBTITLING = 0x59,
  CC_DESCR_TERRESTRIAL_DELIVERY_SYSTEM = 0x5A,
  CC_DESCR_MULTILINGUAL_NETWORK_NAME = 0x5B,
  CC_DESCR_MULTILINGUAL_BOUQUET_NAME = 0x5C,
  CC_DESCR_MULTILINGUAL_SERVICE_NAME = 0x5D,
  CC_DESCR_MULTILINGUAL_COMPONENT = 0x5E,
  CC_DESCR_PRIVATE_DATA_SPECIFIER = 0x5F,
  CC_DESCR_SERVICE_MOVE = 0x60,
  CC_DESCR_SHORT_SMOOTHING_BUFFER = 0x61,
  CC_DESCR_FREQUENCY_LIST = 0x62,
  CC_DESCR_PARTIAL_TRANSPORT_STREAM = 0x63,
  CC_DESCR_DATA_BROADCAST = 0x64,
  CC_DESCR_SCRAMBLING = 0x65,
  CC_DESCR_DATA_BROADCAST_ID = 0x66,
  CC_DESCR_TRANSPORT_STREAM = 0x67,
  CC_DESCR_DSNG = 0x68,
  CC_DESCR_PDC = 0x69,
  CC_DESCR_AC3_SYSTEM_B_DVB = 0x6A,
  CC_DESCR_ANCILLARY_DATA = 0x6B,
  CC_DESCR_CELL_LIST = 0x6C,
  CC_DESCR_CELL_FREQUENCY_LINK = 0x6D,
  CC_DESCR_ANNOUNCEMENT_SUPPORT = 0x6E,
  CC_DESCR_APPLICATION_SIGNALLING = 0x6F,
  CC_DESCR_ADAPTATION_FIELD_DATA = 0x70,
  CC_DESCR_SERVICE_IDENTIFIER = 0x71,
  CC_DESCR_SERVICE_AVAILABILITY = 0x72,
  CC_DESCR_DEFAULT_AUTHORITY = 0x73,
  CC_DESCR_RELATED_CONTENT = 0x74,
  CC_DESCR_TVA_ID = 0x75,
  CC_DESCR_CONTENT_IDENTIFIER = 0x76,
  CC_DESCR_TIME_SLICE_FEC_IDENTIFIER = 0x77,
  CC_DESCR_ECM_REPETITION_RATE = 0x78,
  CC_DESCR_S2_SATELLITE_DELIVERY_SYSTEM = 0x79,
  CC_DESCR_ENHANCED_AC3 = 0x7A,
  CC_DESCR_DTS = 0x7B,
  CC_DESCR_AAC = 0x7C,
  CC_DESCR_XAIT_LOCATION = 0x7D,
  CC_DESCR_FTA_CONTENT_MANAGEMENT = 0x7E,
  CC_DESCR_EXTENSION = 0x7F,
  CC_DESCR_AC3_SYSTEM_A_ATSC = 0x81,
  CC_DESCR_BROADCAST_ID = 0x85,
  CC_DESCR_DTCP = 0x88,
  CC_DESCR_CUE_IDENTIFIER = 0x8A,
  CC_DESCR_HIERARCHICAL_TRANSMISSION = 0xC0,
  CC_DESCR_DIGITAL_COPY_CONTROL = 0xC1,
  CC_DESCR_NETWORK_IDENTIFICATION = 0xC2,
  CC_DESCR_PARTIAL_TS_TIME = 0xC3,
  CC_DESCR_AUDIO_COMPONENT = 0xC4,
  CC_DESCR_HYPERLINK = 0xC5,
  CC_DESCR_TARGET_REGION = 0xC6,
  CC_DESCR_DATA_CONTENT = 0xC7,
  CC_DESCR_VIDEO_DECODE_CONTROL = 0xC8,
  CC_DESCR_TS_INFORMATION = 0xCD,
  CC_DESCR_EXTENDED_BROADCASTER = 0xCE,
  CC_DESCR_SERIES = 0xD5,
  CC_DESCR_EVENT_GROUP = 0xD6,
  CC_DESCR_BROADCASTER_NAME = 0xD8,
  CC_DESCR_COMPONENT_GROUP = 0xD9,
  CC_DESCR_CONTENT_AVAILABILITY = 0xDE,
  CC_DESCR_EMERGENCY_INFORMATION = 0xFC,
  CC_DESCR_DATA_COMPONENT = 0xFD
};
```

MPEG-2 codec

This section describes the classes, interfaces and data types specific for the set of MPEG formats (specifications ISO/IEC 11172 and ISO/IEC 13818) and also for some of their derivatives.

(From Wikipedia, the free encyclopedia)

MPEG-2 is a standard for the generic coding of moving pictures and associated audio information. It describes a combination of lossy video compression and lossy audio data compression methods which permit storage and transmission of movies using currently available storage media and transmission bandwidth.

MPEG-2 is widely used as the format of digital television signals. It also specifies the format of movies and other programs that are distributed on DVD and similar discs. As such, TV stations, TV receivers, DVD players, and other equipment are often designed to this standard. MPEG-2 was the second of several standards developed by the Moving Pictures Expert Group (MPEG) and is an international standard (ISO/IEC 13818).

While MPEG-2 is the core of most digital television and DVD formats, it does not completely specify them. Regional institutions can adapt it to their needs by restricting and augmenting aspects of the standard.

MPEG-2 includes a Systems section, part 1, that defines two distinct, but related, container formats. One is the MPEG transport stream, designed to carry digital video and audio over possibly lossy media, where the beginning and the end of the stream may not be identified, such as broadcasting or magnetic tape, examples of which include ATSC, DVB, SBTVD and HDV. MPEG-2 Systems also defines the MPEG program stream, a container format designed for file-based media such as hard disk drives, optical discs and flash memory.

The Video section, part 2 of MPEG-2, is similar to the previous MPEG-1 standard, but also provides support for interlaced video, the format used by analog broadcast TV systems. MPEG-2 video is not optimized for low bit-rates, especially less than 1 Mbit/s at standard definition resolutions. All standards-compliant MPEG-2 Video decoders are fully capable of playing back MPEG-1 Video streams conforming to the Constrained Parameters Bitstream syntax. MPEG-2/Video is formally known as ISO/IEC 13818-2 and as ITU-T Rec. H.262.

The MPEG-2 Audio section, defined in part 3 of the standard, enhances MPEG-1's audio by

allowing the coding of audio programs with more than two channels, up to 5.1 multichannel. This method is backwards-compatible, allowing MPEG-1 audio decoders to decode the two main stereo components of the presentation. MPEG-2 part 3 also defined additional bit rates and sample rates for MPEG-1 Audio Layer I, II and III.

MPEG-2 video supports a wide range of applications from mobile to high quality HD editing. For many applications, it's unrealistic and too expensive to support the entire standard. To allow such applications to support only subsets of it, the standard defines profile and level.

The profile defines the subset of features such as compression algorithm, chroma format, etc. The level defines the subset of quantitative capabilities such as maximum bit rate, maximum frame size, etc.

A MPEG application then specifies the capabilities in terms of profile and level. For example, a DVD player may say it supports up to main profile and main level (often written as MP@ML). It means the player can play back any MPEG stream encoded as MP@ML or less.

The section consist of three major parts - MPEG Audio, MPEG Video and MPEG Multiplex.

MPEG Video

This section contains descriptions of interfaces and types to work with MPEG video stream.

Types

Enumerations

	Name	Description
	CC_MPG_FRAME_FLAGS (see page 176)	
	CC_MPG_INTRA_VLC_TABLE (see page 177)	The intra_vlc_format value. Refer ISO/IEC 13818-2 7.2.2.1 for details.
	CC_MPG_MB_SCAN_PATTERN (see page 177)	The block's scan patterns.
	CC_MPG_PROFILE_LEVEL (see page 178)	
	CC_MPG_QUANT_SCALE_TYPE (see page 179)	
	CC_MPG_ASPECT_RATIO_CODE (see page 180)	Frame aspect ratios.
	CC_MPG_MOTION_PARAMS (see page 180)	

CC_MPG_FRAME_FLAGS

Syntax

```
[v1_enum]
enum CC_MPG_FRAME_FLAGS {
  CC_MPG_FRAME_FLG_PROGRESSIVE_FRAME = 0x00000001,
  CC_MPG_FRAME_FLG_TOP_FIELD_FIRST = 0x00000002,
  CC_MPG_FRAME_FLG_REPEAT_FIRST_FIELD = 0x00000004,
  CC_MPG_FRAME_FLG_ALTERNATE_SCAN = 0x00000008,
  CC_MPG_FRAME_FLG_Q_SCALE_TYPE = 0x00000010,
  CC_MPG_FRAME_FLG_FRAME_PRED_DCT = 0x00000020,
  CC_MPG_FRAME_FLG_INTRA_VLC_FORMAT = 0x00000040,
  CC_MPG_FRAME_FLG_CONCEALMENT_VEC = 0x00000080,
  CC_MPG_FRAME_FLG_FULL_PEL_FORW_VEC = 0x00000100,
  CC_MPG_FRAME_FLG_FULL_PEL_BACK_VEC = 0x00000200,
  CC_MPG_FRAME_FLG_CHROMA_420_TYPE = 0x00000400,
  CC_MPG_FRAME_FLG_COMPOSITE_DISPLAY = 0x00000800,
  CC_GOP_CLOSED = 0x00010000,
  CC_GOP_BROKEN_LINK = 0x00020000,
  CC_GOP_DROP_FRAME = 0x00040000,
  CC_MPG_HDR_SEQUENCE = 0x01000000,
  CC_MPG_HDR_GOP = 0x02000000,
  CC_MPG_HDR_SEQUENCE_EXT = 0x04000000,
  CC_MPG_HDR_SEQUENCE_DISP_EXT = 0x08000000,
  CC_MPG_HDR_QUANT_MATRIX = 0x10000000,
  CC_MPG_HDR_PICTURE_DISP_EXT = 0x20000000,
```

```
  CC_MPG_HDR_PICTURE_CODING_EXT = 0x40000000
};
```

CC_MPG_INTRA_VLC_TABLE

The intra_vlc_format value. Refer ISO/IEC 13818-2 7.2.2.1 for details.

Syntax

```
[v1_enum]
enum CC_MPG_INTRA_VLC_TABLE {
  CC_MPG_INTRA_VLC_TABLE_AUTO = 0,
  CC_MPG_INTRA_VLC_TABLE_B14,
  CC_MPG_INTRA_VLC_TABLE_B15,
  CC_MPG_INTRA_VLC_TABLE_COUNT
};
```

Members

CC_MPG_INTRA_VLC_TABLE_AUTO

Automatic selection of the intra_vlc_table.

CC_MPG_INTRA_VLC_TABLE_B14

Use VLC table B.14 for intra coeffs.

CC_MPG_INTRA_VLC_TABLE_B15

Use VLC table B.15 for intra coeffs.

CC_MPG_MB_SCAN_PATTERN

The block's scan patterns.

Syntax

```
[v1_enum]
enum CC_MPG_MB_SCAN_PATTERN {
  CC_MPG_MB_SCAN_AUTO = 0,
  CC_MPG_MB_SCAN_ZIGZAG,
  CC_MPG_MB_SCAN_ALTERNATE,
  CC_MPG_MB_SCAN_COUNT
};
```

Members

CC_MPG_MB_SCAN_AUTO

Automatic selection of the block scan pattern.

CC_MPG_MB_SCAN_ZIGZAG

ZigZag scan pattern.

CC_MPG_MB_SCAN_ALTERNATE

Alternate scan pattern.

CC_MPG_PROFILE_LEVEL

Syntax

```
[v1_enum]
enum CC_MPG_PROFILE_LEVEL {
  CC_MPG_PROFILE_LEVEL_UNKNOWN = 0,
  CC_MPEG1_CONSTRAINED = 0x01,
  CC_MPEG1_ESCAPE = 0X81,
  CC_MPEG2_SP_ML = 0x58,
  CC_MPEG2_MP_LL = 0x4A,
  CC_MPEG2_MP_ML = 0x48,
  CC_MPEG2_MP_H14 = 0x46,
  CC_MPEG2_MP_HL = 0x44,
  CC_MPEG2_422_ML = 0x85,
  CC_MPEG2_422_HL = 0x82,
  CC_MPEG2_SNR_LL = 0x3A,
  CC_MPEG2_SNR_ML = 0x38,
  CC_MPEG2_SPAT_H14 = 0x26,
  CC_MPEG2_HP_ML = 0x18,
  CC_MPEG2_HP_H14 = 0x16,
  CC_MPEG2_HP_HL = 0x14,
  CC_MPEG2_MVP_LL = 0x8E,
  CC_MPEG2_MVP_ML = 0x8D,
  CC_MPEG2_MVP_H14 = 0x8B,
  CC_MPEG2_MVP_HL = 0x8A,
  CC_MPEG2_ESCAPE = 0x80,
  CC_MPG_SP_ML = CC_MPEG2_SP_ML,
  CC_MPG_MP_LL = CC_MPEG2_MP_LL,
  CC_MPG_MP_ML = CC_MPEG2_MP_ML,
  CC_MPG_MP_H14 = CC_MPEG2_MP_H14,
  CC_MPG_MP_HL = CC_MPEG2_MP_HL,
  CC_MPG_HP_ML = CC_MPEG2_HP_ML,
  CC_MPG_HP_H14 = CC_MPEG2_HP_H14,
  CC_MPG_HP_HL = CC_MPEG2_HP_HL,
  CC_MPG_ESCAPE = CC_MPEG2_ESCAPE,
  CC_MPG_422_ML = CC_MPEG2_422_ML,
  CC_MPG_422_HL = CC_MPEG2_422_HL
};
```

Members

CC_MPG_PROFILE_LEVEL_UNKNOWN

Unknown profile&level.

CC_MPEG2_SP_ML

Simple Profile @ Main Level

CC_MPEG2_MP_LL

Main Profile @ Low level

CC_MPEG2_MP_ML

Main Profile @ Main level

CC_MPEG2_MP_H14

Main Profile @ High1440 level

CC_MPEG2_MP_HL

Main Profile @ High level

CC_MPEG2_422_ML
4:2:2 Profile @ Main level

CC_MPEG2_422_HL
4:2:2 Profile @ High level

CC_MPEG2_SNR_LL
SNR Scalable Profile @ Low level

CC_MPEG2_SNR_ML
SNR Scalable Profile @ Main level

CC_MPEG2_SPAT_H14
Spatial Scalable Profile @ High1440 level

CC_MPEG2_HP_ML
High Profile @ Main level

CC_MPEG2_HP_H14
High Profile @ High1440 level

CC_MPEG2_HP_HL
High Profile @ High level

CC_MPEG2_MVP_LL
Multi View Profile @ Low level

CC_MPEG2_MVP_ML
Multi View Profile @ Main level

CC_MPEG2_MVP_H14
Multi View Profile @ High1440 level

CC_MPEG2_MVP_HL
Multi View Profile @ High level

CC_MPEG2_ESCAPE
Avoid any limitations (free coding).

CC_MPG_QUANT_SCALE_TYPE

Syntax

```
[v1_enum]
enum CC_MPG_QUANT_SCALE_TYPE {
  CC_MPG_QSCALE_AUTO = 0,
  CC_MPG_QSCALE_LINEAR,
  CC_MPG_QSCALE_NON_LINEAR,
  CC_MPG_QSCALE_COUNT
};
```

Members

CC_MPG_QSCALE_AUTO

Automatic selection of the quantization scale.

CC_MPG_QSCALE_LINEAR

Linear quant scales (1..31).

CC_MPG_QSCALE_NON_LINEAR

Non-linear quant scales (0.5..56).

CC_MPG_ASPECT_RATIO_CODE

Frame aspect ratios.

Syntax

```
[v1_enum]
enum CC_MPG_ASPECT_RATIO_CODE {
  CC_ASPECT_RATIO_UNKNOWN = 0,
  CC_ASPECT_RATIO_VGA = 1,
  CC_ASPECT_RATIO_1_1 = 1,
  CC_ASPECT_RATIO_4_3 = 2,
  CC_ASPECT_RATIO_16_9 = 3
};
```

Members

CC_ASPECT_RATIO_UNKNOWN

Forbidden value.

CC_ASPECT_RATIO_VGA

1:1 - square sample.

CC_ASPECT_RATIO_1_1

1:1 - square sample.

CC_ASPECT_RATIO_4_3

4:3 - commonly used for both mpeg1&2.

CC_ASPECT_RATIO_16_9

16:9 - commonly used for both mpeg1&2.

CC_MPG_MOTION_PARAMS

Syntax

```
[v1_enum]
enum CC_MPG_MOTION_PARAMS {
  CC_MPG_ME_SWX_16 = 0x00000001,
  CC_MPG_ME_SWX_32 = 0x00000000,
  CC_MPG_ME_SWX_48 = 0x00000002,
  CC_MPG_ME_SWX_64 = 0x00000003,
```

```
  CC_MPG_ME_SWX_96 = 0x00000004,
  CC_MPG_ME_SWX_128 = 0x00000005,
  CC_MPG_ME_SWX_192 = 0x00000006,
  CC_MPG_ME_SWX_256 = 0x00000007,
  CC_MPG_ME_SWX_MASK = 0x0000000F,
  CC_MPG_ME_SWY_16 = 0x00000010,
  CC_MPG_ME_SWY_32 = 0x00000000,
  CC_MPG_ME_SWY_48 = 0x00000020,
  CC_MPG_ME_SWY_64 = 0x00000030,
  CC_MPG_ME_SWY_96 = 0x00000040,
  CC_MPG_ME_SWY_128 = 0x00000050,
  CC_MPG_ME_SWY_192 = 0x00000060,
  CC_MPG_ME_SWY_256 = 0x00000070,
  CC_MPG_ME_SWY_MASK = 0x000000F0,
  CC_MPG_ME_WND_16 = CC_MPG_ME_SWX_16|CC_MPG_ME_SWY_16,
  CC_MPG_ME_WND_32 = CC_MPG_ME_SWX_32|CC_MPG_ME_SWY_32,
  CC_MPG_ME_WND_64 = CC_MPG_ME_SWX_64|CC_MPG_ME_SWY_64,
  CC_MPG_ME_WND_96 = CC_MPG_ME_SWX_96|CC_MPG_ME_SWY_96,
  CC_MPG_ME_WND_128 = CC_MPG_ME_SWX_128|CC_MPG_ME_SWY_128,
  CC_MPG_ME_CELL1 = 0x00000100,
  CC_MPG_ME_CELL2 = 0x00000000,
  CC_MPG_ME_CELL4 = 0x00000200,
  CC_MPG_ME_CELL8 = 0x00000300,
  CC_MPG_ME_FUNC_MASK = 0x00000F00,
  CC_MPG_ME_SPC_1 = 0x00001000,
  CC_MPG_ME_SPC_2 = 0x00000000,
  CC_MPG_ME_SPC_4 = 0x00002000,
  CC_MPG_ME_SPC_MASK = 0x00003000,
  CC_MPG_ME_INCR_0 = 0x00004000,
  CC_MPG_ME_INCR_1 = 0x00000000,
  CC_MPG_ME_INCR_2 = 0x00008000,
  CC_MPG_ME_INCR_MASK = 0x0000C000,
  CC_MPG_ME_HALFPEL = 0x00000000,
  CC_MPG_ME_NO_HALFPEL = 0x00010000,
  CC_MPG_ME_QUICK_SAD = 0x00000000,
  CC_MPG_ME_NORMAL_SAD = 0x00020000,
  CC_MPG_ME_OPPOSITE_FLD = 0x00040000,
  CC_MPG_ME_ALIGN_VEC = 0x00080000,
  CC_MPG_ME_NO_INTERLACED_SEARCH = 0x00100000,
  CC_MPG_ME_NO_ADAPTIVE_SEARCH_WINDOW = 0x00200000,
  CC_MPG_ME_NO_PREDICTIVE_SEARCH = 0x00400000,
  CC_MPG_ME_NO_RECALC_MISPREDICTED_VECTORS = 0x00800000,
  CC_MPG_ME_FASTEST =
CC_MPG_ME_WND_16|CC_MPG_ME_CELL4|CC_MPG_ME_SPC_4|CC_MPG_ME_INCR_0|CC_MPG_ME_NO_HALFPEL,
  CC_MPG_ME_NORMAL =
CC_MPG_ME_WND_64|CC_MPG_ME_CELL2|CC_MPG_ME_SPC_2|CC_MPG_ME_INCR_1|CC_MPG_ME_HALFPEL|CC_MPG_ME_ALIG
N_VEC,
  CC_MPG_ME_BEST =
CC_MPG_ME_WND_128|CC_MPG_ME_CELL1|CC_MPG_ME_SPC_1|CC_MPG_ME_INCR_2|CC_MPG_ME_HALFPEL|CC_MPG_ME_ALI
GN_VEC|CC_MPG_ME_NORMAL_SAD|CC_MPG_ME_OPPOSITE_FLD
};
```

Members

CC_MPG_ME_SWX_16

-8..7 pels

CC_MPG_ME_SWX_32

-16..15

CC_MPG_ME_SWX_48

-24..23

CC_MPG_ME_SWX_64

-32..31

CC_MPG_ME_SWX_96

-48..47

CC_MPG_ME_SWX_128

-64..63

CC_MPG_ME_SWX_192

-96..95

CC_MPG_ME_SWX_256

-128..127

CC_MPG_ME_SWY_16

-8..7 pels

CC_MPG_ME_SWY_32

-16..15

CC_MPG_ME_SWY_48

-24..23

CC_MPG_ME_SWY_64

-32..31

CC_MPG_ME_SWY_96

-48..47

CC_MPG_ME_SWY_128

-64..63

CC_MPG_ME_SWY_192

-96..95

CC_MPG_ME_SWY_256

-128..127

CC_MPG_ME_CELL1

1:1 search cell

CC_MPG_ME_CELL2

2:2 search cell

CC_MPG_ME_CELL4

4:4 search cell

CC_MPG_ME_CELL8

8:8 search cell

CC_MPG_ME_SPC_1

Start from original frame.

CC_MPG_ME_SPC_2

Start from 2xdownsampled frame.

CC_MPG_ME_SPC_4

Start from 4xdownsampled frame.

CC_MPG_ME_INCR_0

only central pixel

CC_MPG_ME_INCR_1

- nearest neighbours

CC_MPG_ME_INCR_2

- 2x neighbours

CC_MPG_ME_HALFPEL

Using half-pel search (by default).

CC_MPG_ME_NO_HALFPEL

No half-pel search.

CC_MPG_ME_QUICK_SAD

Quick sad (default) uses 1/4 of block pixels, -20% of quality, +80% of speed.

CC_MPG_ME_NORMAL_SAD

Use normal sad.

CC_MPG_ME_OPPOSITE_FLD

Search opposite fields.

CC_MPG_ME_ALIGN_VEC

Aligned vectors is usually better then exact, but with small deviance.

CC_MPG_ME_NO_INTERLACED_SEARCH

This flag disables interlaced search, even if coding mode is interlaced

CC_MPG_ME_NO_ADAPTIVE_SEARCH_WINDOW

This flag disables adaptive search window

CC_MPG_ME_NO_PREDICTIVE_SEARCH

This flag disables predictive search (usu no prediction from neighboring blocks)

CC_MPG_ME_NO_RECALC_MISPREDICTED_VECTORS

This flag disables recalculation of mispredicted vectors

Interfaces

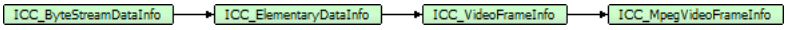
Interfaces

	Name	Description
	ICC_MpegVideoFrameInfo (see page 184)	Provides the particular MPEG video frame description.
	ICC_MpegVideoStreamInfo (see page 187)	Represents the MPEG-specified video stream description.

ICC_MpegVideoFrameInfo Interface

Provides the particular MPEG video frame description.

Class Hierarchy



Syntax

```
[object, uuid(00002204-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_MpegVideoFrameInfo : ICC_VideoFrameInfo;
```

Methods

ICC_MpegVideoFrameInfo Interface

	Name	Description
	GetUserData (see page 186)	Retrieves the specified user data which belongs to the video frame.

Properties

	Name	Description
R	DTS (see page 49)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 49), except the coded video with B-frames.
R	Duration (see page 49)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
R	NumSamples (see page 49)	Number of samples of the elementary data. In case of coded video frames, there is usually 1 or 2 sample(s) (1 frame or 2 fields). In case of coded audio, there are the number of audio samples, encoded into audio frame.

	PresentationDelta (see page 49)	The presentation delta, in samples, of the elementary data. It is actual for data which was reordered during encoding process (f.e. coded video with B-frames).
	PTS (see page 49)	The Presentation Data Stamp (PTS) of the elementary data. The PTS based on CC_TIMEBASE, specified for object which generates the elementary data.
	SampleOffset (see page 49)	The frame's first sample order number.
	SequenceEntryFlag (see page 50)	The sequence entry point flag. In the case of mpeg video, it means that SEQUENCE_HEADER presents in the elementary data. This flag is used to signal the multiplexers to generate the entry point (like System Header) at this elementary data.

ICC_VideoFrameInfo Interface

	Name	Description
	CodingNumber (see page 97)	The number of video frame in the coding order. Zero-based.
	Flags (see page 97)	Various flags of the coded video frame. Value of this field dependend of the video stream type.
	FrameType (see page 97)	The frame coding type (see page 107).
	InterlaceType (see page 98)	The field order of video frame.
	Number (see page 98)	The number of video frame in native (display) order. zero-based.
	PictStruct (see page 98)	The picture structure of the MPEG video frame.
	TimeCode (see page 98)	The timecode of video frame.

ICC_MpegVideoFrameInfo Interface

	Name	Description
	IntraDcPrec (see page 186)	The intra_dc_precision (8-11 bits)
	SecondFieldInfo (see page 186)	Retrieves the second field's information of a frame, if it exist (i.e. frame encoded in a field mode)
	TempRef (see page 187)	The frame's temporal reference.
	UserDataCount (see page 187)	The number of MPEG_USER_DATA, associated with the video frame.
	VBV_Delay (see page 187)	The VBV delay of the frame or field.

Methods

GetUserData

Retrieves the specified user data which belongs to the video frame.

Syntax

```
HRESULT GetUserData(
    [in] DWORD dwUserDataNumber,
    [out, size_is(cbBufSize)] BYTE * pData,
    [in] DWORD cbBufSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

dwUserDataNumber

Specified the user data number, zero-based.

pData

Place to store the user data, if NULL the only size of the specified user data will be returned.

cbBufSize

Buffer size.

pcbRetSize

Place to store the user data size.

Returns

Returns S_OK if successful or E_INVALIDARG in case of incorrect *dwUserDataNumber*.

Properties

IntraDcPrec

The intra_dc_precision (8-11 bits)

Syntax

```
__property DWORD * IntraDcPrec;
```

SecondFieldInfo

Retrieves the second field's information of a frame, if it exist (i.e. frame encoded in a field mode)

Syntax

```
__property ICC_MpegVideoFrameInfo ** SecondFieldInfo;
```

Returns

S_OK if ok, S_FALSE if no second_field exist

TempRef

The frame's temporal reference.

Syntax

```
__property DWORD * TempRef;
```

UserDataCount

The number of MPEG_USER_DATA, associated with the video frame.

Syntax

```
__property DWORD * UserDataCount;
```

VBV_Delay

The VBV delay of the frame or field.

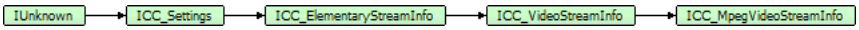
Syntax

```
__property DWORD * VBV_Delay;
```

ICC_MpegVideoStreamInfo Interface

Represents the MPEG-specified video stream description.

Class Hierarchy



Syntax

```
[object, uuid(00002202-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_MpegVideoStreamInfo : ICC_VideoStreamInfo;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_MpegVideoStreamInfo Interface

	Name	Description
	GetUserData (see page 189)	Retrieves the specified user data which associated with the stream (seq_hdr level).

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ElementaryStreamInfo Interface

	Name	Description
R	BitRate (see page 52)	The bitrate (max) of the elementary stream.
R	FrameRate (see page 52)	The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.
R	StreamType (see page 52)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

ICC_VideoStreamInfo Interface

	Name	Description
R	AspectRatio (see page 100)	The aspect ratio cx:cy.
R	FrameSize (see page 100)	The size in pixels of video frame.
R	ProgressiveSequence (see page 100)	If TRUE - all frames in the stream coded without fields (progressive).

ICC_MpegVideoStreamInfo Interface

	Name	Description
R	AspectRatioCode (see page 190)	The Aspect Ratio code.
R	BitRate (see page 190)	The bitrate of video bitstream.
R	ChromaFormat (see page 190)	Chroma format.
R	ColorCoefs (see page 190)	The color transformarion description.
R	DisplaySize (see page 190)	The Display Size (resulting frame size).

R	FrameDuration (see page 190)	The duration of one frame, in CC_TIMEBASE units.
R	Layer (see page 190)	The layer of MPEG video (1 or 2).
R	Mpeg1ConstrainedFlag (see page 190)	MPEG-1 constrained stream flag.
R	ProfileAndLevel (see page 191)	Profile and level.
R	UserDataCount (see page 191)	The number of MPEG_USER_DATA, associated with the video frame.
R	VBV_BufferSize (see page 191)	The VBV buffer size.
R	VideoFormat (see page 191)	The video format.

Methods

GetUserData

Retrieves the specified user data which associated with the stream (seq_hdr level).

Syntax

```
HRESULT GetUserData(
    [in] DWORD dwUserDataNumber,
    [out, size_is(cbBufSize)] BYTE * pData,
    [in] DWORD cbBufSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

dwUserDataNumber

Specified the user data number, zero-based.

pData

Place to store the user data, if NULL the only size of the specified user data will be returned.

cbBufSize

Buffer size.

pcbRetSize

Place to store the user data size.

Returns

Returns S_OK if successful or E_INVALIDARG in case of incorrect *dwUserDataNumber*.

Properties

AspectRatioCode

The Aspect Ratio code.

Syntax

```
__property CC_MPG_ASPECT_RATIO_CODE * AspectRatioCode;
```

BitRate

The bitrate of video bitstream.

Syntax

```
__property CC_BITRATE * BitRate;
```

ChromaFormat

Chroma format.

Syntax

```
__property CC_CHROMA_FORMAT * ChromaFormat;
```

ColorCoefs

The color transformarion description.

Syntax

```
__property CC_COLOUR_DESCRIPTION* ColorCoefs;
```

DisplaySize

The Display Size (resulting frame size).

Syntax

```
__property CC_SIZE * DisplaySize;
```

FrameDuration

The duration of one frame, in CC_TIMEBASE units.

Syntax

```
__property CC_TIME * FrameDuration;
```

Layer

The layer of MPEG video (1 or 2).

Syntax

```
__property DWORD * Layer;
```

Mpeg1ConstrainedFlag

MPEG-1 constrained stream flag.

Syntax

```
__property CC_BOOL * Mpeg1ConstrainedFlag;
```

ProfileAndLevel

Profile and level.

Syntax

```
__property CC_MPG_PROFILE_LEVEL* ProfileAndLevel;
```

UserDataCount

The number of MPEG_USER_DATA, associated with the video frame.

Syntax

```
__property DWORD * UserDataCount;
```

VBV_BufferSize

The VBV buffer size.

Syntax

```
__property DWORD * VBV_BufferSize;
```

VideoFormat

The video format.

Syntax

```
__property CC_VIDEO_FORMAT * VideoFormat;
```

MPEG Video Decoder

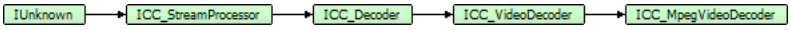
Interfaces

	Name	Description
	ICC_MpegVideoDecoder (see page 192)	The default and main interface to control the instance of CinecoderMpegVideoDecoder class.

ICC_MpegVideoDecoder Interface

The default and main interface to control the instance of CinecoderMpegVideoDecoder class.

Class Hierarchy



Syntax

```
[object, uuid(00002700-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_MpegVideoDecoder : ICC_VideoDecoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_VideoDecoder Interface

	Name	Description
	GetFrame (see page 91)	Retrieve the current video frame.
	GetStride (see page 91)	
	GetVideoFrameInfo (see page 91)	Get the description of current video frame.
	GetVideoStreamInfo (see page 92)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 92)	

ICC_MpegVideoDecoder Interface

	Name	Description
	GetMpegVideoFrameInfo (see page 194)	Get the MPEG-specific description of current video frame.
	GetMpegVideoStreamInfo (see page 194)	Get the MPEG-specific description of video stream which is being decoded.
	SetElementaryDataCallback (see page 195)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_MpegVideoDecoder Interface

	Name	Description
	PictureDecodingLevel (see page 195)	Controls the decoding of specific frame types. CC_I_FRAME allows decoding of I-frames only (P- and B-frames skips). CC_P_FRAME allows decoding of I- or P-frames (B-frames skips). CC_B_FRAME allows decoding of all three frame types (I, P, B).
	ThreadsAffinity (see page 195)	The affinity mask for object's threads. 0 = default (current process) affinity mask.
	ThreadsCount (see page 195)	Maximal number of threads which object can use. 0 = automatic.
	ThreadsPriority (see page 195)	The priority for object's threads.

Methods

GetMpegVideoFrameInfo

Get the MPEG-specific description of current video frame.

Syntax

```
HRESULT GetMpegVideoFrameInfo(
    DWORD field_no,
    [out,retval] ICC_MpegVideoFrameInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

Remarks

The mpeg frame can be coded either as one frame or as two fields. In the last case the second field stored in the elementary stream independently, directly after the first field and has different vbv delay, picture structure and also it can has different coding type. Reference See the ISO/IEC 13818-2 Intro. 4.1.2 Coding interlaced video.

GetMpegVideoStreamInfo

Get the MPEG-specific description of video stream which is being decoded.

Syntax

```
HRESULT GetMpegVideoStreamInfo(
    [out,retval] ICC_MpegVideoStreamInfo ** pDescr
```

```
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

SetElementaryDataCallback

Syntax

```
HRESULT SetElementaryDataCallback(
    [in] IUnknown * pCallback
);
```

Parameters

pCallback

Expected object with ICC_ByteStreamConsumer interface.

Properties

PictureDecodingLevel

Controls the decoding of specific frame types.

CC_I_FRAME allows decoding of I-frames only (P- and B-frames skips).

CC_P_FRAME allows decoding of I- or P-frames (B-frames skips).

CC_B_FRAME allows decoding of all three frame types (I, P, B).

Syntax

```
__property CC_FRAME_TYPE PictureDecodingLevel;
```

ThreadsAffinity

The affinity mask for object's threads. 0 = default (current process) affinity mask.

Syntax

```
__property CC_AFFINITY ThreadsAffinity;
```

ThreadsCount

Maximal number of threads which object can use. 0 = automatic.

Syntax

```
__property CC_AMOUNT ThreadsCount;
```

ThreadsPriority

The pripority for object's threads.

Syntax

```
__property CC_PRIORITY ThreadsPriority;
```

MPEG Video Encoder

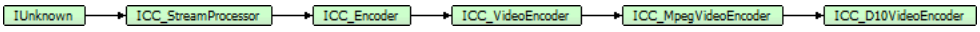
Interfaces

	Name	Description
	ICC_MpegVideoEncoder (see page 197)	The default and main interface to control the instance of CinecoderMpegVideoEncoder class.
	ICC_MpegVideoEncoderSettings (see page 201)	The settings for CinecoderMpegVideoEncoder initialization.

ICC_MpegVideoEncoder Interface

The default and main interface to control the instance of CinecoderMpegVideoEncoder class.

Class Hierarchy



Syntax

```
[object, uuid(00002400-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_MpegVideoEncoder : ICC_VideoEncoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_VideoEncoder Interface

	Name	Description
	AddFrame (see page 94)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 94)	Add a frame with different size to the processing queue.
	GetStride (see page 95)	

	GetVideoFrameInfo (see page 95)	Get the description of current video frame.
	GetVideoStreamInfo (see page 95)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 95)	
	IsScaleAvailable (see page 96)	

ICC_MpegVideoEncoder Interface

	Name	Description
	AddUserData (see page 199)	Adds user data for the subsequent video frame.
	GetMpegVideoFrameInfo (see page 199)	Get the MPEG-specific description of current video frame.

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

ICC_MpegVideoEncoder Interface

	Name	Description
	InitialTimeCode (see page 200)	Specifies the initial timecode for the first frame of the first GOP. Should be used before first call to AddFrame (see page 94).
	ThreadsAffinity (see page 200)	The affinity mask for object's threads. 0 = default (current process) affinity mask.
	ThreadsCount (see page 200)	Maximal number of threads which object can use. 0 = automatic.
	ThreadsPriority (see page 200)	The priority for object's threads.

Methods

AddUserData

Adds user data for the subsequent video frame.

Syntax

```
HRESULT AddUserData(
    [in, size_is(cbSize)] const BYTE * pbUserData,
    [in] DWORD cbSize,
    [in,defaultvalue(CC_FALSE)] CC_BOOL bSecondField
);
```

Parameters

pbUserData

The user data.

cbSize

The size of the user data, in bytes.

bSecondField

Tells that incoming user data must appear at the second picture_start_code (in case of interlaced coding).

Returns

Returns S_OK if successful or an error code otherwise.

Notes

You may call AddUserData several times to add more than one user data.

GetMpegVideoFrameInfo

Get the MPEG-specific description of current video frame.

Syntax

```
HRESULT GetMpegVideoFrameInfo(
    DWORD field_no,
    [out,retval] ICC_MpegVideoFrameInfo ** pDescr
);
```

Returns

Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

Remarks

The mpeg frame can be coded either as one frame or as two fields. In the last case the second field stored in the elementary stream independently, directly after the first field and has different vbv delay, picture structure and also it can has different coding type. Reference See the ISO/IEC 13818-2 Intro. 4.1.2 Coding interlaced video.

Properties

InitialTimeCode

Specifies the initial timecode for the first frame of the first GOP. Should be used before first call to AddFrame (see page 94).

Syntax

```
__property CC_TIMECODE InitialTimeCode;
```

ThreadsAffinity

The affinity mask for object's threads. 0 = default (current process) affinity mask.

Syntax

```
__property CC_AFFINITY ThreadsAffinity;
```

ThreadsCount

Maximal number of threads which object can use. 0 = automatic.

Syntax

```
__property CC_AMOUNT ThreadsCount;
```

ThreadsPriority

The priority for object's threads.

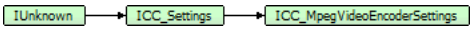
Syntax

```
__property CC_PRIORITY ThreadsPriority;
```

ICC_MpegVideoEncoderSettings Interface

The settings for CinecoderMpegVideoEncoder initialization.

Class Hierarchy



Syntax

```
[object, uuid(00002401-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_MpegVideoEncoderSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_MpegVideoEncoderSettings Interface

	Name	Description
	AddUserData (see page 204)	Adds another user data to the Sequence Layer (see page 208) of the MPEG video stream.
	GetQuantMatrix (see page 204)	
	GetUserData (see page 204)	Retrieves the specified user data which associated with the stream (seq_hdr level).
	SetQuantMatrix (see page 205)	

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_MpegVideoEncoderSettings Interface

	Name	Description
	AspectRatioCode (see page 205)	See CC_MPG_ASPECT_RATIO_CODE (see page 180).

	AvgBitRate (see page 205)	The average bitrate in VBR mode.
	BitRate (see page 205)	The target bitrate in CBR mode or max bitrate in VBR mode.
	BlocksPerSlice (see page 205)	The max number of macroblocks per slice.
	BlurFilterCoef (see page 206)	Blur filter coefficient. Simple 3x3 matrix of ones, with specified central coef (lower value means stronger blurring). 0 = disable blur. negative values = auto blur
	ChromaFormat (see page 206)	The chroma resolution format (4:2:0, 4:2:2 or 4:4:4).
	ClosedGOPs (see page 206)	Make all of GOPs "closed" - without backward referencing at the beginning of GOP (see page 207).
	ColorCoefs (see page 206)	
	DisableSceneDetector (see page 206)	Enable/disable the built-in scene change detector (which is ON by default).
	DisplaySize (see page 206)	The DisplaySize field of seq_disp_extension.
	EncodingLatency (see page 206)	Controls the latency of encoder (in frames) (0 = auto, 1 = min).
	FixedGopStructure (see page 207)	Fix the structure of a GOP (see page 207), even for the first GOP (see page 207) in stream.
	FrameRate (see page 207)	The frame's rate of video stream.
	FrameSize (see page 207)	The physical frame size, in pixels.
	GOP (see page 207)	The GOP settings. See CC_GOP_DESCR (see page 110) for details.
	InitialTimeCode (see page 207)	Specifies the initial timecode for the first frame of the first GOP (see page 207).
	InterlaceType (see page 207)	The video field order.
	IntraDCPrecision (see page 207)	The precision of intra DC coefficient (8-11 bits).
	IntraRefresh (see page 207)	If true,
	IntraVLCTable (see page 208)	A VLC table for encoding intra DC coefficients.

	Layer (see page 208)	The layer of the target MPEG coded video stream. Default is 2 (MPEG-2).
	LowDelay (see page 208)	Low delay mode
	MB_ScanPattern (see page 208)	The scan pattern (zig-zag or alternate).
	MB_Struct (see page 208)	Field/frame macrobloc structure.
	MEQuality (see page 208)	The motion estimation quality. Value in range 0..100.
	MinBitRate (see page 208)	The minimal bitrate in VBR mode (default is 0).
	MotionParams (see page 209)	See CC_MPG_MOTION_PARAMS (see page 180) for details.
	PictureStructure (see page 209)	The picture structure.
	ProfileAndLevel (see page 209)	The profile@level of the destination stream. Restricted by the license.
	ProgressiveSequence (see page 209)	If true, all frames in the stream coded without fields (progressive).
	QuantFunc (see page 209)	Quantization function
	QuantMatrixPictureLevel (see page 209)	Allows writing the quant matrix extension header at every specified picture type. CC_FRAME_TYPE_UNKNOWN disables it.
	QuantScale (see page 209)	The average quantization scale in VBR mode.
	QuantScaleType (see page 210)	Linear or non-linear VLC coefficients scale type.
	RateMode (see page 210)	The mode of bitrate controller - see CC_BITRATE_MODE.
	SequenceHeaderPeriod (see page 210)	The Frequency of Sequence Header generation, in frames. See CC_PERIOD_FLAGS for details. CC_ONCE(0) = only at the beginnig, FRQ_FOREVER(1) = at each GOP (see page 207). You can also specify the period in milliseconds by adding the FRQ_TIMEVAL_MS flag to the value.
	SuppressSeqEndCode (see page 210)	Put sequence_end_code after the last coded picture.
R	UserDataCount (see page 210)	The number of MPEG_USER_DATA, associated with the stream.

	VBV_BufferSize (see page 210)	The VBV buffer size. Please refer to the ISO/IEC 13818-2 Annex C. "Video Buffer Verifier". You can also specify it in milliseconds by adding FRQ_TIMEVAL_MS.
	VideoFormat (see page 210)	The video format.

Methods

AddUserData

Adds another user data to the Sequence Layer (see page 208) of the MPEG video stream.

Syntax

```
HRESULT AddUserData(
    [in,size_is(cbSize)] const BYTE * pbUserData,
    [in] DWORD cbSize
);
```

Parameters

pbUserData

The user data.

cbSize

The user data size.

Returns

Returns S_OK if successful or an error value otherwise.

GetQuantMatrix

Syntax

```
HRESULT GetQuantMatrix(
    [in] CC_MPG_QUANT_MATRIX t,
    [out,size_is(64)] BYTE * m
);
```

GetUserData

Retrieves the specified user data which associated with the stream (seq_hdr level).

Syntax

```
HRESULT GetUserData(
    [in] DWORD dwUserDataNumber,
    [out, size_is(cbBufSize)] BYTE * pData,
    [in] DWORD cbBufSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

dwUserDataNumber

Specified the user data number, zero-based.

pData

Place to store the user data, if NULL the only size of the specified user data will be returned.

cbBufSize

Buffer size.

pcbRetSize

Place to store the user data size.

Returns

Returns S_OK if successful or E_INVALIDARG in case of incorrect *dwUserDataNumber*.

SetQuantMatrix

Syntax

```
HRESULT SetQuantMatrix(
    [in] CC_MPG_QUANT_MATRIX t,
    [in ,size_is(64)] const BYTE * m
);
```

Properties

AspectRatioCode

See CC_MPG_ASPECT_RATIO_CODE (see page 180).

Syntax

```
__property CC_MPG_ASPECT_RATIO_CODE AspectRatioCode;
```

AvgBitRate

The average bitrate in VBR mode.

Syntax

```
__property CC_BITRATE AvgBitRate;
```

BitRate

The target bitrate in CBR mode or max bitrate in VBR mode.

Syntax

```
__property CC_BITRATE BitRate;
```

BlocksPerSlice

The max number of macroblocks per slice.

Syntax

```
__property CC_UINT BlocksPerSlice;
```

BlurFilterCoef

Blur filter coefficient. Simple 3x3 matrix of ones, with specified central coef (lower value means stronger blurring). 0 = disable blur. negative values = auto blur

Syntax

```
__property CC_INT BlurFilterCoef;
```

ChromaFormat

The chroma resolution format (4:2:0, 4:2:2 or 4:4:4).

Syntax

```
__property CC_CHROMA_FORMAT ChromaFormat;
```

ClosedGOPs

Make all of GOPs "closed" - without backward referencing at the beginning of GOP (see page 207).

Syntax

```
__property CC_BOOL ClosedGOPs;
```

ColorCoefs

Syntax

```
__property CC_COLOUR_DESCRIPTION ColorCoefs;
```

DisableSceneDetector

Enable/disable the built-in scene change detector (which is ON by default).

Syntax

```
__property CC_BOOL DisableSceneDetector;
```

DisplaySize

The DisplaySize field of seq_disp_extension.

Syntax

```
__property CC_SIZE DisplaySize;
```

EncodingLatency

Controls the latency of encoder (in frames) (0 = auto, 1 = min).

Syntax

```
__property CC_UINT EncodingLatency;
```

FixedGopStructure

Fix the structure of a GOP (see page 207), even for the first GOP (see page 207) in stream.

Syntax

```
__property CC_BOOL FixedGopStructure;
```

FrameRate

The frame's rate of video stream.

Syntax

```
__property CC_FRAME_RATE FrameRate;
```

FrameSize

The physical frame size, in pixels.

Syntax

```
__property CC_SIZE FrameSize;
```

GOP

The GOP settings. See CC_GOP_DESCR (see page 110) for details.

Syntax

```
__property CC_GOP_DESCR GOP;
```

InitialTimeCode

Specifies the initial timecode for the first frame of the first GOP (see page 207).

Syntax

```
__property CC_TIMECODE InitialTimeCode;
```

InterlaceType

The video field order.

Syntax

```
__property CC_INTERLACE_TYPE InterlaceType;
```

IntraDCPrecision

The precision of intra DC coefficient (8-11 bits).

Syntax

```
__property CC_UINT IntraDCPrecision;
```

IntraRefresh

If true,

Syntax

```
__property CC_BOOL IntraRefresh;
```

IntraVLCTable

A VLC table for encoding intra DC coefficients.

Syntax

```
__property CC_MPG_INTRA_VLC_TABLE IntraVLCTable;
```

Layer

The layer of the target MPEG coded video stream. Default is 2 (MPEG-2).

Syntax

```
__property CC_UINT Layer;
```

LowDelay

Low delay mode

Syntax

```
__property CC_BOOL LowDelay;
```

MB_ScanPattern

The scan pattern (zig-zag or alternate).

Syntax

```
__property CC_MPG_MB_SCAN_PATTERN MB_ScanPattern;
```

MB_Struct

Field/frame macrobloc structure.

Syntax

```
__property CC_MB_STRUCTURE MB_Struct;
```

MEQuality

The motion estimation quality. Value in range 0..100.

Syntax

```
__property CC_UINT MEQuality;
```

MinBitRate

The minimal bitrate in VBR mode (default is 0).

Syntax

```
__property CC_BITRATE MinBitRate;
```

MotionParams

See CC_MPG_MOTION_PARAMS (see page 180) for details.

Syntax

```
__property CC_UINT MotionParams;
```

PictureStructure

The picture structure.

Syntax

```
__property CC_PICTURE_STRUCTURE PictureStructure;
```

ProfileAndLevel

The profile@level of the destination stream. Restricted by the license.

Syntax

```
__property CC_MPG_PROFILE_LEVEL ProfileAndLevel;
```

ProgressiveSequence

If true, all frames in the stream coded without fields (progressive).

Syntax

```
__property CC_BOOL ProgressiveSequence;
```

QuantFunc

Quantization function

Syntax

```
__property CC_MPG_QUANT_FUNC QuantFunc;
```

QuantMatrixPictureLevel

Allows writing the quant matrix extension header at every specified picture type. CC_FRAME_TYPE_UNKNOWN disables it.

Syntax

```
__property CC_FRAME_TYPE QuantMatrixPictureLevel;
```

QuantScale

The average quantization scale in VBR mode.

Syntax

```
__property CC_FLOAT QuantScale;
```

QuantScaleType

Linear or non-linear VLC coefficients scale type.

Syntax

```
__property CC_MPG_QUANT_SCALE_TYPE QuantScaleType;
```

RateMode

The mode of bitrate controller - see CC_BITRATE_MODE.

Syntax

```
__property CC_BITRATE_MODE RateMode;
```

SequenceHeaderPeriod

The Frequency of Sequence Header generation, in frames. See CC_PERIOD_FLAGS for details. CC_ONCE(0) = only at the beginnig, FRQ_FOREVER(1) = at each GOP (see page 207). You can also specify the period in milliseconds by adding the FRQ_TIMEVAL_MS flag to the value.

Syntax

```
__property CC_PERIOD SequenceHeaderPeriod;
```

SuppressSeqEndCode

Put sequence_end_code after the last coded picture.

Syntax

```
__property CC_BOOL SuppressSeqEndCode;
```

UserDataCount

The number of MPEG_USER_DATA, associated with the stream.

Syntax

```
__property CC_UINT* UserDataCount;
```

VBV_BufferSize

The VBV buffer size. Please refer to the ISO/IEC 13818-2 Annex C. "Video Buffer Verifier". You can also specify it in milliseconds by adding FRQ_TIMEVAL_MS.

Syntax

```
__property CC_PERIOD VBV_BufferSize;
```

VideoFormat

The video format.

Syntax

```
__property CC_VIDEO_FORMAT VideoFormat;
```

D10/IMX Video Encoder

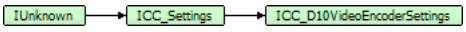
Interfaces

	Name	Description
	ICC_D10VideoEncoderSettings (see page 212)	The settings for CinecoderD10VideoEncoder initialization.
	ICC_D10VideoEncoder (see page 215)	Interface for D10 video encoder is the same as for MpegVideoEncoder.

ICC_D10VideoEncoderSettings Interface

The settings for CinecoderD10VideoEncoder initialization.

Class Hierarchy



Syntax

```
[object, uuid(d6baaecc-900a-4fce-bb7a-5feb665be275), pointer_default(unique), local]
interface ICC_D10VideoEncoderSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_D10VideoEncoderSettings Interface

	Name	Description
	AddUserData (see page 213)	Adds another user data to the Sequence Layer of a MPEG video stream.
	GetUserData (see page 214)	Retrieves the specified user data associated with the stream (seq_hdr level).

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_D10VideoEncoderSettings Interface

	Name	Description
	AspectRatioCode (see page 214)	Optional. See CC_ASPECT_RATIO.
	BitRate (see page 214)	Required. The bitrate of target IMX stream. Possible values are 30000000, 40000000 and 50000000 bits/sec.
	ColorCoefs (see page 215)	The color transformarion description.
	FrameSize (see page 215)	The frame size. You can specify 720x576, 720x608 for PAL and 720x480, 720x512 for NTSC. In the first cases 32 blank lines will be added automatically.
	QuantFunc (see page 215)	Quantization function
R	UserDataCount (see page 215)	The number of MPEG_USER_DATA, associated with the stream.
	VideoFormat (see page 215)	The IMX video format (CC_VIDEO_FORMAT_PAL or CC_VIDEO_FORMAT_NTSC). If no FrameSize (see page 215) is specified, 720x608 or 720x512 are assumed.

Methods

AddUserData

Adds another user data to the Sequence Layer of a MPEG video stream.

Syntax

```
HRESULT AddUserData(
    [in,size_is(cbSize)] const BYTE * pbUserData,
    [in] DWORD cbSize
);
```

Parameters

pbUserData

The user data.

cbSize

The user data size.

Returns

Returns S_OK if successful or an error value otherwise.

GetUserData

Retrieves the specified user data associated with the stream (seq_hdr level).

Syntax

```
HRESULT GetUserData(
    [in] DWORD dwUserDataNumber,
    [out, size_is(cbBufSize)] BYTE * pData,
    [in] DWORD cbBufSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

dwUserDataNumber

Specified the user data number, zero-based.

pData

Place to store the user data, if NULL the only size of the specified user data will be returned.

cbBufSize

Buffer size.

pcbRetSize

Place to store the user data size.

Returns

Returns S_OK if successful or E_INVALIDARG in case of incorrect dwUserDataNumber.

Properties

AspectRatioCode

Optional. See CC_ASPECT_RATIO.

Syntax

```
__property CC_MPG_ASPECT_RATIO_CODE AspectRatioCode;
```

BitRate

Required. The bitrate of target IMX stream. Possible values are 30000000, 40000000 and 50000000 bits/sec.

Syntax

```
__property CC_BITRATE BitRate;
```

ColorCoefs

The color transformarion description.

Syntax

```
__property CC_COLOUR_DESCRIPTION ColorCoefs;
```

FrameSize

The frame size. You can specify 720x576, 720x608 for PAL and 720x480, 720x512 for NTSC. In the first cases 32 blank lines will be added automatically.

Syntax

```
__property CC_SIZE FrameSize;
```

QuantFunc

Quantization function

Syntax

```
__property CC_MPG_QUANT_FUNC QuantFunc;
```

UserDataCount

The number of MPEG_USER_DATA, associated with the stream.

Syntax

```
__property DWORD* UserDataCount;
```

VideoFormat

The IMX video format (CC_VIDEO_FORMAT_PAL or CC_VIDEO_FORMAT_NTSC). If no FrameSize (see page 215) is specified, 720x608 or 720x512 are assumed.

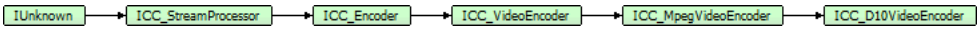
Syntax

```
__property CC_VIDEO_FORMAT VideoFormat;
```

ICC_D10VideoEncoder Interface

Interface for D10 video encoder is the same as for MpegVideoEncoder.

Class Hierarchy



Syntax

```
[object, uuid(aa7effd7-7830-4400-b51e-ac7b3510f9c1), pointer_default(unique), local]
interface ICC_D10VideoEncoder : ICC_MpegVideoEncoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_VideoEncoder Interface

	Name	Description
	AddFrame (see page 94)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 94)	Add a frame with different size to the processing queue.
	GetStride (see page 95)	
	GetVideoFrameInfo (see page 95)	Get the description of current video frame.
	GetVideoStreamInfo (see page 95)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 95)	
	IsScaleAvailable (see page 96)	

ICC_MpegVideoEncoder Interface

	Name	Description
	AddUserData (see page 199)	Adds user data for the subsequent video frame.
	GetMpegVideoFrameInfo (see page 199)	Get the MPEG-specific description of current video frame.

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream

R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

ICC_MpegVideoEncoder Interface

	Name	Description
W	InitialTimeCode (see page 200)	Specifies the initial timecode for the first frame of the first GOP. Should be used before first call to AddFrame (see page 94).
	ThreadsAffinity (see page 200)	The affinity mask for object's threads. 0 = default (current process) affinity mask.
	ThreadsCount (see page 200)	Maximal number of threads which object can use. 0 = automatic.
	ThreadsPriority (see page 200)	The priority for object's threads.

MPEG Audio

Interfaces

	Name	Description
	ICC_MpegAudioStreamInfo (see page 227)	Represents the MPEG-specified video stream description.

Enumerations

	Name	Description
	CC_MPG_AUDIO_CHANNEL_MODE (see page 230)	
	CC_MPG_AUDIO_EMPHASIS (see page 231)	The MPEG Audio emphasis info. Indicates the type of de-emphasis that shall be used (see iso/iec 11172-3 2.4.2.3 for details).

MPEG Audio Encoder

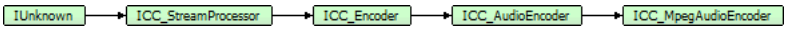
Interfaces

	Name	Description
	ICC_MpegAudioEncoder (see page 219)	The main interface to access the CC_MpegAudioEncoder class.
	ICC_MpegAudioEncoderSettings (see page 221)	The settings for CinecoderMpegAudioEncoder initialization.

ICC_MpegAudioEncoder Interface

The main interface to access the CC_MpegAudioEncoder class.

Class Hierarchy



Syntax

```
[object, uuid(a8810f12-baf4-449f-a5d7-052e24adb356), pointer_default(unique), local]
interface ICC_MpegAudioEncoder : ICC_AudioEncoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_AudioEncoder Interface

	Name	Description
	GetAudioFrameInfo (see page 129)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 129)	Get the description of audio stream which is being decoded.

	ProcessAudio (see page 130)	Puts another audio uncompressed data to the audio consumer.

ICC_MpegAudioEncoder Interface

	Name	Description
	GetMpegAudioFrameInfo (see page 220)	
	GetMpegAudioStreamInfo (see page 221)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

Methods

GetMpegAudioFrameInfo

Syntax

```
HRESULT GetMpegAudioFrameInfo(
    [out,retval] ICC_MpegAudioFrameInfo ** pDescr
);
```

GetMpegAudioStreamInfo

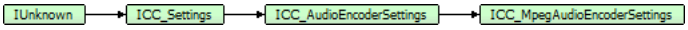
Syntax

```
HRESULT GetMpegAudioStreamInfo(
    [out,retval] ICC_MpegAudioStreamInfo ** pDescr
);
```

ICC_MpegAudioEncoderSettings Interface

The settings for CinecoderMpegAudioEncoder initialization.

Class Hierarchy



Syntax

```
[object, uuid(40858c51-86e4-496c-a6e8-81cbfe2dcc1c), pointer_default(unique), local]
interface ICC_MpegAudioEncoderSettings : ICC_AudioEncoderSettings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_AudioEncoderSettings Interface

	Name	Description
	BitRate (see page 135)	The bitrate of entire audio bitstream or corresponding audio frame.
	BitsPerSample (see page 135)	The bit depth of one audio sample (of one channel).
	FrameRate (see page 135)	The number of audio frames per second.

	NumChannels (see page 135)	The number of channels in the waveform-audio data.
	SampleRate (see page 135)	The audio sampling frequency.

ICC_MpegAudioEncoderSettings Interface

	Name	Description
	BitRate (see page 222)	Destination bitrate.
	ChannelMode (see page 222)	Mpeg audio channel mode (see CC_MPG_AUDIO_CHANNEL_MODE (see page 230)).
	CopyrightedFlag (see page 222)	The Copyright flag.
	Emphasis (see page 223)	Emphasis (see CC_MPG_AUDIO_EMPHASIS (see page 231)).
	ErrorProtection (see page 223)	The coded audio stream is error-protected.
	Layer (see page 223)	The layer of mpeg audio, default is 2.
	OriginalFlag (see page 223)	The Original flag.
	SampleRate (see page 223)	The audio sample rate.

Properties

BitRate

Destination bitrate.

Syntax

```
__property CC_BITRATE BitRate;
```

ChannelMode

Mpeg audio channel mode (see CC_MPG_AUDIO_CHANNEL_MODE (see page 230)).

Syntax

```
__property CC_MPG_AUDIO_CHANNEL_MODE ChannelMode;
```

CopyrightedFlag

The Copyright flag.

Syntax

```
__property CC_BOOL CopyrightedFlag;
```

Emphasis

Emphasis (see CC_MPG_AUDIO_EMPHASIS (see page 231)).

Syntax

```
__property CC_MPG_AUDIO_EMPHASIS Emphasis;
```

ErrorProtection

The coded audio stream is error-protected.

Syntax

```
__property CC_BOOL ErrorProtection;
```

Layer

The layer of mpeg audio, default is 2.

Syntax

```
__property CC_UINT Layer;
```

OriginalFlag

The Original flag.

Syntax

```
__property CC_BOOL OriginalFlag;
```

SampleRate

The audio sample rate.

Syntax

```
__property CC_UINT SampleRate;
```

MPEG Audio Decoder

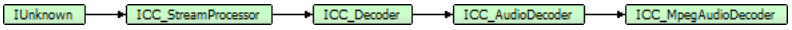
Interfaces

	Name	Description
	ICC_MpegAudioDecoder (see page 224)	The main interface to access the CC_MpegAudioDecoder class.

ICC_MpegAudioDecoder Interface

The main interface to access the CC_MpegAudioDecoder class.

Class Hierarchy



Syntax

```
[object, uuid(7b9c9deb-d2c4-4239-bd19-37214ae27b58), pointer_default(unique), local]
interface ICC_MpegAudioDecoder : ICC_AudioDecoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_AudioDecoder Interface

	Name	Description
	GetAudio (see page 125)	Retrieves the uncompressed audio data from the audio producer. Remark: To obtain the size of the buffer to hold the resulted samples, put NULL instead of pbData.
	GetAudioFrameInfo (see page 126)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 126)	Get the description of the audio stream.
	GetSampleBytes (see page 126)	
	IsFormatSupported (see page 127)	

ICC_MpegAudioDecoder Interface

	Name	Description
	GetMpegAudioFrameInfo (see page 226)	
	GetMpegAudioStreamInfo (see page 226)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_AudioDecoder Interface

	Name	Description
	NumSamples (see page 127)	The number of ready audio samples at the output.

Methods

GetMpegAudioFrameInfo

Syntax

```
HRESULT GetMpegAudioFrameInfo(
    [out,retval] ICC_MpegAudioFrameInfo ** pDescr
);
```

GetMpegAudioStreamInfo

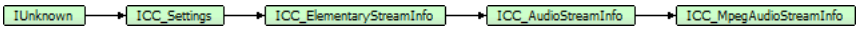
Syntax

```
HRESULT GetMpegAudioStreamInfo(
    [out,retval] ICC_MpegAudioStreamInfo ** pDescr
);
```

ICC_MpegAudioStreamInfo Interface

Represents the MPEG-specified video stream description.

Class Hierarchy



Syntax

```
[object, uuid(16bd8d1b-5245-4f88-a43f-2363ef8f4b2b), pointer_default(unique), local]
interface ICC_MpegAudioStreamInfo : ICC_AudioStreamInfo;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ElementaryStreamInfo Interface

	Name	Description
R	BitRate (see page 52)	The bitrate (max) of the elementary stream.
R	FrameRate (see page 52)	The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.
R	StreamType (see page 52)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

ICC_AudioStreamInfo Interface

	Name	Description
	BitsPerSample (see page 133)	The bit depth of one audio sample (of one channel).
	ChannelMask (see page 133)	The mask of channels. See the CC_AUDIO_CHANNEL_MASK for details
	NumChannels (see page 133)	The number of channels in the waveform-audio data.
	SampleRate (see page 133)	The audio sampling frequency.

ICC_MpegAudioStreamInfo Interface

	Name	Description
	ChannelMode (see page 228)	The audio channel mode.
	CopyrightedFlag (see page 228)	The Copyright flag.
	Emphasis (see page 229)	The emphasis, specified in ISO/IEC 11172-3 standard.
	ErrorProtection (see page 229)	The coded audio stream is error-protected.
	Layer (see page 229)	The layer of MPEG audio (1-3).
	OriginalFlag (see page 229)	The Original flag.

Properties

ChannelMode

The audio channel mode.

Syntax

```
__property CC_MPG_AUDIO_CHANNEL_MODE* ChannelMode;
```

CopyrightedFlag

The Copyright flag.

Syntax

```
__property CC_BOOL * CopyrightedFlag;
```

Emphasis

The emphasis, specified in ISO/IEC 11172-3 standard.

Syntax

```
__property CC_MPG_AUDIO_EMPHASIS* Emphasis;
```

ErrorProtection

The coded audio stream is error-protected.

Syntax

```
__property CC_BOOL * ErrorProtection;
```

Layer

The layer of MPEG audio (1-3).

Syntax

```
__property CC_UINT * Layer;
```

OriginalFlag

The Original flag.

Syntax

```
__property CC_BOOL * OriginalFlag;
```

CC_MPG_AUDIO_CHANNEL_MODE

Syntax

```
[v1_enum]
enum CC_MPG_AUDIO_CHANNEL_MODE {
  CC_MPG_ACH_UNKNOWN = -1,
  CC_MPG_ACH_STEREO = 0,
  CC_MPG_ACH_JOINT_STEREO = 1,
  CC_MPG_ACH_DUAL_CHANNEL = 2,
  CC_MPG_ACH_MONO = 3
};
```

Members

CC_MPG_ACH_UNKNOWN

Unknown audio mode.

CC_MPG_ACH_STEREO

2 corellated audio channels (standard music).

CC_MPG_ACH_JOINT_STEREO

2 highly corellated audio channels, 1 master channel and 1 "difference" channel.

CC_MPG_ACH_DUAL_CHANNEL

2 independent audio channels.

CC_MPG_ACH_MONO

1 audio channel.

CC_MPG_AUDIO_EMPHASIS

The MPEG Audio emphasis info. Indicates the type of de-emphasis that shall be used (see iso/iec 11172-3 2.4.2.3 for details).

Syntax

```
[v1_enum]
enum CC_MPG_AUDIO_EMPHASIS {
  CC_MPG_EMPHASIS_UNKNOWN = 0,
  CC_MPG_EMPHASIS_0_15 = 1,
  CC_MPG_EMPHASIS_CCITT_J17 = 3
};
```

Members

CC_MPG_EMPHASIS_UNKNOWN

Unknown emphasis.

CC_MPG_EMPHASIS_0_15

50/15 microseconds.

CC_MPG_EMPHASIS_CCITT_J17

CCITT J.17.

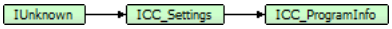
MPEG Multiplex

Interfaces

	Name	Description
	ICC_ProgramInfo (see page 233)	
	ICC_SystemDescriptorsManager (see page 235)	The extension to ICC_SystemDescriptorsReader (see page 244), which allows you to add and remove the descriptors.
	ICC_PES_Info (see page 238)	
	ICC_TS_ProgramDescr (see page 241)	
	ICC_SystemDescriptorsReader (see page 244)	The interface to read the descriptors associated with Program, Transport or any elementary stream.

ICC_ProgramInfo Interface

Class Hierarchy



Syntax

```
[object, uuid(00001802-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_ProgramInfo : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_ProgramInfo Interface

	Name	Description
	GetStream (see page 234)	

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ProgramInfo Interface

	Name	Description
R	Descriptors (see page 234)	
R	NumStreams (see page 234)	
R	PCR_PID (see page 234)	
R	PMT_PID (see page 234)	
R	ProgNum (see page 234)	

Methods

GetStream

Syntax

```
HRESULT GetStream(
    [in]DWORD StreamNumber,
    [out,retval]ICC_PES_Info**
);
```

Properties

Descriptors

Syntax

```
__property ICC_SystemDescriptorsReader** Descriptors;
```

NumStreams

Syntax

```
__property DWORD* NumStreams;
```

PCR_PID

Syntax

```
__property CC_PID* PCR_PID;
```

PMT_PID

Syntax

```
__property CC_PID* PMT_PID;
```

ProgNum

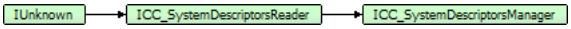
Syntax

```
__property WORD* ProgNum;
```

ICC_SystemDescriptorsManager Interface

The extension to ICC_SystemDescriptorsReader (see page 244), which allows you to add and remove the descriptors.

Class Hierarchy



Syntax

```
[object, uuid(00001fff-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_SystemDescriptorsManager : ICC_SystemDescriptorsReader;
```

Methods

	Name	Description
	Get (see page 244)	Retrieves the specified descriptor by its ordinal number.
	IndexOf (see page 245)	Retrieves the descriptor index with the specified code.
	StoreToBuffer (see page 245)	Get (see page 244) all stored descriptors as raw data.

ICC_SystemDescriptorsManager Interface

	Name	Description
	Add (see page 236)	Add the new descriptor to the object.
	AddFromBuffer (see page 236)	Add (see page 236) descriptors from raw buffer. Descriptors must be collocated, in form [code:byte][length:byte][data:byte[]]...
	Clear (see page 237)	Empties the list of descriptors.
	CopyFrom (see page 237)	Add (see page 236) all of descriptors from another object.
	Remove (see page 237)	The method removes the specified descriptor from a list.

Properties

	Name	Description
R	NumDescr (see page 246)	Retrieves the number of descriptors in the list.
R	Size (see page 246)	Retrieves the size of bytes of all of descriptors.

Methods

Add

Add the new descriptor to the object.

Syntax

```
HRESULT Add(
    [in] MPEG_SYSTEM_DESCRIPTOR_TAG DescrCode,
    [in,size_is(cbDescrSize)] BYTE * pbData,
    [in] DWORD cbDescrSize
);
```

Parameters

DescrCode

The code of the descriptor to be added.

pbData

The descriptor's body. (Can be NULL, see above).

cbDescrSize

The size of the descriptor to be added.

Returns

S_OK if successful, E_INVALIDARG if DescrCode invalid, E_OUTOFMEMORY if there is no space in the list.

AddFromBuffer

Add (see page 236) descriptors from raw buffer. Descriptors must be collocated, in form [code:byte][length:byte][data:byte[]]...

Syntax

```
HRESULT AddFromBuffer(
    [in,size_is(cbSize)] BYTE * pbData,
    [in] DWORD cbSize
);
```

Parameters

pbData

The raw descriptors buffer.

cbSize

The buffer size.

Returns

S_OK if successful, E_OUTOFMEMORY if no space in the list.

Clear

Empties the list of descriptors.

Syntax

```
HRESULT Clear();
```

Returns

S_OK if successful.

CopyFrom

Add (see page 236) all of descriptors from another object.

Syntax

```
HRESULT CopyFrom(
    [in] ICC_SystemDescriptorsReader * pSrc
);
```

Parameters

pSrc

The source.

Returns

S_OK if successful, E_OUTOFMEMORY if no space in the list.

Remove

The method removes the specified descriptor from a list.

Syntax

```
HRESULT Remove(
    [in] INT dwDescrNumber
);
```

Parameters

dwDescrNumber

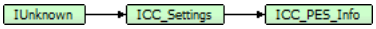
The descriptor index to be removed.

Returns

S_OK if successful, E_INVALIDARG if index is out of bounds.

ICC_PES_Info Interface

Class Hierarchy



Syntax

```
[object, uuid(00001801-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_PES_Info : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_PES_Info Interface

	Name	Description
R	AdditionalCopyInfo (see page 239)	
R	BasePTS (see page 239)	
R	CrcProtected (see page 239)	
R	Descriptors (see page 239)	
R	OriginalFlag (see page 239)	
R	PacketHeaderSize (see page 239)	
R	PacketSize (see page 240)	

R	PID (see page 240)	
R	PriorityFlag (see page 240)	
R	ScramblingControl (see page 240)	
R	StreamID (see page 240)	
R	StreamType (see page 240)	

Properties

AdditionalCopyInfo

Syntax

```
__property BYTE* AdditionalCopyInfo;
```

BasePTS

Syntax

```
__property CC_TIME* BasePTS;
```

CrcProtected

Syntax

```
__property CC_BOOL* CrcProtected;
```

Descriptors

Syntax

```
__property ICC_SystemDescriptorsReader** Descriptors;
```

OriginalFlag

Syntax

```
__property CC_BOOL* OriginalFlag;
```

PacketHeaderSize

Syntax

```
__property DWORD* PacketHeaderSize;
```

PacketSize

Syntax

```
__property INT* PacketSize;
```

PID

Syntax

```
__property CC_PID* PID;
```

PriorityFlag

Syntax

```
__property CC_BOOL* PriorityFlag;
```

ScramblingControl

Syntax

```
__property DWORD* ScramblingControl;
```

StreamID

Syntax

```
__property BYTE* StreamID;
```

StreamType

Syntax

```
__property CC_ELEMENTARY_STREAM_TYPE* StreamType;
```

ICC_TS_ProgramDescr Interface

Class Hierarchy

Syntax

```
[object, uuid(00001C11-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_TS_ProgramDescr : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_TS_ProgramDescr Interface

	Name	Description
	CrcProtected (see page 242)	
R	Descriptors (see page 242)	
	DiscontinuityThreshold (see page 242)	
	InitialPTS (see page 242)	
	PCR_Period (see page 242)	
	PCR_PID (see page 242)	
	PMT_Period (see page 242)	

	PMT_PID (see page 242)	
	ProgNum (see page 243)	

Properties

CrcProtected

Syntax

```
__property [in] CrcProtected;
```

Descriptors

Syntax

```
__property ICC_SystemDescriptorsManager** Descriptors;
```

DiscontinuityThreshold

Syntax

```
__property [in] DiscontinuityThreshold;
```

InitialPTS

Syntax

```
__property [in] InitialPTS;
```

PCR_Period

Syntax

```
__property [in] PCR_Period;
```

PCR_PID

Syntax

```
__property [in] PCR_PID;
```

PMT_Period

Syntax

```
__property [in] PMT_Period;
```

PMT_PID

Syntax

```
__property [in] PMT_PID;
```

ProgNum

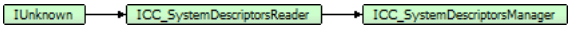
Syntax

```
__property [in] ProgNum;
```

ICC_SystemDescriptorsReader Interface

The interface to read the descriptors associated with Program, Transport or any elementary stream.

Class Hierarchy



Syntax

```
[object, uuid(00001ffe-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_SystemDescriptorsReader : IUnknown;
```

Methods

	Name	Description
	Get (see page 244)	Retrieves the specified descriptor by its ordinal number.
	IndexOf (see page 245)	Retrieves the descriptor index with the specified code.
	StoreToBuffer (see page 245)	Get (see page 244) all stored descriptors as raw data.

Properties

	Name	Description
R	NumDescr (see page 246)	Retrieves the number of descriptors in the list.
R	Size (see page 246)	Retrieves the size of bytes of all of descriptors.

Methods

Get

Retrieves the specified descriptor by its ordinal number.

Syntax

```
HRESULT Get(
    [in] INT dwDescrIndex,
    [out,retval] CC_SYSDESCR * pDescr
);
```

Parameters

dwDescrIndex

The index of descriptor.

pDescr

Place to store the descriptor.

Returns

S_OK if successful, E_INVALIDARG if index out of bounds.

IndexOf

Retrieves the descriptor index with the specified code.

Syntax

```
HRESULT IndexOf(
    [in] MPEG_SYSTEM_DESCRIPTOR_TAG DescrCode,
    [in,defaultvalue(0)] INT idxSearchFrom,
    [out,retval] INT * pDescrIndex
);
```

Parameters

DescrCode

The descriptor code to be found.

idxSearchFrom

The index from where the search begins.

pDescrIndex

Place to store the index. If no descriptors with specified code wiil be found, -1 will be returned.

Returns

S_OK if found and S_FALSE if not found.

StoreToBuffer

Get (see page 244) all stored descriptors as raw data.

Syntax

```
HRESULT StoreToBuffer(
    [out,size_is(cbBufSize)] BYTE * pbData,
    [in] DWORD cbBufSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

pbData

The raw descriptors buffer.

cbBufSize

The buffer size.

pcbRetSize

The descriptors' reaw size.

Returns

S_OK if successful, E_OUTOFMEMORY if no space in the list.

Properties

NumDescr

Retrieves the number of descriptors in the list.

Syntax

```
__property INT * NumDescr;
```

Parameters

pNumDescr

Place to store the quantity of descriptors.

Returns

S_OK.

Size

Retrieves the size of bytes of all of descriptors.

Syntax

```
__property DWORD * Size;
```

Parameters

pcbSize

Place to store the size of descriptors' list.

Returns

S_OK.

MPEG-2 Program Stream

Interfaces

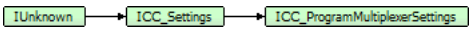
	Name	Description
	ICC_ProgramMultiplexerSettings (see page 247)	
	ICC_ProgramMuxerPinSettings (see page 250)	

Program Stream Multiplexer

Program Stream Demultiplexer

ICC_ProgramMultiplexerSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001B02-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_ProgramMultiplexerSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_ProgramMultiplexerSettings Interface

	Name	Description
	AddStream (see page 249)	
	GetStream (see page 249)	

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ProgramMultiplexerSettings Interface

	Name	Description
	BitRate (see page 249)	
	CrcProtected (see page 249)	
R	Descriptors (see page 249)	
R	NumStreams (see page 249)	
	PacketAlignment (see page 249)	
	PacketHeaderSize (see page 249)	
	PacketSize (see page 250)	
	PackHeaderPeriod (see page 250)	
	PayloadAlignment (see page 250)	
	PutProgramStreamMap (see page 250)	
	RateMode (see page 250)	
	StandaloneSystemHeader (see page 250)	
	SuppressIsoEndCode (see page 250)	
	SystemHeaderPeriod (see page 250)	
	TrailingAlignment (see page 250)	

Methods

AddStream

Syntax

```
HRESULT AddStream(
    [in] ICC_ProgramMuxerPinSettings*
);
```

GetStream

Syntax

```
HRESULT GetStream(
    [in] DWORD StreamNumber,
    [out,retval]ICC_ProgramMuxerPinSettings**
);
```

Properties

BitRate

Syntax

```
__property [in] BitRate;
```

CrcProtected

Syntax

```
__property [in] CrcProtected;
```

Descriptors

Syntax

```
__property ICC_SystemDescriptorsManager** Descriptors;
```

NumStreams

Syntax

```
__property DWORD* NumStreams;
```

PacketAlignment

Syntax

```
__property [in] PacketAlignment;
```

PacketHeaderSize

Syntax

```
__property [in] PacketHeaderSize;
```

PacketSize

Syntax

```
__property [in] PacketSize;
```

PackHeaderPeriod

Syntax

```
__property [in] PackHeaderPeriod;
```

PayloadAlignment

Syntax

```
__property [in] PayloadAlignment;
```

PutProgramStreamMap

Syntax

```
__property [in] PutProgramStreamMap;
```

RateMode

Syntax

```
__property [in] RateMode;
```

StandaloneSystemHeader

Syntax

```
__property [in] StandaloneSystemHeader;
```

SuppressIsoEndCode

Syntax

```
__property [in] SuppressIsoEndCode;
```

SystemHeaderPeriod

Syntax

```
__property [in] SystemHeaderPeriod;
```

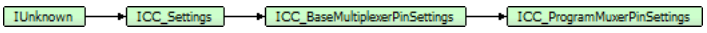
TrailingAlignment

Syntax

```
__property [in] TrailingAlignment;
```

ICC_ProgramMuxerPinSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001B12-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_ProgramMuxerPinSettings : ICC_BaseMultiplexerPinSettings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_BaseMultiplexerPinSettings Interface

	Name	Description
	BasePTS (see page 149)	
	BitRate (see page 149)	
	DataAlignPeriod (see page 149)	The period at which the access units will align to the packet's boundary. By default, audio streams aligned only at the beginning, video streams are aligned each I-frame.
	FrameRate (see page 149)	
	SampleRate (see page 149)	
	StreamID (see page 149)	
	StreamType (see page 149)	

ICC_ProgramMuxerPinSettings Interface

	Name	Description
	AdditionalCopyInfo (see page 252)	

	CrcProtected (see page 252)	
R	Descriptors (see page 252)	
	HideFromSystemHeader (see page 252)	
	OriginalFlag (see page 253)	
	PacketAlignment (see page 253)	
	PacketHeaderSize (see page 253)	
	PacketSize (see page 253)	
	PayloadAlignment (see page 253)	
	PriorityFlag (see page 253)	
	ScramblingControl (see page 253)	

Properties

AdditionalCopyInfo

Syntax

```
__property [in] AdditionalCopyInfo;
```

CrcProtected

Syntax

```
__property [in] CrcProtected;
```

Descriptors

Syntax

```
__property ICC_SystemDescriptorsReader** Descriptors;
```

HideFromSystemHeader

Syntax

```
__property [in] HideFromSystemHeader;
```

OriginalFlag

Syntax

```
__property [in] OriginalFlag;
```

PacketAlignment

Syntax

```
__property [in] PacketAlignment;
```

PacketHeaderSize

Syntax

```
__property [in] PacketHeaderSize;
```

PacketSize

Syntax

```
__property [in] PacketSize;
```

PayloadAlignment

Syntax

```
__property [in] PayloadAlignment;
```

PriorityFlag

Syntax

```
__property [in] PriorityFlag;
```

ScramblingControl

Syntax

```
__property [in] ScramblingControl;
```

MPEG-2 Transport Stream

Interfaces

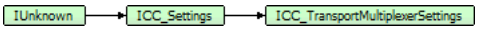
	Name	Description
	ICC_TransportMultiplexerSettings (see page 254)	
	ICC_TransportMuxerPinSettings (see page 257)	
	ICC_M2TSMP_MultiplexerSettings (see page 259)	

Transport Stream Multiplexer

Transport Stream Demultiplexer

ICC_TransportMultiplexerSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001C01-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_TransportMultiplexerSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_TransportMultiplexerSettings Interface

	Name	Description
	BitRate (see page 255)	
	CrcProtected (see page 256)	
R	Descriptors (see page 256)	
	PAT_Period (see page 256)	
	PCR_Period (see page 256)	
	PCR_PID (see page 256)	
	PMT_Period (see page 256)	
	PMT_PID (see page 256)	
	ProgNum (see page 256)	
	RateMode (see page 256)	
	StreamID (see page 256)	
	TrailingAlignment (see page 257)	

Properties

BitRate

Syntax

```
__property [in] BitRate;
```

CrcProtected

Syntax

```
__property [in] CrcProtected;
```

Descriptors

Syntax

```
__property ICC_SystemDescriptorsManager** Descriptors;
```

PAT_Period

Syntax

```
__property [in] PAT_Period;
```

PCR_Period

Syntax

```
__property [in] PCR_Period;
```

PCR_PID

Syntax

```
__property [in] PCR_PID;
```

PMT_Period

Syntax

```
__property [in] PMT_Period;
```

PMT_PID

Syntax

```
__property [in] PMT_PID;
```

ProgNum

Syntax

```
__property [in] ProgNum;
```

RateMode

Syntax

```
__property [in] RateMode;
```

StreamID

Syntax

```
__property [in] StreamID;
```

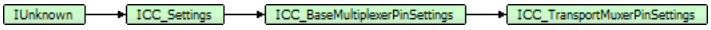
TrailingAlignment

Syntax

```
__property [in] TrailingAlignment;
```

ICC_TransportMuxerPinSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001C21-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_TransportMuxerPinSettings : ICC_BaseMultiplexerPinSettings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_BaseMultiplexerPinSettings Interface

	Name	Description
	BasePTS (see page 149)	
	BitRate (see page 149)	
	DataAlignPeriod (see page 149)	The period at which the access units will align to the packet's boundary. By default, audio streams aligned only at the beginning, video streams are aligned each I-frame.
	FrameRate (see page 149)	
	SampleRate (see page 149)	

	StreamID (see page 149)	
	StreamType (see page 149)	

ICC_TransportMuxerPinSettings Interface

	Name	Description
	AdditionalCopyInfo (see page 258)	
	CrcProtected (see page 258)	
R	Descriptors (see page 258)	
	OriginalFlag (see page 259)	
	PesHeaderPeriod (see page 259)	The period at which PES headers will be added to the corresponding access unit. By default at each audio or video access unit.
	PID (see page 259)	
	PriorityFlag (see page 259)	
	ScramblingControl (see page 259)	

Properties

AdditionalCopyInfo

Syntax

```
__property [in] AdditionalCopyInfo;
```

CrcProtected

Syntax

```
__property [in] CrcProtected;
```

Descriptors

Syntax

```
__property ICC_SystemDescriptorsManager** Descriptors;
```

OriginalFlag

Syntax

```
__property [in] OriginalFlag;
```

PesHeaderPeriod

The period at which PES headers will be added to the corresponding access unit. By default at each audio or video access unit.

Syntax

```
__property [in] PesHeaderPeriod;
```

PID

Syntax

```
__property [in] PID;
```

PriorityFlag

Syntax

```
__property [in] PriorityFlag;
```

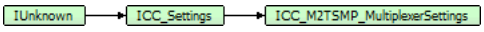
ScramblingControl

Syntax

```
__property [in] ScramblingControl;
```

ICC_M2TSMP_MultiplexerSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001C02-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_M2TSMP_MultiplexerSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_M2TSMP_MultiplexerSettings Interface

	Name	Description
	AddProgram (see page 261)	
	GetProgram (see page 261)	

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_M2TSMP_MultiplexerSettings Interface

	Name	Description
	BitRate (see page 261)	
	CrcProtected (see page 261)	
R	NumPrograms (see page 261)	
	PAT_Period (see page 261)	
	PCR_Period (see page 261)	
	PMT_Period (see page 261)	
	ProgNumBase (see page 261)	
	RateMode (see page 262)	
	StreamID (see page 262)	
	TrailingAlignment (see page 262)	

Methods

AddProgram

Syntax

```
HRESULT AddProgram(
    [in] ICC_TS_ProgramDescr*
);
```

GetProgram

Syntax

```
HRESULT GetProgram(
    [in] DWORD ProgramIdx,
    [out,retval]ICC_TS_ProgramDescr**
);
```

Properties

BitRate

Syntax

```
__property [in] BitRate;
```

CrcProtected

Syntax

```
__property [in] CrcProtected;
```

NumPrograms

Syntax

```
__property DWORD* NumPrograms;
```

PAT_Period

Syntax

```
__property [in] PAT_Period;
```

PCR_Period

Syntax

```
__property [in] PCR_Period;
```

PMT_Period

Syntax

```
__property [in] PMT_Period;
```

ProgNumBase

Syntax

```
__property [in] ProgNumBase;
```

RateMode

Syntax

```
__property [in] RateMode;
```

StreamID

Syntax

```
__property [in] StreamID;
```

TrailingAlignment

Syntax

```
__property [in] TrailingAlignment;
```

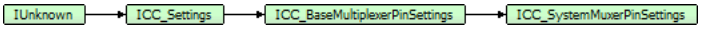
MPEG-1 System Stream

Interfaces

	Name	Description
	ICC_SystemMuxerPinSettings (see page 263)	
	ICC_SystemMuxerSettings (see page 265)	

ICC_SystemMuxerPinSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001B11-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_SystemMuxerPinSettings : ICC_BaseMultiplexerPinSettings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_BaseMultiplexerPinSettings Interface

	Name	Description
	BasePTS (see page 149)	
	BitRate (see page 149)	

	DataAlignPeriod (see page 149)	The period at which the access units will align to the packet's boundary. By default, audio streams aligned only at the beginning, video streams are aligned each I-frame.
	FrameRate (see page 149)	
	SampleRate (see page 149)	
	StreamID (see page 149)	
	StreamType (see page 149)	

ICC_SystemMuxerPinSettings Interface

	Name	Description
	PacketAlignment (see page 264)	
	PacketHeaderSize (see page 264)	
	PacketSize (see page 264)	
	PayloadAlignment (see page 264)	

Properties

PacketAlignment

Syntax

```
__property [in] PacketAlignment;
```

PacketHeaderSize

Syntax

```
__property [in] PacketHeaderSize;
```

PacketSize

Syntax

```
__property [in] PacketSize;
```

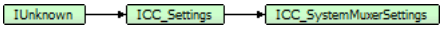
PayloadAlignment

Syntax

```
__property [in] PayloadAlignment;
```

ICC_SystemMuxerSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001B01-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_SystemMuxerSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_SystemMuxerSettings Interface

	Name	Description
	BitRate (see page 266)	
	PacketAlignment (see page 266)	
	PacketHeaderSize (see page 266)	
	PacketSize (see page 266)	
	PackHeaderPeriod (see page 266)	
	PayloadAlignment (see page 266)	
	RateMode (see page 266)	
	StandaloneSystemHeader (see page 266)	

	SuppressIsoEndCode (see page 267)	
	SystemHeaderPeriod (see page 267)	
	TrailingAlignment (see page 267)	

Properties

BitRate

Syntax

```
__property [in] BitRate;
```

PacketAlignment

Syntax

```
__property [in] PacketAlignment;
```

PacketHeaderSize

Syntax

```
__property [in] PacketHeaderSize;
```

PacketSize

Syntax

```
__property [in] PacketSize;
```

PackHeaderPeriod

Syntax

```
__property [in] PackHeaderPeriod;
```

PayloadAlignment

Syntax

```
__property [in] PayloadAlignment;
```

RateMode

Syntax

```
__property [in] RateMode;
```

StandaloneSystemHeader

Syntax

```
__property [in] StandaloneSystemHeader;
```

SuppressIsoEndCode

Syntax

```
__property [in] SuppressIsoEndCode;
```

SystemHeaderPeriod

Syntax

```
__property [in] SystemHeaderPeriod;
```

TrailingAlignment

Syntax

```
__property [in] TrailingAlignment;
```

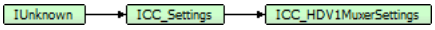
HDV-1 Multiplexer

Interfaces

	Name	Description
	ICC_HDV1MuxerSettings (see page 268)	

ICC_HDV1MuxerSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001D01-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_HDV1MuxerSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_HDV1MuxerSettings Interface

	Name	Description
	ProgNum (see page 269)	
	StreamID (see page 269)	

Properties

ProgNum

Syntax

```
__property [in] ProgNum;
```

StreamID

Syntax

```
__property [in] StreamID;
```

HDV-2 Multiplexer

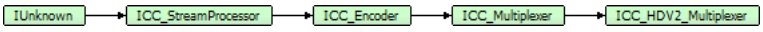
Interfaces

	Name	Description
	ICC_HDV2_Multiplexer (see page 270)	This interface provides a methods specific for the HDV2_Multilexer class.
	ICC_HDV2MuxerSettings (see page 272)	

ICC_HDV2_Multiplexer Interface

This interface provides a methods specific for the HDV2_Multilexer class.

Class Hierarchy



Syntax

```
[object, uuid(00001D12-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_HDV2_Multiplexer : ICC_Multiplexer;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_Multiplexer Interface

	Name	Description
	CreatePin (see page 145)	Method creates an input pin to add the specified type of elementary data needed to be multipolexed.
	CreatePinByXml (see page 146)	Method creates an input pin to add the specified type of elementary data needed to be multipolexed.

	GetPin (see page 146)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

ICC_Multiplexer Interface

	Name	Description
R	PinCount (see page 146)	/// Method creates an input pin to add the specified type of elementary data needed to be multipolexed. /// Returns: Returns S_OK if successful or error code otherwise. HRESULT CreatePinByType([in] CC_ELEMENTARY_STREAM_TYPE (see page 160) stream_type, // Pin type [out,retval] ICC_ByteStreamConsumer (see page 12) **pOutput // Address where pointer to the created object will be stored.);

ICC_HDV2_Multiplexer Interface

	Name	Description
	InitialTimeCode (see page 272)	

Properties

InitialTimeCode

Syntax

```
__property [in] InitialTimeCode;
```

ICC_HDV2MuxerSettings Interface

Class Hierarchy

Syntax

```
[object, uuid(00001D02-be08-11dc-aa88-005056c00008), pointer_default(unique), local]
interface ICC_HDV2MuxerSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_HDV2MuxerSettings Interface

	Name	Description
	ProgNum (see page 272)	
	StreamID (see page 273)	

Properties

ProgNum

Syntax

```
__property [in] ProgNum;
```

StreamID

Syntax

```
__property [in] StreamID;
```

MPEG-4 codec

This section describes the classes, interfaces and data types specific for the set of MPEG-4 formats (specification ISO/IEC 14496).

The section consist of three major parts - AAC Audio, AVC Video and MP4 Multiplex.

MPEG-4

(From Wikipedia, the free encyclopedia)

MPEG-4 is a collection of methods defining compression of audio and visual (AV) digital data. It was introduced in late 1998 and designated a standard for a group of audio and video coding formats and related technology agreed upon by the ISO/IEC Moving Picture Experts Group (MPEG) under the formal standard ISO/IEC 14496. Uses of MPEG-4 include compression of AV data for web (streaming media) and CD distribution, voice (telephone, videophone) and broadcast television applications.

MPEG-4 absorbs many of the features of MPEG-1 and MPEG-2 and other related standards, adding new features such as (extended) VRML support for 3D rendering, object-oriented composite files (including audio, video and VRML objects), support for externally-specified Digital Rights Management and various types of interactivity. AAC (Advanced Audio Codec) was standardized as an adjunct to MPEG-2 (as Part 7) before MPEG-4 was issued.

MPEG-4 is still a developing standard and is divided into a number of parts. Companies promoting MPEG-4 compatibility do not always clearly state which "part" level compatibility they are referring to. The key parts to be aware of are MPEG-4 part 2 (MPEG-4 SP/ASP, used by codecs such as DivX, Xvid, Nero Digital and 3ivx and by Quicktime 6) and MPEG-4 part 10 (MPEG-4 AVC/H.264, used by the x264 codec, by Nero Digital AVC, by Quicktime 7, and by next-gen DVD formats like HD DVD and Blu-ray Disc).

Most of the features included in MPEG-4 are left to individual developers to decide whether to implement them. This means that there are probably no complete implementations of the entire MPEG-4 set of standards. To deal with this, the standard includes the concept of "profiles" and "levels", allowing a specific set of capabilities to be defined in a manner appropriate for a subset of applications.

Initially, MPEG-4 was aimed primarily at low bit-rate video communications; however, its scope was later expanded to be much more of a multimedia coding standard. MPEG-4 is efficient

across a variety of bit-rates ranging from a few kilobits per second to tens of megabits per second. MPEG-4 provides the following functionalities:

- Improved coding efficiency;
- Ability to encode mixed media data (video, audio, speech);
- Error resilience to enable robust transmission;
- Ability to interact with the audio-visual scene generated at the receiver.

AVC/H.264 Video

H.264/MPEG-4 AVC

(From Wikipedia, the free encyclopedia)

MPEG-4 is a suite of standards which has many "parts", where each part standardizes various entities related to multimedia, such as audio, video, and file formats. To learn more about various parts and what they mean, please see the entry for MPEG-4 (see page 275).

H.264 is a standard for video compression, and is equivalent to MPEG-4 Part 10, or MPEG-4 AVC (for Advanced Video Coding). As of 2008, it is the latest block-oriented motion-compensation-based codec standard developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG), and it was the product of a partnership effort known as the Joint Video Team (JVT). The ITU-T H.264 standard and the ISO/IEC MPEG-4 Part 10 standard (formally, ISO/IEC 14496-10) are jointly maintained so that they have identical technical content. The final drafting work on the first version of the standard was completed in May 2003.

Overview

The intent of the H.264/AVC project was to create a standard capable of providing good video quality at substantially lower bit rates than previous standards (e.g. half or less the bit rate of MPEG-2, H.263, or MPEG-4 Part 2), without increasing the complexity of design so much that it would be impractical or excessively expensive to implement. An additional goal was to provide enough flexibility to allow the standard to be applied to a wide variety of applications on a wide variety of networks and systems, including low and high bit rates, low and high resolution video, broadcast, DVD storage, RTP/IP packet networks, and ITU-T multimedia telephony systems.

The standardization of the first version of H.264/AVC was completed in May 2003. The JVT then developed extensions to the original standard that are known as the Fidelity Range Extensions (FRExt). These extensions enable higher quality video coding by supporting increased sample bit depth precision and higher-resolution color information, including sampling structures known as YUV 4:2:2 and YUV 4:4:4. Several other features are also included in the Fidelity Range Extensions project, such as adaptive switching between 4?4 and 8?8 integer transforms, encoder-specified perceptual-based quantization weighting matrices, efficient inter-picture lossless coding, and support of additional color spaces. The design work on the Fidelity Range Extensions was completed in July 2004, and the drafting work on them was completed in September 2004.

Further recent extensions of the standard have included adding five new profiles intended primarily for professional applications, adding extended-gamut color space support, defining additional aspect ratio indicators, defining two additional types of "supplemental enhancement information" (post-filter hint and tone mapping), and deprecating one of the prior FRExt profiles that industry feedback indicated should have been designed differently.

Scalable Video Coding as specified in Annex G of H.264/AVC allows the construction of bitstreams that contain sub-bitstreams that conform to H.264/AVC. For temporal bitstream scalability, i.e., the presence of a sub-bitstream with a smaller temporal sampling rate than the bitstream, complete access units are removed from the bitstream when deriving the sub-bitstream. In this case, high-level syntax and inter prediction reference pictures in the bitstream are constructed accordingly. For spatial and quality bitstream scalability, i.e. the presence of a sub-bitstream with lower spatial resolution or quality than the bitstream, NAL units are removed from the bitstream when deriving the sub-bitstream. In this case, inter-layer prediction, i.e., the prediction of the higher spatial resolution or quality signal by data of the lower spatial resolution or quality signal, is typically used for efficient coding. The Scalable Video

Coding extension was completed in November 2007.

The H.264 name follows the ITU-T naming convention, where the standard is a member of the H.26x line of VCEG video coding standards; the MPEG-4 AVC name relates to the naming convention in ISO/IEC MPEG, where the standard is part 10 of ISO/IEC 14496, which is the suite of standards known as MPEG-4. The standard was developed jointly in a partnership of VCEG and MPEG, after earlier development work in the ITU-T as a VCEG project called H.26L. It is thus common to refer to the standard with names such as H.264/AVC, AVC/H.264, H.264/MPEG-4 AVC, or MPEG-4/H.264 AVC, to emphasize the common heritage. The name H.26L, referring to its ITU-T history, is less common, but still used. Occasionally, it is also referred to as "the JVT codec", in reference to the Joint Video Team (JVT) organization that developed it. (Such partnership and multiple naming is not uncommon — for example, the video codec standard known as MPEG-2 also arose from the partnership between MPEG and the ITU-T, where MPEG-2 video is known to the ITU-T community as H.262.)

Types

The types corresponding to the H.264 video codec will be listed below in this chapter.

Enumerations

	Name	Description
	CC_H264_DEBLOCKING_FILTER_MODE (see page 281)	H264 deblocking filter modes
	CC_H264_DIRECT_PRED_MODE (see page 281)	
	CC_H264_ENTROPY_CODING_MODE (see page 281)	
	CC_H264_FRAME_FLAGS (see page 282)	
	CC_H264_MOTION_FUNC (see page 282)	
	CC_H264_PROFILE (see page 282)	
	CC_H264_SCALING_MATRIX (see page 283)	Scaling matrices for 8x8 quantization.
	CC_H264_SUBBLOCK_SPLIT_MODE (see page 283)	

Structures

	Name	Description
	CC_H264_DEBLOCKING_FILTER_DESCR (see page 280)	H.264 deblocking filter parameters.

CC_H264_DEBLOCKING_FILTER_DESCR

H.264 deblocking filter parameters.

Syntax

```
struct CC_H264_DEBLOCKING_FILTER_DESCR {
  CC_H264_DEBLOCKING_FILTER_MODE Mode;
  CC_INT Alpha;
  CC_INT Beta;
};
```

Members

Alpha

(-6..6), specifies the offset used in accessing the alpha and tC0 deblocking filter tables. Refer H.264 8.7 "Deblocking filter process" for details.

Beta

(-6..6), specifies the offset used in accessing the beta deblocking filter table. Refer H.264 8.7 "Deblocking filter process" for details.

CC_H264_DEBLOCKING_FILTER_MODE

H264 deblocking filter modes

Syntax

```
[v1_enum]
enum CC_H264_DEBLOCKING_FILTER_MODE {
  CC_H264_DEBLOCKING_ON = 0,
  CC_H264_DEBLOCKING_OFF,
  CC_H264_DEBLOCKING_WITHIN_SLICE
};
```

Members

CC_H264_DEBLOCKING_ON

Deblocking filter is ON (by default).

CC_H264_DEBLOCKING_OFF

Deblocking filter is OFF.

CC_H264_DEBLOCKING_WITHIN_SLICE

Deblocking filter is ON but without crossing slice boundaries.

CC_H264_DIRECT_PRED_MODE

Syntax

```
[v1_enum]
enum CC_H264_DIRECT_PRED_MODE {
  CC_H264_DIRECT_PRED_NONE,
  CC_H264_DIRECT_PRED_TEMPORAL,
  CC_H264_DIRECT_PRED_SPATIAL
};
```

CC_H264_ENTROPY_CODING_MODE

Syntax

```
[v1_enum]
enum CC_H264_ENTROPY_CODING_MODE {
  CC_H264_CAVLC = 0,
  CC_H264_CABAC = 1,
  CC_H264_CABAC_0 = 1,
  CC_H264_CABAC_1 = 2,
  CC_H264_CABAC_2 = 3
};
```

Members

CC_H264_CAVLC
cavlc.

CC_H264_CABAC
cabac_init_idc = 0.

CC_H264_CABAC_0
cabac_init_idc = 0.

CC_H264_CABAC_1
cabac_init_idc = 1.

CC_H264_CABAC_2
cabac_init_idc = 2.

CC_H264_FRAME_FLAGS

Syntax

```
[v1_enum]
enum CC_H264_FRAME_FLAGS {
  CC_H264_FRAME_FLG_PROGRESSIVE_FRAME = 0x00000001,
  CC_H264_FRAME_FLG_TOP_FIELD_FIRST = 0x00000002,
  CC_H264_FRAME_IDR = 0x00001000,
  CC_H264_HDR_SEQ_PARAM_SET = 0x01000000,
  CC_H264_HDR_PIC_PARAM_SET = 0x02000000,
  CC_H264_HDR_AU_DELIMITER = 0x08000000
};
```

CC_H264_MOTION_FUNC

Syntax

```
[v1_enum]
enum CC_H264_MOTION_FUNC {
  CC_H264_ME_AUTO = -1,
  CC_H264_ME_FULL_SEARCH = 0,
  CC_H264_ME_CLASSIC_LOG = 1,
  CC_H264_ME_LOG = 2,
  CC_H264_ME_EPZS = 3,
  CC_H264_ME_FULL_ORTHOGONAL = 4,
  CC_H264_ME_LOG_ORTHOGONAL = 5,
  CC_H264_ME_TTS = 6
};
```

CC_H264_PROFILE

Syntax

```
[v1_enum]
enum CC_H264_PROFILE {
  CC_H264_PROFILE_UNKNOWN = 0,
  CC_H264_BASELINE_PROFILE = 66,
```

```
  CC_H264_MAIN_PROFILE = 77,
  CC_H264_EXTENDED_PROFILE = 88,
  CC_H264_HIGH_PROFILE = 100,
  CC_H264_HIGH_10_PROFILE = 110,
  CC_H264_HIGH_422_PROFILE = 122,
  CC_H264_HIGH_444_PROFILE = 144
};
```

CC_H264_SCALING_MATRIX

Scaling matrices for 8x8 quantization.

Syntax

```
[v1_enum]
enum CC_H264_SCALING_MATRIX {
  CC_H264_STANDARD_SCALING_MATRIX,
  CC_H264_DEFAULT_SCALING_MATRIX
};
```

CC_H264_SUBBLOCK_SPLIT_MODE

Syntax

```
[v1_enum]
enum CC_H264_SUBBLOCK_SPLIT_MODE {
  CC_H264_SUBBLK_NO_SPLIT = 0,
  CC_H264_SUBBLK_SPLIT_8x8,
  CC_H264_SUBBLK_SPLIT_4x4
};
```

Members

CC_H264_SUBBLK_NO_SPLIT

No macroblock split (16x16) used in ME.

CC_H264_SUBBLK_SPLIT_8x8

Up to 4 subblocks (16x8, 8x16, 8x8 in any combinations).

CC_H264_SUBBLK_SPLIT_4x4

Up to 16 subblocks (8x4, 4x8, 4x4, in any combinations) - not implemented yet.

Interfaces

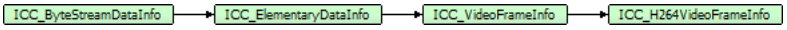
Interfaces

	Name	Description
	ICC_H264VideoFrameInfo (see page 284)	Provides the particular H.264 video frame description.
	ICC_H264VideoStreamInfo (see page 286)	The interface represents H.264 video bitstream parameters.

ICC_H264VideoFrameInfo Interface

Provides the particular H.264 video frame description.

Class Hierarchy



Syntax

```
[object, uuid(ecfd3260-3a6b-474a-94e5-7d35c7482c8b), pointer_default(unique), local]
interface ICC_H264VideoFrameInfo : ICC_VideoFrameInfo;
```

Methods

ICC_H264VideoFrameInfo Interface

	Name	Description
	GetUserData (see page 285)	Retrieves the specified user data which belongs to the video frame.

Properties

	Name	Description
R	DTS (see page 49)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 49), except the coded video with B-frames.
R	Duration (see page 49)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
R	NumSamples (see page 49)	Number of samples of the elementary data. In case of coded video frames, there is usually 1 or 2 sample(s) (1 frame or 2 fields). In case of coded audio, there are the number of audio samples, encoded into audio frame.

	PresentationDelta (see page 49)	The presentation delta, in samples, of the elementary data. It is actual for data which was reordered during encoding process (f.e. coded video with B-frames).
	PTS (see page 49)	The Presentation Data Stamp (PTS) of the elementary data. The PTS based on CC_TIMEBASE, specified for object which generates the elementary data.
	SampleOffset (see page 49)	The frame's first sample order number.
	SequenceEntryFlag (see page 50)	The sequence entry point flag. In the case of mpeg video, it means that SEQUENCE_HEADER presents in the elementary data. This flag is used to signal the multiplexers to generate the entry point (like System Header) at this elementary data.

ICC_VideoFrameInfo Interface

	Name	Description
	CodingNumber (see page 97)	The number of video frame in the coding order. Zero-based.
	Flags (see page 97)	Various flags of the coded video frame. Value of this field dependend of the video stream type.
	FrameType (see page 97)	The frame coding type (see page 107).
	InterlaceType (see page 98)	The field order of video frame.
	Number (see page 98)	The number of video frame in native (display) order. zero-based.
	PictStruct (see page 98)	The picture structure of the MPEG video frame.
	TimeCode (see page 98)	The timecode of video frame.

ICC_H264VideoFrameInfo Interface

	Name	Description
	UserDataCount (see page 286)	The number of user_data SEI messages, associated with the video frame.

Methods

GetUserData

Retrieves the specified user data which belongs to the video frame.

Syntax

```
HRESULT GetUserData(
```

```
    [in] DWORD dwUserDataNumber,
    [out, size_is(cbBufSize)] BYTE * pData,
    [in] DWORD cbBufSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

dwUserDataNumber

Specified the user data number, zero-based.

pData

Place to store the user data, if NULL the only size of the specified user data will be returned.

cbBufSize

Buffer size.

pcbRetSize

Place to store the user data size.

Returns

Returns S_OK if successful or E_INVALIDARG in case of incorrect *dwUserDataNumber*.

Properties

UserDataCount

The number of user_data SEI messages, associated with the video frame.

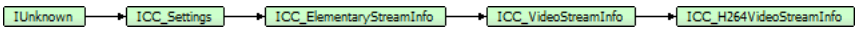
Syntax

```
__property DWORD * UserDataCount;
```

ICC_H264VideoStreamInfo Interface

The interface represents H.264 video bitstream parameters.

Class Hierarchy



Syntax

```
[object, uuid(63cf41b3-b6ac-448f-ac25-5b2160825d9c), pointer_default(unique), local]
interface ICC_H264VideoStreamInfo : ICC_VideoStreamInfo;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.

	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ElementaryStreamInfo Interface

	Name	Description
R	BitRate (see page 52)	The bitrate (max) of the elementary stream.
R	FrameRate (see page 52)	The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.
R	StreamType (see page 52)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

ICC_VideoStreamInfo Interface

	Name	Description
R	AspectRatio (see page 100)	The aspect ratio cx:cy.
R	FrameSize (see page 100)	The size in pixels of video frame.
R	ProgressiveSequence (see page 100)	If TRUE - all frames in the stream coded without fields (progressive).

ICC_H264VideoStreamInfo Interface

	Name	Description
R	BitDepthChroma (see page 288)	The chroma samples' bit depth.
R	BitDepthLuma (see page 288)	The luma samples' bit depth
R	BitRate (see page 288)	The video bitstream bitrate.
R	ChromaFormat (see page 288)	Chroma format.

	ColorCoefs (see page 288)	The color transformarion description.
	Level (see page 289)	The Level is a specified set of constraints imposed on values of the syntax elements in the bitstream. Levels are specified within each profile. For more information please refer to the H.264 standard Annex A.
	Profile (see page 289)	The Profile of an H.264 stream. For more information please refer to the H.264 standard Annex A.
	VideoFormat (see page 289)	The video format.

Properties

BitDepthChroma

The chroma samples' bit depth.

Syntax

```
__property DWORD* BitDepthChroma;
```

BitDepthLuma

The luma samples' bit depth

Syntax

```
__property DWORD* BitDepthLuma;
```

BitRate

The video bitstream bitrate.

Syntax

```
__property CC_BITRATE * BitRate;
```

ChromaFormat

Chroma format.

Syntax

```
__property CC_CHROMA_FORMAT * ChromaFormat;
```

ColorCoefs

The color transformarion description.

Syntax

```
__property CC_COLOUR_DESCRIPTION* ColorCoefs;
```

Level

The Level is a specified set of constraints imposed on values of the syntax elements in the bitstream. Levels are specified within each profile. For more information please refer to the H.264 standard Annex A.

Syntax

```
__property DWORD * Level;
```

Profile

The Profile of an H.264 stream. For more information please refer to the H.264 standard Annex A.

Syntax

```
__property CC_H264_PROFILE * Profile;
```

VideoFormat

The video format.

Syntax

```
__property CC_VIDEO_FORMAT * VideoFormat;
```

H.264 Video Decoder

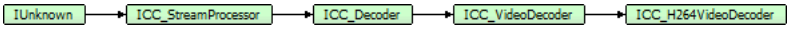
Interfaces

	Name	Description
	ICC_H264VideoDecoder (see page 290)	The interface gives the control to the instance of CC_H264VideoDecoder class.
	ICC_AVC1VideoDecoder (see page 291)	The interface gives the control to the instance of CC_AVC1VideoDecoder class. The main reason to implement such decoder is that AVC1 format is differs from elementary H.264 stream by changing the startcodes prefixes 0x00000001 to the lengths of corresponding NALUs. The data should start with chunk start and should contain a whole number of chunks.

ICC_H264VideoDecoder Interface

The interface gives the control to the instance of CC_H264VideoDecoder class.

Class Hierarchy



Syntax

```
[object, uuid(f7b72085-b7b8-42a2-a6ec-e81814e84f32), pointer_default(unique), local]
interface ICC_H264VideoDecoder : ICC_VideoDecoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().

	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_VideoDecoder Interface

	Name	Description
	GetFrame (see page 91)	Retrieve the current video frame.
	GetStride (see page 91)	
	GetVideoFrameInfo (see page 91)	Get the description of current video frame.
	GetVideoStreamInfo (see page 92)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 92)	

Properties

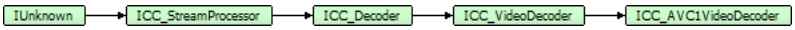
	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_AVC1VideoDecoder Interface

The interface gives the control to the instance of CC_AVC1VideoDecoder class. The main reason to implement such decoder is that AVC1 format is differs from elementary H.264 stream by changing the startcodes prefixes 0x00000001 to the lengths of corresponding NALUs. The

data should start with chunk start and should contain a whole number of chunks.

Class Hierarchy



Syntax

```
[object, uuid(0976d3ec-341a-4805-954a-3a8cb0d1d33a), pointer_default(unique), local]
interface ICC_AVC1VideoDecoder : ICC_VideoDecoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_VideoDecoder Interface

	Name	Description
	GetFrame (see page 91)	Retrieve the current video frame.
	GetStride (see page 91)	
	GetVideoFrameInfo (see page 91)	Get the description of current video frame.
	GetVideoStreamInfo (see page 92)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 92)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

H.264 Video Encoder

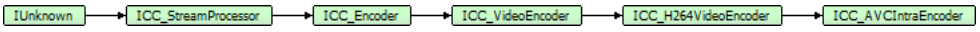
Interfaces

	Name	Description
	ICC_H264VideoEncoder (see page 294)	Interface gives the control to the instance of CC_H264VideoEncoder class.
	ICC_H264VideoEncoderSettings (see page 297)	The settings for initialization the instance of CinecoderH264VideoEncoder class.

ICC_H264VideoEncoder Interface

Interface gives the control to the instance of CC_H264VideoEncoder class.

Class Hierarchy



Syntax

```
[object, uuid(60DAA884-1861-4771-BF4D-4A82570DDC8E), pointer_default(unique), local]
interface ICC_H264VideoEncoder : ICC_VideoEncoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_VideoEncoder Interface

	Name	Description
	AddFrame (see page 94)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 94)	Add a frame with different size to the processing queue.
	GetStride (see page 95)	

	GetVideoFrameInfo (see page 95)	Get the description of current video frame.
	GetVideoStreamInfo (see page 95)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 95)	
	IsScaleAvailable (see page 96)	

ICC_H264VideoEncoder Interface

	Name	Description
	AddUserData (see page 296)	Adds user data for the subsequent video frame.
	GetH264VideoFrameInfo (see page 296)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

ICC_H264VideoEncoder Interface

	Name	Description
	ThreadsAffinity (see page 296)	The affinity mask for object's threads. 0 = default (current process) affinity mask

	ThreadsCount (see page 297)	Maximum number of threads which object can use. 0 = automatic.
	ThreadsPriority (see page 297)	The pripority for object's threads.

Methods

AddUserData

Adds user data for the subsequent video frame.

Syntax

```
HRESULT AddUserData(
    [in, size_is(cbSize)] const BYTE * pbUserData,
    [in] DWORD cbSize,
    [in,defaultvalue(CC_FALSE)] CC_BOOL bSecondField
);
```

Parameters

pbUserData

The user data.

cbSize

The size of the user data, in bytes.

bSecondField

Tells that incoming user data must appear at the second field (in case of interlaced coding).

Returns

Returns S_OK if successful or an error code otherwise.

Notes

You may call AddUserData several times to add more than one user data.

GetH264VideoFrameInfo

Syntax

```
HRESULT GetH264VideoFrameInfo(
    DWORD field_no,
    [out,retval] ICC_H264VideoFrameInfo ** pDescr
);
```

Properties

ThreadsAffinity

The affinity mask for object's threads. 0 = default (current process) affinity mask

Syntax

```
__property [in] ThreadsAffinity;
```

ThreadsCount

Maximum number of threads which object can use. 0 = automatic.

Syntax

```
__property [in] ThreadsCount;
```

ThreadsPriority

The pripority for object's threads.

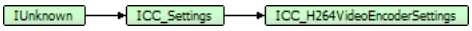
Syntax

```
__property [in] ThreadsPriority;
```

ICC_H264VideoEncoderSettings Interface

The settings for initialization the instance of CinecoderH264VideoEncoder class.

Class Hierarchy



Syntax

```
[object, uuid(7baac45c-1da0-4fa6-b645-f9afdd84246f), pointer_default(unique), local]
interface ICC_H264VideoEncoderSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_H264VideoEncoderSettings Interface

	Name	Description
	AspectRatio (see page 300)	The aspect ratio cx:cy.
	AvgBitRate (see page 300)	The average bitrate in VBR mode.
	BitDepthChroma (see page 300)	The chroma samples' bit depth (8-12).
	BitDepthLuma (see page 300)	The luma samples' bit depth (8-12).
	BitRate (see page 300)	The target bitrate in CBR mode or max bitrate in VBR mode.
	BlurFilterCoef (see page 301)	Blur filter coefficient. Simple 3x3 matrix of ones, with specified central coef (lower value means stronger blurring). 0 = disable blur. negative values = auto blur
	ChromaFormat (see page 301)	The chroma resolution format (4:2:0, 4:2:2 or 4:4:4).
	ChromaQPOffset (see page 301)	The offset from the luma QP (see page 304) value for chroma components
	ColorCoefs (see page 301)	Sequence display extension - color coefficients.
	CpbSize (see page 301)	The Coded Picture Buffer (CPB) size. You can set it up either by bytes or milliseconds (just add FRQ_TIMEVAL_MS flag to the value)
	DeblockingFilter (see page 301)	Deblocking filter params.
	Deinterlace (see page 301)	Deinterlace method
	DirectPredMode (see page 302)	Direct Prediction Mode.
	DisableSceneDetector (see page 302)	Enable/disable the built-in scene change detector (which is ON by default).
	Enable8x8Transform (see page 302)	
	EntropyCodingMode (see page 302)	The entropy coding method.
	FrameRate (see page 302)	The frame rate of video stream.

	FrameSize (see page 302)	The physical frame size, in pixels.
	GOP (see page 302)	The GOP settings. See CC_GOP_DESCR (see page 110) for details.
	IDR_Period (see page 302)	The Instantaneous Decoding Refresh period.
	InitialCpbLevel (see page 303)	The initial CPB fullness. You can specify it either in bytes or milliseconds.
	InterlaceType (see page 303)	The field order video.
	Level (see page 303)	The level of destination stream.
	MB_Struct (see page 303)	Field/frame macrobloc structure.
	MinBitRate (see page 303)	The minimal bitrate in VBR mode (default is 0).
	MotionFunc (see page 303)	The motion estimation method.
	MotionWindow (see page 303)	The motion estimation window.
	NumRefFrames (see page 304)	The number of reference frames.
	NumSlices (see page 304)	The max number of slices per frame.
	PictureStructure (see page 304)	The picture structure.
	Profile (see page 304)	The destination stream profile.
	PutAccessUnitDelimiter (see page 304)	Toggle access unit delimiter (nal_unit_type = 9) on/off.
	PutSeqEndCode (see page 304)	Put sequence_end_code after the last coded picture.
	QP (see page 304)	The initial quantization parameter for VBR mode.
	QPPrimeY0TransformBypass (see page 305)	
	RateMode (see page 305)	The mode of bitrate controller - see CC_BITRATE_MODE.
	ScalingMatrix (see page 305)	Standard/Default scaling matrix for 8x8 transform.
	SequenceHeaderPeriod (see page 305)	You can also specify the period in milliseconds by adding the FRQ_TIMEVAL_MS flag to the value.

	SubBlockSplitMode (see page 305)	The subblock split mode.
	UseWeightedBiPrediction (see page 305)	Weighted BiPrediction.
	UseWeightedPrediction (see page 305)	Weighted Prediction.
	VideoFormat (see page 305)	The video format.

Properties

AspectRatio

The aspect ratio cx:cy.

Syntax

```
__property [in] AspectRatio;
```

AvgBitRate

The average bitrate in VBR mode.

Syntax

```
__property [in] AvgBitRate;
```

BitDepthChroma

The chroma samples' bit depth (8-12).

Syntax

```
__property [in] BitDepthChroma;
```

BitDepthLuma

The luma samples' bit depth (8-12).

Syntax

```
__property [in] BitDepthLuma;
```

BitRate

The target bitrate in CBR mode or max bitrate in VBR mode.

Syntax

```
__property [in] BitRate;
```

BlurFilterCoef

Blur filter coefficient. Simple 3x3 matrix of ones, with specified central coef (lower value means stronger blurring). 0 = disable blur. negative values = auto blur

Syntax

```
__property CC_INT BlurFilterCoef;
```

ChromaFormat

The chroma resolution format (4:2:0, 4:2:2 or 4:4:4).

Syntax

```
__property [in] ChromaFormat;
```

ChromaQPOffset

The offset from the luma QP (see page 304) value for chroma components

Syntax

```
__property CC_INT ChromaQPOffset;
```

ColorCoefs

Sequence display extension - color coefficients.

Syntax

```
__property [in] ColorCoefs;
```

CpbSize

The Coded Picture Buffer (CPB) size. You can set it up either by bytes or milliseconds (just add FRQ_TIMEVAL_MS flag to the value)

Syntax

```
__property CC_PERIOD CpbSize;
```

DeblockingFilter

Deblocking filter params.

Syntax

```
__property [in] DeblockingFilter;
```

Deinterlace

Deinterlace method

Syntax

```
__property CC_DEINTERLACE_METHOD Deinterlace;
```

DirectPredMode

Direct Prediction Mode.

Syntax

```
__property [in] DirectPredMode;
```

DisableSceneDetector

Enable/disable the built-in scene change detector (which is ON by default).

Syntax

```
__property [in] DisableSceneDetector;
```

Enable8x8Transform

Syntax

```
__property [in] Enable8x8Transform;
```

EntropyCodingMode

The entropy coding method.

Syntax

```
__property [in] EntropyCodingMode;
```

FrameRate

The frame rate of video stream.

Syntax

```
__property [in] FrameRate;
```

FrameSize

The physical frame size, in pixels.

Syntax

```
__property [in] FrameSize;
```

GOP

The GOP settings. See CC_GOP_DESCR (see page 110) for details.

Syntax

```
__property [in] GOP;
```

IDR_Period

The Instantaneous Decoding Refresh period.

Syntax

```
__property [in] IDR_Period;
```

InitialCpbLevel

The initial CPB fullness. You can specify it either in bytes or milliseconds.

Syntax

```
__property CC_PERIOD InitialCpbLevel;
```

InterlaceType

The field order video.

Syntax

```
__property [in] InterlaceType;
```

Level

The level of destination stream.

Syntax

```
__property [in] Level;
```

MB_Struct

Field/frame macrobloc structure.

Syntax

```
__property [in] MB_Struct;
```

MinBitRate

The minimal bitrate in VBR mode (default is 0).

Syntax

```
__property [in] MinBitRate;
```

MotionFunc

The motion estimation method.

Syntax

```
__property [in] MotionFunc;
```

MotionWindow

The motion estimation window.

Syntax

```
__property [in] MotionWindow;
```

NumRefFrames

The number of reference frames.

Syntax

```
__property [in] NumRefFrames;
```

NumSlices

The max number of slices per frame.

Syntax

```
__property [in] NumSlices;
```

PictureStructure

The picture structure.

Syntax

```
__property [in] PictureStructure;
```

Profile

The destination stream profile.

Syntax

```
__property [in] Profile;
```

PutAccessUnitDelimiter

Toggle access unit delimiter (nal_unit_type = 9) on/off.

Syntax

```
__property [in] PutAccessUnitDelimiter;
```

PutSeqEndCode

Put sequence_end_code after the last coded picture.

Syntax

```
__property CC_BOOL PutSeqEndCode;
```

QP

The initial quantization parameter for VBR mode.

Syntax

```
__property [in] QP;
```

QPPrimeY0TransformBypass

Syntax

```
__property [in] QPPrimeY0TransformBypass;
```

RateMode

The mode of bitrate controller - see CC_BITRATE_MODE.

Syntax

```
__property [in] RateMode;
```

ScalingMatrix

Standard/Default scaling matrix for 8x8 transform.

Syntax

```
__property [in] ScalingMatrix;
```

SequenceHeaderPeriod

You can also specify the period in milliseconds by adding the FRQ_TIMEVAL_MS flag to the value.

Syntax

```
__property CC_PERIOD SequenceHeaderPeriod;
```

SubBlockSplitMode

The subblock split mode.

Syntax

```
__property [in] SubBlockSplitMode;
```

UseWeightedBiPrediction

Weighted BiPrediction.

Syntax

```
__property [in] UseWeightedBiPrediction;
```

UseWeightedPrediction

Weighted Prediction.

Syntax

```
__property [in] UseWeightedPrediction;
```

VideoFormat

The video format.

Syntax

```
__property [in] VideoFormat;
```

AVC-Intra Encoder

Interfaces

	Name	Description
	ICC_AVCIntraEncoder (see page 309)	Interface for AVC Intra video encoder
	ICC_AVCIntraEncoderSettings (see page 311)	The Settings for AVC-Intra Encoder initialization.

Overview

AVC-Intra is a type of video coding developed by Panasonic that is fully compliant with the H.264/MPEG-4 AVC standard and additionally follows the SMPTE RP 2027-2007 recommended practice specification. AVC-Intra is available in a number of Panasonic's high definition broadcast products, such as, for example, their P2 card equipped broadcast cameras. It is now also supported in various products made by other companies.

Panasonic announced AVC-Intra codec support in April 2007. The use of AVC-Intra provides production quality HD video at bit rates more normally associated with ENG (Electronic news gathering) applications, permitting full resolution, 10 bit field capture of high quality HD imagery in one piece camera-recorders.

AVC-Intra is intended to serve needs of video professionals who have to store HD digital video on digital storage media for editing and archiving purposes. It defines 10-bit intra-frame only compression, which is easy for editing and preserves maximum video quality. The new standard significantly outperforms the older HDV (MPEG2 based) and DVCPRO HD (DV based) formats, allowing the codec to maintain better quality in 2x less storage.

There are two classes:

AVC-Intra 50

- nominally 50 Mbit/s, size of each frame is fixed;
- CABAC entropy coding only;
- 1920x1080 formats are High 10 Intra Profile, Level 4;
- 1280x720 formats are High 10 Intra Profile, Level 3.2;
- 4:2:0 chrominance sampling;
- frames are horizontally scaled by 3/4 (1920x1080 is scaled to 1440x1080. 1280x720 is scaled

to 960x720).

AVC-Intra 100

- nominally 100 Mbit/s, size of each frame is fixed;
- CAVLC entropy coding only;
- All formats are High 4:2:2 Intra Profile, Level 4.1;
- 4:2:2 chrominance sampling;
- frames are not scaled.

Common to both classes:

- Frame rates: 1920x1080 (23.98p / 25p / 29.97p / 50i / 59.94i), 1280x720 (23.98p / 25p / 29.97p / 50p / 59.94p);
- 10 bit luma and chroma.

Types

Enumerations

	Name	Description
	CC_AVCI_MODE (see page 308)	

CC_AVCI_MODE

Syntax

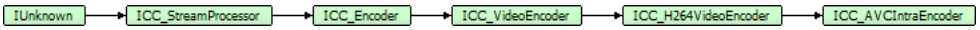
```
[v1_enum]
enum CC_AVCI_MODE {
  CC_AVCI_MODE_UNKNOWN = 0x00,
  CC_AVCI_50_720P_2398 = 0x100,
  CC_AVCI_50_720P_25 = 0x101,
  CC_AVCI_50_720P_2997 = 0x102,
  CC_AVCI_50_720P_50 = 0x09,
  CC_AVCI_50_720P_5994 = 0x08,
  CC_AVCI_50_1080P_2398 = 0x104,
  CC_AVCI_50_1080P_25 = 0x04,
  CC_AVCI_50_1080P_2997 = 0x03,
  CC_AVCI_50_1080I_50 = 0x02,
  CC_AVCI_50_1080I_5994 = 0x01,
  CC_AVCI_100_720P_2398 = 0x105,
  CC_AVCI_100_720P_25 = 0x106,
  CC_AVCI_100_720P_2997 = 0x107,
  CC_AVCI_100_720P_50 = 0x29,
  CC_AVCI_100_720P_5994 = 0x28,
  CC_AVCI_100_1080P_2398 = 0x108,
  CC_AVCI_100_1080P_25 = 0x24,
  CC_AVCI_100_1080P_2997 = 0x23,
  CC_AVCI_100_1080I_50 = 0x22,
```

```
  CC_AVCI_100_1080I_5994 = 0x21,
  CC_AVCI_4K_422_2398 = 0x400,
  CC_AVCI_4K_422_25 = 0x401,
  CC_AVCI_4K_422_2997 = 0x402,
  CC_AVCI_4K_422_50 = 0x403,
  CC_AVCI_4K_422_5994 = 0x404
};
```

ICC_AVCIntraEncoder Interface

Interface for AVC Intra video encoder

Class Hierarchy



Syntax

```
[object, uuid(0242b581-180a-430e-bb9c-1a78e60de3e5), pointer_default(unique), local]
interface ICC_AVCIntraEncoder : ICC_H264VideoEncoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_VideoEncoder Interface

	Name	Description
	AddFrame (see page 94)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 94)	Add a frame with different size to the processing queue.
	GetStride (see page 95)	
	GetVideoFrameInfo (see page 95)	Get the description of current video frame.
	GetVideoStreamInfo (see page 95)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 95)	

	IsScaleAvailable (see page 96)	

ICC_H264VideoEncoder Interface

	Name	Description
	AddUserData (see page 296)	Adds user data for the subsequent video frame.
	GetH264VideoFrameInfo (see page 296)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

ICC_H264VideoEncoder Interface

	Name	Description
	ThreadsAffinity (see page 296)	The affinity mask for object's threads. 0 = default (current process) affinity mask
	ThreadsCount (see page 297)	Maximum number of threads which object can use. 0 = automatic.
	ThreadsPriority (see page 297)	The pripority for object's threads.

ICC_AVCIntraEncoder Interface

	Name	Description
	InitialTimeCode (see page 311)	Specifies the initial timecode for the first frame.

Properties

InitialTimeCode

Specifies the initial timecode for the first frame.

Syntax

```
__property CC_TIMECODE InitialTimeCode;
```

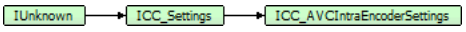
Notes

Should be used before first call to AddFrame (see page 94).

ICC_AVCIntraEncoderSettings Interface

The Settings for AVC-Intra Encoder initialization.

Class Hierarchy



Syntax

```
[object, uuid(54761d8c-4180-46c7-8364-144a64ed1e8e), pointer_default(unique), local]
interface ICC_AVCIntraEncoderSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_AVCIntraEncoderSettings Interface

	Name	Description
	ColorCoefs (see page 312)	Sequence display extension - color coefficients.
	Mode (see page 312)	The AVC Intra encoder mode.
	NumSingleEncoders (see page 312)	Number of single encoders to be used.
	SequenceHeaderPeriod (see page 313)	The Frequency of SPS/PPS generation, in frames. See CC_PERIOD_FLAGS for details. CC_ONCE(0) = only at the beginnig, FRQ_FOREVER(1) = at each frame. You can also specify the period in milliseconds by adding the FRQ_TIMEVAL_MS flag to the value. The default value is CC_ONCE.

Properties

ColorCoefs

Sequence display extension - color coefficients.

Syntax

```
__property CC_COLOUR_DESCRIPTION ColorCoefs;
```

Mode

The AVC Intra encoder mode.

Syntax

```
__property CC_AVCI_MODE Mode;
```

NumSingleEncoders

Number of single encoders to be used.

Syntax

```
__property CC_UINT NumSingleEncoders;
```

SequenceHeaderPeriod

The Frequency of SPS/PPS generation, in frames. See CC_PERIOD_FLAGS for details. CC_ONCE(0) = only at the beginnig, FRQ_FOREVER(1) = at each frame. You can also specify the period in milliseconds by adding the FRQ_TIMEVAL_MS flag to the value. The default value is CC_ONCE.

Syntax

```
__property CC_PERIOD SequenceHeaderPeriod;
```

AAC Audio

Overview

Advanced Audio Coding (AAC) is a standardized, lossy compression and encoding scheme for digital audio. Designed to be the successor of the MP3 format, AAC generally achieves better sound quality than MP3 at similar bit rates.

Types

Enumerations

	Name	Description
	CC_AAC_FORMAT (see page 316)	
	CC_AAC_PROFILE (see page 316)	AAC Profiles

CC_AAC_FORMAT

Syntax

```
[v1_enum]
enum CC_AAC_FORMAT {
  CC_AAC_FMT_ADTS = 0,
  CC_AAC_FMT_ADIF = 1,
  CC_AAC_FMT_RAW = 2,
  CC_AAC_FMT_LATM = 3
};
```

Members

CC_AAC_FMT_ADTS

ADTS (Audio Data Transport Stream) streams have headers for each raw data block, thus making it more suitable for streaming.

CC_AAC_FMT_ADIF

ADIF (Audio Data Interchange Format) streams have the header at the beginnig only.

CC_AAC_FMT_RAW

No headers, raw data blocks only.

CC_AAC_FMT_LATM

MPEG-4 14496-3 stream type

CC_AAC_PROFILE

AAC Profiles

Syntax

```
[v1_enum]
enum CC_AAC_PROFILE {
  CC_AAC_PROFILE_UNKNOWN = 0,
```

```
  CC_AAC_PROFILE_MAIN = 1,
  CC_AAC_PROFILE_LC = 2,
  CC_AAC_PROFILE_SSR = 3,
  CC_AAC_PROFILE_LTP = 4,
  CC_AAC_PROFILE_SBR = 16,
  CC_AAC_PROFILE_PS = 32,
  CC_AAC_PROFILE_HE = CC_AAC_PROFILE_LC|CC_AAC_PROFILE_SBR,
  CC_AAC_PROFILE_HE2 = CC_AAC_PROFILE_LC|CC_AAC_PROFILE_SBR|CC_AAC_PROFILE_PS
};
```

Members

CC_AAC_PROFILE_UNKNOWN

Unknown.

CC_AAC_PROFILE_MAIN

Main Profile.

CC_AAC_PROFILE_LC

Low Complexity Profile.

CC_AAC_PROFILE_SSR

Scalable Sampling Rate Profile.

CC_AAC_PROFILE_LTP

Long Term Predictor Profile.

CC_AAC_PROFILE_SBR

+Spectral Band Replication

CC_AAC_PROFILE_PS

+Parametric Stereo

Interfaces

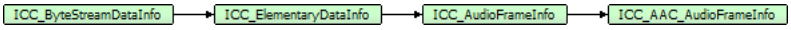
Interfaces

	Name	Description
	ICC_AAC_AudioFrameInfo (see page 318)	The information about a particular AAC audio frame.
	ICC_AAC_AudioStreamInfo (see page 319)	Represents the AAC-specified video stream description.

ICC_AAC_AudioFrameInfo Interface

The information about a particular AAC audio frame.

Class Hierarchy



Syntax

```
[object, uuid(a212acd4-c2d2-45ed-856f-0cae23f14352), pointer_default(unique), local]
interface ICC_AAC_AudioFrameInfo : ICC_AudioFrameInfo;
```

Properties

	Name	Description
R	DTS (see page 49)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 49), except the coded video with B-frames.
R	Duration (see page 49)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
R	NumSamples (see page 49)	Number of samples of the elementary data. In case of coded video frames, there is usually 1 or 2 sample(s) (1 frame or 2 fields). In case of coded audio, there are the number of audio samples, encoded into audio frame.
R	PresentationDelta (see page 49)	The presentation delta, in samples, of the elementary data. It is actual for data which was reordered during encoding process (f.e. coded video with B-frames).
R	PTS (see page 49)	The Presentation Data Stamp (PTS) of the elementary data. The PTS based on CC_TIMEBASE, specified for object which generates the elementary data.

	SampleOffset (see page 49)	The frame's first sample order number.
	SequenceEntryFlag (see page 50)	The sequence entry point flag. In the case of mpeg video, it means that SEQUENCE_HEADER presents in the elementary data. This flag is used to signal the multiplexers to generate the entry point (like System Header) at this elementary data.

ICC_AAC_AudioStreamInfo Interface

Represents the AAC-specified video stream description.

Class Hierarchy

Syntax

```
[object, uuid(8d85d96b-1359-4173-99f1-d940420426cc), pointer_default(unique), local]
interface ICC_AAC_AudioStreamInfo : ICC_AudioStreamInfo;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ElementaryStreamInfo Interface

	Name	Description
	BitRate (see page 52)	The bitrate (max) of the elementary stream.
	FrameRate (see page 52)	The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.

	StreamType (see page 52)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

ICC_AudioStreamInfo Interface

	Name	Description
	BitsPerSample (see page 133)	The bit depth of one audio sample (of one channel).
	ChannelMask (see page 133)	The mask of channels. See the CC_AUDIO_CHANNEL_MASK for details
	NumChannels (see page 133)	The number of channels in the waveform-audio data.
	SampleRate (see page 133)	The audio sampling frequency.

ICC_AAC_AudioStreamInfo Interface

	Name	Description
	Format (see page 320)	AAC format.
	Profile (see page 320)	AAC profile.

Properties

Format

AAC format.

Syntax

```
__property CC_AAC_FORMAT * Format;
```

Profile

AAC profile.

Syntax

```
__property CC_AAC_PROFILE * Profile;
```

AAC Encoder

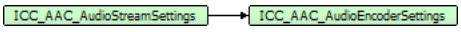
Interfaces

	Name	Description
	ICC_AAC_AudioEncoderSettings (see page 321)	The settings for the CinecoderMpegAudioEncoder initialization.
	ICC_AAC_AudioEncoder (see page 321)	The main interface to access the CC_AAC_AudioEncoder class.

ICC_AAC_AudioEncoderSettings Interface

The settings for the CinecoderMpegAudioEncoder initialization.

Class Hierarchy



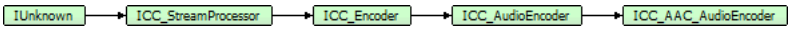
Syntax

```
[object, uuid(393e9fcf-eed3-4e74-86d1-2ce034bc680c), pointer_default(unique), local]
interface ICC_AAC_AudioEncoderSettings : ICC_AAC_AudioStreamSettings;
```

ICC_AAC_AudioEncoder Interface

The main interface to access the CC_AAC_AudioEncoder class.

Class Hierarchy



Syntax

```
[object, uuid(b0fb4156-bb6a-4000-b3c5-c75c93f607a7), pointer_default(unique), local]
interface ICC_AAC_AudioEncoder : ICC_AudioEncoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_AudioEncoder Interface

	Name	Description
	GetAudioFrameInfo (see page 129)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 129)	Get the description of audio stream which is being decoded.
	ProcessAudio (see page 130)	Puts another audio uncompressed data to the audio consumer.

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

AAC Decoder

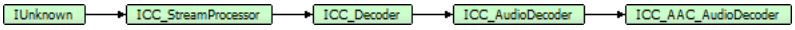
Interfaces

	Name	Description
	ICC_AAC_AudioDecoder (see page 323)	The main interface to access the CC_AAC_AudioDecoder class.

ICC_AAC_AudioDecoder Interface

The main interface to access the CC_AAC_AudioDecoder class.

Class Hierarchy



Syntax

```
[object, uuid(461528f1-d3d3-4349-b728-c5a27ed9afdc), pointer_default(unique), local]
interface ICC_AAC_AudioDecoder : ICC_AudioDecoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_AudioDecoder Interface

	Name	Description
	GetAudio (see page 125)	Retrieves the uncompressed audio data from the audio producer. Remark: To obtain the size of the buffer to hold the resulted samples, put NULL instead of pbData.
	GetAudioFrameInfo (see page 126)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 126)	Get the description of the audio stream.
	GetSampleBytes (see page 126)	
	IsFormatSupported (see page 127)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_AudioDecoder Interface

	Name	Description
R	NumSamples (see page 127)	The number of ready audio samples at the output.

MP4 Multiplex

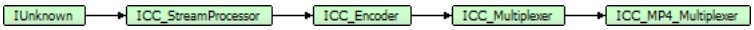
Interfaces

	Name	Description
	ICC_MP4_Multiplexer (see page 326)	The interface to operate with the MP4 multiplexer.
	ICC_MP4_MultiplexerSettings (see page 328)	The interface to operate with MP4 multiplexer.
	ICC_MP4_MuxerPinSettings (see page 330)	The major stream properties for creating the pin in the multiplexer. Only StreamType field is mandatory - all the others can be grabbed from the stream.

ICC_MP4_Multiplexer Interface

The interface to operate with the MP4 multiplexer.

Class Hierarchy



Syntax

```
[object, uuid(4a4fc69f-4a41-4785-97d6-8fa37507769c), pointer_default(unique), local]
interface ICC_MP4_Multiplexer : ICC_Multiplexer;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_Multiplexer Interface

	Name	Description
	CreatePin (see page 145)	Method creates an input pin to add the specified type of elementary data needed to be multipolexed.
	CreatePinByXml (see page 146)	Method creates an input pin to add the specified type of elementary data needed to be multipolexed.
	GetPin (see page 146)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.

	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
	DataSize (see page 64)	

ICC_Multiplexer Interface

	Name	Description
	PinCount (see page 146)	/// Method creates an input pin to add the specified type of elementary data needed to be multipolexed. /// Returns: Returns S_OK if successful or error code otherwise. HRESULT CreatePinByType([in] CC_ELEMENTARY_STREAM_TYPE (see page 160) stream_type, // Pin type [out,retval] ICC_ByteStreamConsumer (see page 12) **pOutput // Address where pointer to the created object will be stored.);

ICC_MP4_Multiplexer Interface

	Name	Description
	Duration (see page 327)	

Properties

Duration

Syntax

```
__property CC_TIME Duration;
```

ICC_MP4_MultiplexerSettings Interface

The interface to operate with MP4 multiplexer.

Class Hierarchy

Syntax

```
[object, uuid(29d573e4-e709-41e8-a73c-70876f01a578), pointer_default(unique), local]
interface ICC_MP4_MultiplexerSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_MP4_MultiplexerSettings Interface

	Name	Description
	DataGranularity (see page 329)	Output Block.
	Format (see page 329)	
	Fragmented (see page 329)	
	InterleavingPeriod (see page 329)	
	UpdatePeriod (see page 329)	Index update interval (default is never - only at close).
	WriteSingleFragment (see page 329)	

Properties

DataGranularity

Output Block.

Syntax

```
__property CC_UINT DataGranularity;
```

Format

Syntax

```
__property ICC_Settings * Format;
```

Fragmented

Syntax

```
__property CC_BOOL Fragmented;
```

InterleavingPeriod

Syntax

```
__property CC_PERIOD InterleavingPeriod;
```

UpdatePeriod

Index update interval (default is never - only at close).

Syntax

```
__property CC_PERIOD UpdatePeriod;
```

WriteSingleFragment

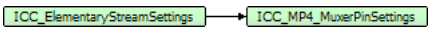
Syntax

```
__property CC_BOOL WriteSingleFragment;
```

ICC_MP4_MuxerPinSettings Interface

The major stream properties for creating the pin in the multiplexer. Only StreamType field is mandatory - all the others can be grabbed from the stream.

Class Hierarchy



Syntax

```
[object, uuid(bc0a4c83-d677-4a69-8514-5be234b14bda), pointer_default(unique), local]
interface ICC_MP4_MuxerPinSettings : ICC_ElementaryStreamSettings;
```

Properties

	Name	Description
	BasePTS (see page 330)	The time stamp of the first stream sample. Default is 0.
	ReferenceURL (see page 330)	The URL of external (reference) file where the stream will be actually stored (see ICC_ReferenceDataConsumer interface)
	SampleRate (see page 331)	The stream sample rate (frequency). In the case of audio, it is a native sample rate. In the case of video it is a FIELD rate.
	SubTypeCode (see page 331)	Any additional type specification (AVC_Intra type, LPCM format, etc)
	TrackTimeScale (see page 331)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

Properties

BasePTS

The time stamp of the first stream sample. Default is 0.

Syntax

```
__property [in] BasePTS;
```

ReferenceURL

The URL of external (reference) file where the stream will be actually stored (see ICC_ReferenceDataConsumer interface)

Syntax

```
__property CC_STRING ReferenceURL;
```

SampleRate

The stream sample rate (frequency). In the case of audio, it is a native sample rate. In the case of video it is a FIELD rate.

Syntax

```
__property [in] SampleRate;
```

SubTypeCode

Any additional type specification (AVC_Intra type, LPCM format, etc)

Syntax

```
__property [in] SubTypeCode;
```

TrackTimeScale

The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

Syntax

```
__property [in] TrackTimeScale;
```

Daniel2 Codec

Daniel2 codec is a new breakthrough compression technology from CINEGY making massive network based post-production and TV branding operations far more efficient.

- World’s fastest production and acquisition codec
- Designed for GPUs for maximum performance
- Architected to be very conservative with GPU memory bandwidth to leave computing resources to simultaneously perform GPU accelerated FX and compositing
- Currently the only way to process professional quality 8K and larger streams on affordable, commodity hardware or even using a consumer-grade laptop

Overview

The Daniel2 codec provides an unparalleled combination of multi-stream, real-time editing performance coupled with impressive image quality at reduced storage rates.

Additionally, the codec preserves frames of any sizes, even 8K and beyond, at full resolution.

As a variable bit rate (VBR) codec technology, Daniel2 uses fewer bits on simple frames that would not benefit from encoding at a higher data rate. Daniel2 is a frame-independent (or “intra-frame”) codec, meaning that each frame is encoded and decoded independently of any other frame. This technique provides the greatest editing performance and flexibility.

Daniel2 codec preserves motion image sequences in 4:2:2 YUV and 4:4:4:4 RGBA color spaces. With a remarkably low data rate (as compared to uncompressed 4:4:4:4), Daniel2 supports from 8 up to 12-bit pixel depth.

Daniel2 quality can be tuned to the customer’s needs, from virtually lossless at 1:5 for high-end production and down to a very good quality at 1:20 compression for massive network operations.

The key points of the Daniel2 technology are:

- Compression comparable with leading industrial production codecs
- Extremely fast both GPU and CPU implementations
- Optimized for parallel architectures (first of all, NVidia)
- Tight integration with the GPU video processing (CPU deals only with compressed video)
- Major color models and chroma sampling at 8..12 bit color
- Native alpha channel support
- Native support of sparse alpha channel
- Wide range of VBR and CBR bitrates
- No-degradation multiple decode-encode cycles is important for complex post production

Daniel2 Features

Daniel2 has all the features required for a professional production codec:

- 4:4:4:4, 4:2:2 and 4:2:0 color spaces native support
- Native alpha channel
- 8 to 12 color bit depth
- Variable and constant bit rate
- Constant quality mode
- Extremely low encoding/decoding latency
- Precise quality/bit rate control
- No degradation on multiple reencodes
- GPU pipeline ready
- CPU and GPU implementation
- RTP ready
- The codec scales effectively on multi-core GPUs and CPUs
- The codec scales effectively on multi-GPU and multi-CPU setups

Input and Output Formats

The Daniel2 coded supports a number of color formats directly on input and output.

Generally, if a format is not supported directly it shall be converted automatically in CPU (i.e. creating additional memory traffic and CPU load).

input/output CCF_format	GPU decoder	CPU decoder	GPU encoder	CPU encoder
CCF_YUY2	+	+	+	+
CCF_UYVY	+	+	+	+
CCF_Y216	+	+	+	+
CCF_V216	+	+	+	+
CCF_V210	+	+	+	+
CCF_RGB30		+		+
CCF_RGBA32	+	+	+	+
CCF_RGBA64	+	+	+	+
CCF_UYVY_32F (Adobe)		+		+
CCF_BGRA16u (Adobe)	+	+	+	+
CCF_YUYV_4x4			+	+
CCF_V210_4x4K			+	+

The GPU Pipeline

In order to achieve the best performance with Daniel2 technology it is necessary to understand the advantages and pecularities of a GPU pipeline.

The major encoding bottleneck in such pipeline is usually the bus between the main memory, the grabber board and the CUDA board. If you want to deliver the decoded frames back to CPU, the bus speed is the major limiting factor.

One must use **pinned memory** (refer to CUDA documentation for details) for all the intermediate buffers in the RAM in order to achieve the expected performance. Use the dedicated Cinecoder API calls to allocate/deallocate the pinned memory for your buffers.

Refer to the Daniel2 GPU code sample for a simple illustration of the approach.

Multi-GPU and Multi-CPU Setups

Daniel2 scales effectively on dual and quad CPU machines.

Daniel2 does support multiple GPU boards in a computer. For certain configurations, a few simpler boards can be more effective than a powerful one.

NB: If the computer uses NUMA, the behavior of a GPU board can be different on the NUMA nodes, this one can detect only by an experiment.

GPU Player-Decoder

Daniel2 has special means to organize an effective players of large frame streams on smaller screens.

The player-decoder produces the RGBA64 format from any sort of input, to allow sending the frame directly to displayable OpenGL/DirectX texture.

In addition, the player-decoder can scale the output 2 and 4 times already during decoding, saving a lot of memory traffic.

Note, the transfer to the RGBA64 texture is a heavy operation, usually slower than even complete frame decoding.

Thus, sending 1:2 or 1:4 frame to a texture saves a lot of bandwidth and GPU clocks comparing for a full size variant.

This is important for smaller machines with smaller screens, like notebooks.

Types

Enumerations

	Name	Description
	CC_D2DEC_SCALE (see page 341)	defines scales as arithmetic shifts for the D2 decoder

Structures

	Name	Description
	CC_D2_DECODER_PARAMS (see page 340)	parameters of the DANIEL2 decoder

CC_D2_DECODER_PARAMS

parameters of the DANIEL2 decoder

Syntax

```
struct CC_D2_DECODER_PARAMS {
  void* gDecodedBuffer;
  CC_INT DecodedPitch;
  CC_INT DecodedHeight;
  CC_D2DEC_SCALE decScale;
  RECT decROI;
};
```

Members

gDecodedBuffer

pointer to the decoded frame buffer

DecodedPitch

the frame pitch in bytes

DecodedHeight

the frame height in pixels

decScale

descale on-the-fly (player only)

decROI

region of interest (player only)

CC_D2DEC_SCALE

defines scales as arithmetic shifts for the D2 decoder

Syntax

```
[v1_enum]
enum CC_D2DEC_SCALE {
  D2DEC_SCALE_1 = 0,
  D2DEC_SCALE_2 = 1,
  D2DEC_SCALE_4 = 2
};
```

Members

D2DEC_SCALE_1

no scale

D2DEC_SCALE_2

1:2 scale

D2DEC_SCALE_4

1:4 scale

Interfaces

Interfaces

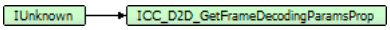
	Name	Description
	ICC_D2D_GetFrameDecodingParamsProp (see page 344)	This interface retrieves the current GPU decoding frame parameters for the player-decoder.
	ICC_D2D_GetUpdateVideoBufferPtrProp (see page 345)	the method swaps the current buffer with the **p parameter and is used for managing GPU pipelines with own GDR queue buffering
	ICC_DanielVideoDecoder (see page 346)	The general interface to access to DANIEL2 video decoder
	ICC_DanielVideoDecoder_CUDA (see page 348)	The general interface to access to DANIEL2 CUDA video decoder
	ICC_DanielVideoDecoder_CudaPlayer (see page 350)	The class is to work with the CUDA player-decoder, which can scale the decoded frames on-the-fly and convert these to RGBA which can be sent to a displayable texture. This mode works only with own buffered queues in the graphic memory. See "Player-decoder chapter" and the corresponding sample.
	ICC_DanielVideoEncoder (see page 354)	The general interface to access to DANIEL2 video encoder
	ICC_DanielVideoEncoder_CUDA (see page 357)	The general interface to access to DANIEL2 CUDA video encoder
	ICC_DanielVideoEncoderSettings (see page 359)	Class of settings to tune the D2 video encoder
	ICC_DanielVideoEncoderSettings_CUDA (see page 365)	Additional settings for the GPU encoder
	ICC_DanielVideoFrameInfo (see page 368)	The object describes the encoded D2 frame
	ICC_DanielVideoSplitter (see page 372)	To perse an input byte-stream into an elementary D2 video stream with separated frames and their descriptions

	ICC_DanielVideoStreamInfo (see page 373)	

ICC_D2D_GetFrameDecodingParamsProp Interface

This interface retrieves the current GPU decoding frame parameters for the player-decoder.

Class Hierarchy



Syntax

```
[object, uuid(d45d4f96-a218-4c54-96fc-230f2c1537a7), pointer_default(unique), local]
interface ICC_D2D_GetFrameDecodingParamsProp : IUnknown;
```

Properties

	Name	Description
R	FrameDecodingParams (see page 344)	Returns the current frame decoding parameters

Properties

FrameDecodingParams

Returns the current frame decoding parameters

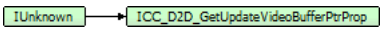
Syntax

```
__property CC_D2_DECODER_PARAMS * FrameDecodingParams;
```

ICC_D2D_GetUpdateVideoBufferPtrProp Interface

the method swaps the current buffer with the **p parameter and is used for managing GPU pipelines with own GDR queue buffering

Class Hierarchy



Syntax

```
[object, uuid(f14dab53-5dc0-4edd-88e6-63d058f278b8), pointer_default(unique), local]
interface ICC_D2D_GetUpdateVideoBufferPtrProp : IUnknown;
```

Methods

	Name	Description
	GetUpdateVideoBufferPtr (see page 345)	

Methods

GetUpdateVideoBufferPtr

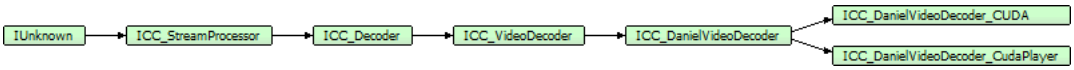
Syntax

```
HRESULT GetUpdateVideoBufferPtr(
    [in,out] void ** p
);
```

ICC_DanielVideoDecoder Interface

The general interface to access to DANIEL2 video decoder

Class Hierarchy



Syntax

```
[object, uuid(2070fe32-e7b4-4063-ac68-86b134f1e8bd), pointer_default(unique), local]
interface ICC_DanielVideoDecoder : ICC_VideoDecoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_VideoDecoder Interface

	Name	Description
	GetFrame (see page 91)	Retrieve the current video frame.
	GetStride (see page 91)	
	GetVideoFrameInfo (see page 91)	Get the description of current video frame.
	GetVideoStreamInfo (see page 92)	Get the description of video stream which is being decoded.

	IsFormatSupported (see page 92)	

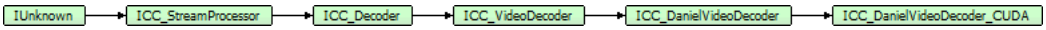
Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_DanielVideoDecoder_CUDA Interface

The general interface to access to DANIEL2 CUDA video decoder

Class Hierarchy



Syntax

```
[object, uuid(60966b59-a6fb-4742-a562-223f7e4c45e3), pointer_default(unique), local]
interface ICC_DanielVideoDecoder_CUDA : ICC_DanielVideoDecoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_VideoDecoder Interface

	Name	Description
	GetFrame (see page 91)	Retrieve the current video frame.
	GetStride (see page 91)	
	GetVideoFrameInfo (see page 91)	Get the description of current video frame.
	GetVideoStreamInfo (see page 92)	Get the description of video stream which is being decoded.

	IsFormatSupported (see page 92)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_DanielVideoDecoder_CUDA Interface

	Name	Description
	DeviceID (see page 349)	The nVidia device ID (actually, sequemtial number of a board)
	TargetColorFormat (see page 349)	The target color format used for decoder's kernel initialization

Properties

DeviceID

The nVidia device ID (actually, sequemtial number of a board)

Syntax

```
__property CC_INT DeviceID;
```

TargetColorFormat

The target color format used for decoder's kernel initialization

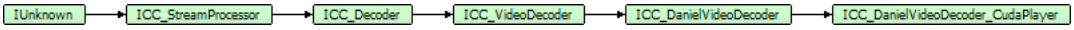
Syntax

```
__property CC_COLOR_FMT TargetColorFormat;
```

ICC_DanielVideoDecoder_CudaPlayer Interface

The class is to work with the CUDA player-decoder, which can scale the decoded frames on-the-fly and convert these to RGBA which can be sent to a displayable texture. This mode works only with own buffered queues in the graphic memory. See "Player-decoder chapter" and the corresponding sample.

Class Hierarchy



Syntax

```
[object, uuid(0add9766-f751-4a15-b891-1495605eef3b), pointer_default(unique), local]
interface ICC_DanielVideoDecoder_CudaPlayer : ICC_DanielVideoDecoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_VideoDecoder Interface

	Name	Description
	GetFrame (see page 91)	Retrieve the current video frame.
	GetStride (see page 91)	
	GetVideoFrameInfo (see page 91)	Get the description of current video frame.

	GetVideoStreamInfo (see page 92)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 92)	

ICC_DanielVideoDecoder_CudaPlayer Interface

	Name	Description
	DecodeFrame (see page 352)	Decode a D2 frame note, the frame format must be the same used in the InitDecoder (see page 352) method
	GetOutputBufferDimensions (see page 352)	Provides recommended buffer dimension in pixels
	InitDecoder (see page 352)	Initializes the decoder using a D2 header as parameter

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_DanielVideoDecoder_CudaPlayer Interface

	Name	Description
	DeviceID (see page 353)	The nVidia device ID (actually, sequential number of a board)

Methods

DecodeFrame

Decode a D2 frame note, the frame format must be the same used in the InitDecoder (see page 352) method

Syntax

```
HRESULT DecodeFrame(
    [in, size_is(cbSize)] CC_PCBYTE pbData,
    [in] CC_UINT cbSize,
    [in] CC_TIME pts,
    [in] CC_D2_DECODER_PARAMS * pDecPar
);
```

Parameters

pbData

Pointer to the buffer containing the D2 coded frame

cbSize

The D2 coded frame size in bytes.

pts

(Optional) presentation time of the first access unit commencing in the data.

pDecPar

The parameters for the decoded data

GetOutputBufferDimensions

Provides recommended buffer dimension in pixels

Syntax

```
HRESULT GetOutputBufferDimensions(
    [in] CC_SIZE src_size,
    [out, retval] CC_SIZE * ptarget_size
);
```

Parameters

src_size

actual buffer's dimensions

ptarget_size

recommended buffer's dimensions

InitDecoder

Initializes the decoder using a D2 header as parameter

Syntax

```
HRESULT InitDecoder(
    [in, size_is(cbSize)] CC_PCBYTE pbData,
    [in] CC_UINT cbSize
);
```

Parameters

pbData

Pointer to the buffer containing the D2 coded data (i.e. first frame, or just header)

cbSize

The data size in bytes.

Properties

DeviceID

The nVidia device ID (actually, sequential number of a board)

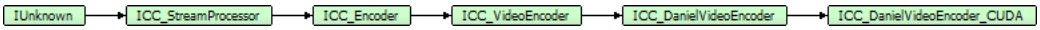
Syntax

```
__property CC_INT DeviceID;
```

ICC_DanielVideoEncoder Interface

The general interface to access to DANIEL2 video encoder

Class Hierarchy



Syntax

```
[object, uuid(ecab2803-01f0-4233-a346-3ab90e197129), pointer_default(unique), local]
interface ICC_DanielVideoEncoder : ICC_VideoEncoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_VideoEncoder Interface

	Name	Description
	AddFrame (see page 94)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 94)	Add a frame with different size to the processing queue.
	GetStride (see page 95)	
	GetVideoFrameInfo (see page 95)	Get the description of current video frame.
	GetVideoStreamInfo (see page 95)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 95)	
	IsScaleAvailable (see page 96)	

ICC_DanielVideoEncoder Interface

	Name	Description
	AddUserData (see page 355)	Adds user data for the subsequent video frame.

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

ICC_DanielVideoEncoder Interface

	Name	Description
W	InitialTimeCode (see page 356)	Specifies the initial timecode for the first frame of the first GOP. Should be used before first call to AddFrame (see page 94).

Methods

AddUserData

Adds user data for the subsequent video frame.

Syntax

```
HRESULT AddUserData(
    [in, size_is(cbSize)] const BYTE * pbUserData,
```

```
    [in] DWORD cbSize,
    [in,defaultvalue(CC_FALSE)] CC_BOOL bSecondField
);
```

Parameters

pbUserData

The user data.

cbSize

The size of the user data, in bytes.

bSecondField

Tells that incoming user data must appear at the second picture_start_code (in case of interlaced coding).

Returns

Returns S_OK if successful or an error code otherwise.

Notes

You may call AddUserData several times to add more than one user data.

Properties

InitialTimeCode

Specifies the initial timecode for the first frame of the first GOP. Should be used before first call to AddFrame (see page 94).

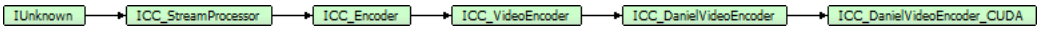
Syntax

```
__property CC_TIMECODE InitialTimeCode;
```

ICC_DanielVideoEncoder_CUDA Interface

The general interface to access to DANIEL2 CUDA video encoder

Class Hierarchy



Syntax

```
[object, uuid(9a156bd1-f05f-4d3c-9b55-c49f65200342), pointer_default(unique), local]
interface ICC_DanielVideoEncoder_CUDA : ICC_DanielVideoEncoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_VideoEncoder Interface

	Name	Description
	AddFrame (see page 94)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 94)	Add a frame with different size to the processing queue.
	GetStride (see page 95)	
	GetVideoFrameInfo (see page 95)	Get the description of current video frame.
	GetVideoStreamInfo (see page 95)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 95)	
	IsScaleAvailable (see page 96)	

ICC_DanielVideoEncoder Interface

	Name	Description
	AddUserData (see page 355)	Adds user data for the subsequent video frame.

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_Encoder Interface

	Name	Description
R	DataSize (see page 64)	

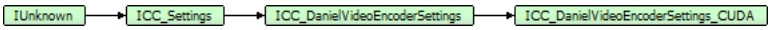
ICC_DanielVideoEncoder Interface

	Name	Description
W	InitialTimeCode (see page 356)	Specifies the initial timecode for the first frame of the first GOP. Should be used before first call to AddFrame (see page 94).

ICC_DanielVideoEncoderSettings Interface

Class of settings to tune the D2 video encoder

Class Hierarchy



Syntax

```
[object, uuid(5f03f830-497f-471a-bcc1-d5cfb417f544), pointer_default(unique), local]
interface ICC_DanielVideoEncoderSettings : ICC_Settings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_DanielVideoEncoderSettings Interface

	Name	Description
	AddUserData (see page 361)	Adds another user data to the Sequence Layer of the MPEG video stream.
	GetUserData (see page 361)	Retrieves the specified user data which associated with the stream (seq_hdr level).

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_DanielVideoEncoderSettings Interface

	Name	Description
	AlphaQuantScaleAdd (see page 362)	(0-255)
	AspectRatio (see page 362)	The aspect ratio of a frame W:H.

	BitDepth (see page 362)	The sample bit depth (8-12)
	BitRate (see page 362)	The Max bitrate in CBR mode or max bitrate in VBR mode.
	ChromaFormat (see page 362)	The chroma resolution format
	ChromaQuantScaleAdd (see page 362)	(0-255)
	CodingMethod (see page 363)	The coding method
	ColorCoefs (see page 363)	The color transformarion description.
	FrameRate (see page 363)	The frame rate of video stream.
	FrameSize (see page 363)	The physical frame size, in pixels.
	GOP (see page 363)	The GOP settings. See CC_GOP_DESCR (see page 110) for details.
	InitialTimeCode (see page 363)	Specifies the initial timecode for the first frame of the first GOP (see page 363).
	InputColorFormat (see page 363)	The actual input color format (requred by the GPU encoder)
	InterlaceType (see page 364)	The field order video.
	NumSingleEncoders (see page 364)	Number of single encoders to be used.
	PictureOrientation (see page 364)	The orientation of frames in the stream.
	QuantScale (see page 364)	The quality scale factor (0 or 1..1000)
	RateMode (see page 364)	The mode of bitrate control - see CC_BITRATE_MODE
R	UserDataCount (see page 364)	The number of MPEG_USER_DATA, associated with the stream.
	VideoFormat (see page 364)	The video format.

Methods

AddUserData

Adds another user data to the Sequence Layer of the MPEG video stream.

Syntax

```
HRESULT AddUserData(
    [in,size_is(cbSize)] const BYTE * pbUserData,
    [in] DWORD cbSize
);
```

Parameters

pbUserData

The user data.

cbSize

The user data size.

Returns

Returns S_OK if successful or an error value otherwise.

GetUserData

Retrieves the specified user data which associated with the stream (seq_hdr level).

Syntax

```
HRESULT GetUserData(
    [in] DWORD dwUserDataNumber,
    [out, size_is(cbBufSize)] BYTE * pData,
    [in] DWORD cbBufSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

dwUserDataNumber

Specified the user data number, zero-based.

pData

Place to store the user data, if NULL the only size of the specified user data will be returned.

cbBufSize

Buffer size.

pcbRetSize

Place to store the user data size.

Returns

Returns S_OK if successful or E_INVALIDARG in case of incorrect *dwUserDataNumber*.

Properties

AlphaQuantScaleAdd

(0-255)

Syntax

```
__property CC_INT AlphaQuantScaleAdd;
```

AspectRatio

The aspect ratio of a frame W:H.

Syntax

```
__property CC_RATIONAL AspectRatio;
```

BitDepth

The sample bit depth (8-12)

Syntax

```
__property CC_UINT BitDepth;
```

BitRate

The Max bitrate in CBR mode or max bitrate in VBR mode.

Syntax

```
__property CC_BITRATE BitRate;
```

ChromaFormat

The chroma resolution format

Syntax

```
__property CC_CHROMA_FORMAT ChromaFormat;
```

ChromaQuantScaleAdd

(0-255)

Syntax

```
__property CC_INT ChromaQuantScaleAdd;
```

CodingMethod

The coding method

Syntax

```
__property CC_DANIEL2_CODING_METHOD CodingMethod;
```

ColorCoefs

The color transformarion description.

Syntax

```
__property CC_COLOUR_DESCRIPTION ColorCoefs;
```

FrameRate

The frame rate of video stream.

Syntax

```
__property CC_FRAME_RATE FrameRate;
```

FrameSize

The physical frame size, in pixels.

Syntax

```
__property CC_SIZE FrameSize;
```

GOP

The GOP settings. See CC_GOP_DESCR (see page 110) for details.

Syntax

```
__property CC_GOP_DESCR GOP;
```

InitialTimeCode

Specifies the initial timecode for the first frame of the first GOP (see page 363).

Syntax

```
__property CC_TIMECODE InitialTimeCode;
```

InputColorFormat

The actual input color format (requred by the GPU encoder)

Syntax

```
__property CC_COLOR_FMT InputColorFormat;
```

InterlaceType

The field order video.

Syntax

```
__property CC_INTERLACE_TYPE InterlaceType;
```

NumSingleEncoders

Number of single encoders to be used.

Syntax

```
__property CC_UINT NumSingleEncoders;
```

PictureOrientation

The orientation of frames in the stream.

Syntax

```
__property CC_PICTURE_ORIENTATION PictureOrientation;
```

QuantScale

The quality scale factor (0 or 1..1000)

Syntax

```
__property CC_FLOAT QuantScale;
```

RateMode

The mode of bitrate control - see CC_BITRATE_MODE

Syntax

```
__property CC_BITRATE_MODE RateMode;
```

UserDataCount

The number of MPEG_USER_DATA, associated with the stream.

Syntax

```
__property CC_UINT* UserDataCount;
```

VideoFormat

The video format.

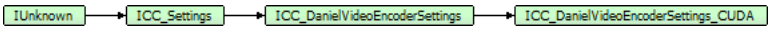
Syntax

```
__property CC_VIDEO_FORMAT VideoFormat;
```

ICC_DanielVideoEncoderSettings_CUDA Interface

Additional settings for the GPU encoder

Class Hierarchy



Syntax

```
[object, uuid(64310105-efcc-4027-9c46-acba0014c703), pointer_default(unique), local]
interface ICC_DanielVideoEncoderSettings_CUDA : ICC_DanielVideoEncoderSettings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

ICC_DanielVideoEncoderSettings Interface

	Name	Description
	AddUserData (see page 361)	Adds another user data to the Sequence Layer of the MPEG video stream.
	GetUserData (see page 361)	Retrieves the specified user data which associated with the stream (seq_hdr level).

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_DanielVideoEncoderSettings Interface

	Name	Description
	AlphaQuantScaleAdd (see page 362)	(0-255)
	AspectRatio (see page 362)	The aspect ratio of a frame W:H.

	BitDepth (see page 362)	The sample bit depth (8-12)
	BitRate (see page 362)	The Max bitrate in CBR mode or max bitrate in VBR mode.
	ChromaFormat (see page 362)	The chroma resolution format
	ChromaQuantScaleAdd (see page 362)	(0-255)
	CodingMethod (see page 363)	The coding method
	ColorCoefs (see page 363)	The color transformarion description.
	FrameRate (see page 363)	The frame rate of video stream.
	FrameSize (see page 363)	The physical frame size, in pixels.
	GOP (see page 363)	The GOP settings. See CC_GOP_DESCR (see page 110) for details.
	InitialTimeCode (see page 363)	Specifies the initial timecode for the first frame of the first GOP (see page 363).
	InputColorFormat (see page 363)	The actual input color format (requred by the GPU encoder)
	InterlaceType (see page 364)	The field order video.
	NumSingleEncoders (see page 364)	Number of single encoders to be used.
	PictureOrientation (see page 364)	The orientation of frames in the stream.
	QuantScale (see page 364)	The quality scale factor (0 or 1..1000)
	RateMode (see page 364)	The mode of bitrate control - see CC_BITRATE_MODE
R	UserDataCount (see page 364)	The number of MPEG_USER_DATA, associated with the stream.
	VideoFormat (see page 364)	The video format.

ICC_DanielVideoEncoderSettings_CUDA Interface

	Name	Description
	DeviceID (see page 367)	The nVidia device ID (actually, sequemtial number of a board)

Properties

DeviceID

The nVidia device ID (actually, sequemtial number of a board)

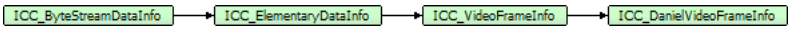
Syntax

```
__property [in] DeviceID;
```

ICC_DanielVideoFrameInfo Interface

The object describes the encoded D2 frame

Class Hierarchy



Syntax

```
[object, uuid(5e60a260-ffc6-4325-b6b1-f743aa1046b5), pointer_default(unique), local]
interface ICC_DanielVideoFrameInfo : ICC_VideoFrameInfo;
```

Methods

ICC_DanielVideoFrameInfo Interface

	Name	Description
	GetUserData (see page 369)	Retrieves the specified user data which belongs to the video frame.

Properties

	Name	Description
R	DTS (see page 49)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 49), except the coded video with B-frames.
R	Duration (see page 49)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
R	NumSamples (see page 49)	Number of samples of the elementary data. In case of coded video frames, there is usually 1 or 2 sample(s) (1 frame or 2 fields). In case of coded audio, there are the number of audio samples, encoded into audio frame.
R	PresentationDelta (see page 49)	The presentation delta, in samples, of the elementary data. It is actual for data which was reordered during encoding process (f.e. coded video with B-frames).
R	PTS (see page 49)	The Presentation Data Stamp (PTS) of the elementary data. The PTS based on CC_TIMEBASE, specified for object which generates the elementary data.
R	SampleOffset (see page 49)	The frame's first sample order number.

	SequenceEntryFlag (see page 50)	The sequence entry point flag. In the case of mpeg video, it means that SEQUENCE_HEADER presents in the elementary data. This flag is used to signal the multiplexers to generate the entry point (like System Header) at this elementary data.

ICC_VideoFrameInfo Interface

	Name	Description
	CodingNumber (see page 97)	The number of video frame in the coding order. Zero-based.
	Flags (see page 97)	Various flags of the coded video frame. Value of this field dependend of the video stream type.
	FrameType (see page 97)	The frame coding type (see page 107).
	InterlaceType (see page 98)	The field order of video frame.
	Number (see page 98)	The number of video frame in native (display) order. zero-based.
	PictStruct (see page 98)	The picture structure of the MPEG video frame.
	TimeCode (see page 98)	The timecode of video frame.

ICC_DanielVideoFrameInfo Interface

	Name	Description
	CodingMethod (see page 370)	The coding method
	PictureOrientation (see page 370)	The orientation of frame
	QuantScale (see page 370)	the quality factor (0, 1..1000)
	UserDataCount (see page 370)	The number of MPEG_USER_DATA, associated with the video frame.

Methods

GetUserData

Retrieves the specified user data which belongs to the video frame.

Syntax

```
HRESULT GetUserData(
    [in] DWORD dwUserDataNumber,
```

```
    [out, size_is(cbBufSize)] BYTE * pData,
    [in] DWORD cbBufSize,
    [out,retval] DWORD * pcbRetSize
);
```

Parameters

dwUserDataNumber

Specified the user data number, zero-based.

pData

Place to store the user data, if NULL the only size of the specified user data will be returned.

cbBufSize

Buffer size.

pcbRetSize

Place to store the user data size.

Returns

Returns S_OK if successful or E_INVALIDARG in case of incorrect *dwUserDataNumber*.

Properties

CodingMethod

The coding method

Syntax

```
__property CC_DANIEL2_CODING_METHOD* CodingMethod;
```

PictureOrientation

The orientation of frame

Syntax

```
__property CC_PICTURE_ORIENTATION* PictureOrientation;
```

QuantScale

the quality factor (0, 1..1000)

Syntax

```
__property CC_FLOAT * QuantScale;
```

UserDataCount

The number of MPEG_USER_DATA, associated with the video frame.

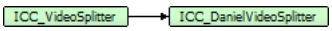
Syntax

```
__property DWORD * UserDataCount;
```

ICC_DanielVideoSplitter Interface

To perse an input byte-stream into an elementary D2 video stream with separated frames and their descriptions

Class Hierarchy



Syntax

```
[object, uuid(25f3f204-1537-4c7b-97bd-a5eb76788509), pointer_default(unique), local]
interface ICC_DanielVideoSplitter : ICC_VideoSplitter;
```

ICC_DanielVideoStreamInfo Interface

Class Hierarchy

Syntax

```
[object, uuid(fd99aee9-6b26-43ec-bbd1-ff658ef6d864), pointer_default(unique), local]
interface ICC_DanielVideoStreamInfo : ICC_VideoStreamInfo;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ElementaryStreamInfo Interface

	Name	Description
R	BitRate (see page 52)	The bitrate (max) of the elementary stream.
R	FrameRate (see page 52)	The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.
R	StreamType (see page 52)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

ICC_VideoStreamInfo Interface

	Name	Description
R	AspectRatio (see page 100)	The aspect ratio cx:cy.

	FrameSize (see page 100)	The size in pixels of video frame.
	ProgressiveSequence (see page 100)	If TRUE - all frames in the stream coded without fields (progressive).

ICC_DanielVideoStreamInfo Interface

	Name	Description
	AlphaQuantScaleAdd (see page 374)	
	BitDepth (see page 374)	The sample bit depth (8-12)
	ChromaFormat (see page 375)	The chroma resolution format
	ChromaQuantScaleAdd (see page 375)	
	CodingMethod (see page 375)	The coding method
	ColorCoefs (see page 375)	The color transformarion description.
	PictureOrientation (see page 375)	The orientation of frame
	QuantScale (see page 375)	
	VideoFormat (see page 375)	The video format.

Properties

AlphaQuantScaleAdd

Syntax

```
__property CC_INT * AlphaQuantScaleAdd;
```

BitDepth

The sample bit depth (8-12)

Syntax

```
__property DWORD* BitDepth;
```

ChromaFormat

The chroma resolution format

Syntax

```
__property CC_CHROMA_FORMAT* ChromaFormat;
```

ChromaQuantScaleAdd

Syntax

```
__property CC_INT * ChromaQuantScaleAdd;
```

CodingMethod

The coding method

Syntax

```
__property CC_DANIEL2_CODING_METHOD * CodingMethod;
```

ColorCoefs

The color transformarion description.

Syntax

```
__property CC_COLOUR_DESCRIPTION * ColorCoefs;
```

PictureOrientation

The orientation of frame

Syntax

```
__property CC_PICTURE_ORIENTATION * PictureOrientation;
```

QuantScale

Syntax

```
__property CC_FLOAT * QuantScale;
```

VideoFormat

The video format.

Syntax

```
__property CC_VIDEO_FORMAT* VideoFormat;
```

Additional Codecs

AES-3 (SMPTE-302M) Audio codec

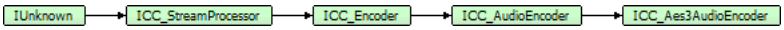
Interfaces

	Name	Description
	ICC_Aes3AudioEncoder (see page 379)	The main interface to access the CC_Aes3AudioEncoder class.
	ICC_Aes3AudioEncoderSettings (see page 381)	The settings for CC_Aes3AudioEncoder initialization.
	ICC_Aes3AudioStreamInfo (see page 383)	Represents the AES3-specified video stream description.
	ICC_Aes3AudioDecoder (see page 385)	The main interface to access the CC_Aes3AudioDecoder class.

ICC_Aes3AudioEncoder Interface

The main interface to access the CC_Aes3AudioEncoder class.

Class Hierarchy



Syntax

```
[object, uuid(17a5ca65-d735-4e22-a2f9-2971cee7e81e), pointer_default(unique), local]
interface ICC_Aes3AudioEncoder : ICC_AudioEncoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Encoder Interface

	Name	Description
	GetData (see page 64)	The real number of bytes will be placed here

ICC_AudioEncoder Interface

	Name	Description
	GetAudioFrameInfo (see page 129)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 129)	Get the description of audio stream which is being decoded.
	ProcessAudio (see page 130)	Puts another audio uncompressed data to the audio consumer.

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.

	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

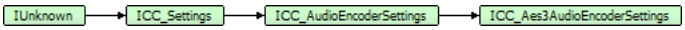
ICC_Encoder Interface

	Name	Description
	DataSize (see page 64)	

ICC_Aes3AudioEncoderSettings Interface

The settings for CC_Aes3AudioEncoder initialization.

Class Hierarchy



Syntax

```
[object, uuid(0aac0e66-5eb9-4f8e-9888-8dc511ffa669), pointer_default(unique), local]
interface ICC_Aes3AudioEncoderSettings : ICC_AudioEncoderSettings;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_AudioEncoderSettings Interface

	Name	Description
	BitRate (see page 135)	The bitrate of entire audio bitstream or corresponding audio frame.
	BitsPerSample (see page 135)	The bit depth of one audio sample (of one channel).
	FrameRate (see page 135)	The number of audio frames per second.
	NumChannels (see page 135)	The number of channels in the waveform-audio data.
	SampleRate (see page 135)	The audio sampling frequency.

ICC_Aes3AudioEncoderSettings Interface

	Name	Description
	BitsPerSample (see page 382)	The quantization of the audio data word. Possible values are 16, 20 or 24.
	ChannelID (see page 382)	The channel number of first data channel - see SMPTE 302M subclause 5.6 for details.
	FrameRate (see page 382)	The number of audio frames per second.
	NumChannels (see page 382)	The number of audio channels.

Properties

BitsPerSample

The quantization of the audio data word. Possible values are 16, 20 or 24.

Syntax

```
__property CC_UINT BitsPerSample;
```

ChannelID

The channel number of first data channel - see SMPTE 302M subclause 5.6 for details.

Syntax

```
__property CC_BYTE ChannelID;
```

FrameRate

The number of audio frames per second.

Syntax

```
__property CC_FRAME_RATE FrameRate;
```

NumChannels

The number of audio channels.

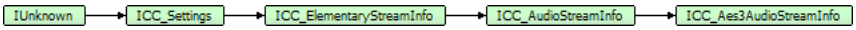
Syntax

```
__property CC_UINT NumChannels;
```

ICC_Aes3AudioStreamInfo Interface

Represents the AES3-specified video stream description.

Class Hierarchy



Syntax

```
[object, uuid(6a7d55ae-19f4-46e8-87c9-27d95712e966), pointer_default(unique), local]
interface ICC_Aes3AudioStreamInfo : ICC_AudioStreamInfo;
```

Methods

	Name	Description
	Assigned (see page 73)	Tests the specified value (or any of values if specified "*") assigned.
	Clear (see page 73)	Marks the specified value (or all values if specified "*") as not assigned.

Properties

	Name	Description
	XML (see page 73)	Exports the contents of the settings into the string XML format (only values which were actually assigned). Imports the string XML and assigns the variables listed there. No assigned variables cleared so the result will be the intersection between old state and the variables assigned by XML.

ICC_ElementaryStreamInfo Interface

	Name	Description
R	BitRate (see page 52)	The bitrate (max) of the elementary stream.
R	FrameRate (see page 52)	The frame rate of the stream. In the case of video, it is native video frame rate. In the case of audio, it is rate of the coded audio frames.
R	StreamType (see page 52)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 160) for details.

ICC_AudioStreamInfo Interface

	Name	Description
	BitsPerSample (see page 133)	The bit depth of one audio sample (of one channel).
	ChannelMask (see page 133)	The mask of channels. See the CC_AUDIO_CHANNEL_MASK for details
	NumChannels (see page 133)	The number of channels in the waveform-audio data.
	SampleRate (see page 133)	The audio sampling frequency.

ICC_Aes3AudioStreamInfo Interface

	Name	Description
	ChannelID (see page 384)	The channel number of first data channel - see SMPTE 302M subclause 5.6 for details.

Properties

ChannelID

The channel number of first data channel - see SMPTE 302M subclause 5.6 for details.

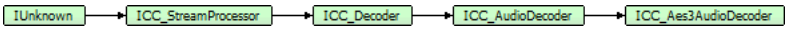
Syntax

```
__property CC_BYTE * ChannelID;
```

ICC_Aes3AudioDecoder Interface

The main interface to access the CC_Aes3AudioDecoder class.

Class Hierarchy



Syntax

```
[object, uuid(9d054d64-c980-4108-918c-3496373314fc), pointer_default(unique), local]
interface ICC_Aes3AudioDecoder : ICC_AudioDecoder;
```

Methods

	Name	Description
	Done (see page 18)	Stops the current processing.
	Init (see page 19)	Initializes the stream processor with the specified parameters. After Init, object state becomes active. If the object was active before, Init() will cancel the current process and initialize the object for a new one.
	InitByXml (see page 19)	Initialization of the object by XML profile.

ICC_Decoder Interface

	Name	Description
	Break (see page 61)	Breaks the continuity of current processing, flushes the buffers if necessary and continues as after Init().
	ProcessData (see page 61)	Process the chunk of data. The method handles bytestream chunks from various bytestream producers like encoders, multiplexers etc. In case of assigned buffer allocator (see page 42) check that *pbData* points to the internal buffer. If so, you may not copy the data by yourself.

ICC_AudioDecoder Interface

	Name	Description
	GetAudio (see page 125)	Retrieves the uncompressed audio data from the audio producer. Remark: To obtain the size of the buffer to hold the resulted samples, put NULL instead of pbData.
	GetAudioFrameInfo (see page 126)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 126)	Get the description of the audio stream.

	GetSampleBytes (see page 126)	
	IsFormatSupported (see page 127)	

Properties

	Name	Description
R	BitRate (see page 19)	The bitrate of the generated or processed bytestream
R	DataInfo (see page 19)	The description of the currently generated data. If NULL - data is not ready.
R	IsActive (see page 20)	The object's state.
R	IsDataReady (see page 20)	Indicates that object has prepared data.
	OutputCallback (see page 20)	The consumer for the data generated by the object.
R	StreamInfo (see page 20)	The description of the stream which is currently being processed. If NULL - object is not active.
	TimeBase (see page 20)	The time base for all CC_TIME values, relating to any time stamps (PTS or DTS) of data, handled by stream processor.

ICC_AudioDecoder Interface

	Name	Description
R	NumSamples (see page 127)	The number of ready audio samples at the output.

Samples

H.264 Video Decoder Sample

C++

```
// SampleMpegVideoDecoder.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include "Cinecoder/Cinecoder_h.h"
#include "Cinecoder/Cinecoder_i.c"
#include "Cinecoder/comapi.h"

#include <memory>

//---------------------------------------------------------------------
HRESULT print_error(HRESULT hr)
//---------------------------------------------------------------------
{
    wchar_t buf[1024] = {0};

    FormatMessageW(FORMAT_MESSAGE_FROM_SYSTEM, 0, hr, 0, buf, sizeof(buf), 0);
    wprintf(L"error %08x: %s", hr, buf);

    return hr;
}

//---------------------------------------------------------------------
class C_PPMWriter : public C_Unknown, public ICC_DataReadyCallback
//---------------------------------------------------------------------
{
  BYTE *m_pBuffer;
  DWORD m_cbFrameBytes;

public:
  C_PPMWriter() : m_pBuffer(NULL), m_cbFrameBytes(0)
  {
  }

  virtual ~C_PPMWriter()
  {
    if(m_pBuffer)
      delete [] m_pBuffer;
  }

  _IMPLEMENT_IUNKNOWN_1(ICC_DataReadyCallback);

  STDMETHOD(DataReady)(IUnknown *pUnk)
  {
    HRESULT hr = S_OK;

    C_ComPtr<ICC_VideoProducer> spProducer;
    if(FAILED(hr = pUnk->QueryInterface(IID_ICC_VideoProducer, (void**)&spProducer)))
      return hr;

    C_ComPtr<ICC_VideoStreamInfo> spVideoInfo;
    if(FAILED(hr = spProducer->GetStreamInfo(&spVideoInfo)))
      return hr;
```

```
    SIZE szFrame;
    if(spVideoInfo->get_FrameSize(&szFrame) != S_OK)
      return E_UNEXPECTED;

    if(m_pBuffer == NULL)
    {
      printf("Frame size = %d x %d, Frame rate = ", szFrame.cx, szFrame.cy);

      CC_FRAME_RATE rFrameRate;
      if(S_OK == spVideoInfo->get_FrameRate(&rFrameRate))
          printf("%g\n", double(rFrameRate.num) / rFrameRate.denom);
      else
          printf("<unknown>\n");

      if(FAILED(hr = spProducer->GetFrame(CCF_BGR24, NULL, 0, szFrame.cx*3, &m_cbFrameBytes)))
        return hr;

      if(NULL == (m_pBuffer = new BYTE[m_cbFrameBytes]))
        return E_OUTOFMEMORY;
    }

    C_ComPtr<ICC_VideoFrameInfo> spFrame;
      if(FAILED(hr = spProducer->GetFrameInfo(&spFrame)))
        return hr;

    DWORD dwFrameNumber = 0;
    if(hr = spFrame->get_Number(&dwFrameNumber))
      return hr;

      wchar_t filename[128];
      wsprintf(filename, L"test%05d.ppm", dwFrameNumber);
      wprintf(L"%s:", filename);

    HANDLE hFile = ::CreateFileW(filename, GENERIC_WRITE, 0, NULL,
                                 CREATE_ALWAYS, 0, NULL);

    if(hFile == INVALID_HANDLE_VALUE)
      return HRESULT_FROM_WIN32(::GetLastError());

      /* PPM header */
      char hdr[128];
    sprintf(hdr,"P6\n%d %d\n255\n",szFrame.cx, szFrame.cy);

    DWORD dwBytesWrote = 0;

    if(!::WriteFile(hFile, hdr, DWORD(strlen(hdr)), &dwBytesWrote, NULL))
      return HRESULT_FROM_WIN32(::GetLastError());

    if(FAILED(hr = spProducer->GetFrame(CCF_BGR24, m_pBuffer, m_cbFrameBytes,
                                        szFrame.cx * 3, &dwBytesWrote)))
      return hr;

    if(!::WriteFile(hFile, m_pBuffer, m_cbFrameBytes, &dwBytesWrote, NULL))
      return HRESULT_FROM_WIN32(::GetLastError());

    CloseHandle(hFile);

    printf("Ok\n");

    return S_OK;
  }
};

//------------------------------------------------------------------
int _tmain(int argc, _TCHAR* argv[])
//------------------------------------------------------------------
{
```

```
    puts("\n* * * The Cinegy(R) Cinecoder Video Decoder sample * * *");

    if(argc < 2)
    {
      puts("Usage: test <input_file>");
      return -1;
    }

    HRESULT hr = S_OK;

    // Ppening the input file ---------------------------------------
    wprintf(L"Opening input file '%s': ", argv[1]);

    HANDLE hSrcFile = ::CreateFileW(argv[1], GENERIC_READ, FILE_SHARE_READ,
                                    NULL, OPEN_EXISTING, 0, NULL);

    if(hSrcFile == INVALID_HANDLE_VALUE)
    {
      print_error(::GetLastError());
      return -2;
    }
    puts("ok");

    // Initializing the decoder -------------------------------------
    printf("Initializing decoder: ");

    C_ComPtr<ICC_ClassFactory> spFactory;
    C_ComPtr<ICC_MpegVideoDecoder> spVideoDec;

    do
    {
      if(FAILED(hr = Cinecoder_CreateClassFactory(&spFactory)))
        break;

      spFactory->AssignLicense("TEST",
"6MZM1AFMUB0TXPY9F5JKXSWZPU3N3U9JA0756X33FW3RLKNSKTFU7JK25ENYPZ1S");

      //if(FAILED(hr = spFactory->CreateInstance(CLSID_CC_MpegVideoDecoder,
IID_ICC_MpegVideoDecoder, (IUnknown**)&spVideoDec)))
      if(FAILED(hr = spFactory->CreateInstance(CLSID_CC_H264VideoDecoder, IID_ICC_VideoDecoder,
(IUnknown**)&spVideoDec)))
        break;

      if(FAILED(hr = spVideoDec->put_OutputCallback(static_cast<ICC_DataReadyCallback*>(new
C_PPMWriter()))))
        break;
    }
    while(false);

    if(FAILED(hr)) return print_error(hr);

    puts("ok");

    // 4. Main cycle ------------------------------------------------
    for(;;)
    {
      static BYTE buffer[65536];
      DWORD dwBytesRead = 0, dwBytesProcessed = 0;

      if(!ReadFile(hSrcFile, buffer, sizeof(buffer), &dwBytesRead, NULL))
      {
        hr = HRESULT_FROM_WIN32(::GetLastError());
        break;
      }

      if(dwBytesRead != 0)
      {
        if(FAILED(hr = spVideoDec->ProcessData(buffer, dwBytesRead, 0, -1, &dwBytesProcessed)))
```

```
          break;

        assert(dwBytesProcessed == dwBytesRead);
      }

      if(dwBytesRead != sizeof(buffer))
      {
        hr = spVideoDec->Done(CC_TRUE);
        break;
      }
    }

    if(FAILED(hr)) return print_error(hr);

    return 0;
}
```